Файл взят с сайта - http://www.natahaus.ru/

где есть ещё множество интересных и редких книг,
программ и прочих вещей.
Данный файл представлен исключительно в
ознакомительных целях.
Уважаемый читатель!
Если вы скопируете его,
Вы должны незамедлительно удалить его
 сразу после ознакомления с содержанием.
Копируя и сохраняя его Вы принимаете на себя всю
ответственность, согласно действующему
международному законодательству .
Все авторские права на данный файл
сохраняются за правообладателем.
Любое коммерческое и иное использование
кроме предварительного ознакомления запрещено.
Публикация данного документа не преследует за
собой никакой коммерческой выгоды. Но такие документы
способствуют быстрейшему профессиональному и
духовному росту читателей и являются рекламой
бумажных изданий таких документов.

Все авторские права сохраняются за правообладателем.
Если Вы являетесь автором данного документа и хотите
дополнить его или изменить, уточнить реквизиты автора
или опубликовать другие документы, пожалуйста,
свяжитесь с нами по e-mail - мы будем рады услышать ваши
пожелания.

Лаура Томсон и Люк Веллинг

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ НА PHP и MySQL

ИЗДАНИЕ ВТОРОЕ
исправленное

торгово-издательский дом
DiaSoft

Москва • Санкт-Петербург • Киев
2003

PHP and MySQL™ Web Development

Luke Welling and Laura Thomson

SAMS

201 West 103rd Street, Indianapolis, Indiana 46290

Разработка Web-приложений на PHP и MySQL

Лаура Томсон и Люк Веллинг

торгово-издательский дом DiaSoft

Москва • Санкт-Петербург • Киев

2003

ББК 32.973.2
УДК 681.3.06(075)
Т 77

Томсон Лаура

Т 77 Разработка Web-приложений на PHP и MySQL: Пер. с англ./Лаура Томсон, Люк Веллинг. — 2-е изд., испр. — СПб: ООО «ДиаСофтЮП», 2003. — 672 с.

ISBN 5-93772-090-3

Книга *Разработка Web-приложений на PHP и MySQL* представляет собой всеобъемлющее руководство по совместному применению PHP и MySQL для разработки высокоэффективных и интерактивных Web-сайтов с динамическим содержимым.

Несомненным достоинством книги является ее ориентация на решение реальных бизнес-задач, что воплощено во множестве типовых примеров, столь часто встречающихся при повседневной разработке. Среди этих примеров создание покупательской тележки для электронных магазинов, аутентификация пользователей, генерация динамических PDF-документов, разработка систем электронной почты через Web, написание систем поддержки Web-форумов. Помимо пошагового анализа реальных бизнес-примеров, в книге широко рассматривается формальный синтаксис и семантика языка PHP, основы построения приложений баз данных и особенности применения объектно-ориентированной методологии при разработке приложений для Web.

Сопровождающий книгу CD-ROM содержит тексты всех примеров, рассмотренных в книге, а также множество материала, который окажется полезным для каждого разработчика Web-приложений.

Несмотря на то что книга, в основном, рассчитана на профессиональных программистов и разработчиков, она принесет несомненную пользу и в качестве учебника для начинающих, кто совершает только первые шаги в бесконечном мире создания приложений для Internet.

ББК 32.973.2

Научное издание

Лаура Томсон, Люк Веллинг

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ НА PHP И MYSQL

Заведующий редакцией *С.Н.Козлов*

Научный редактор *Ю.Н.Артеменко*

Верстка *Т.Н.Артеменко*

Главный дизайнер *О.А.Шадрин*

Н/К

ООО «ДиаСофтЮП», 196105, Санкт-Петербург, пр. Ю.Гагарина, д. 1, ком. 108.
Лицензия №000328 от 9 декабря 1999 г.
Сдано в набор 10.03.2003. Подписано в печать 16.04.2003. Формат 70×100/16.
Бумага типографская. Гарнитура Тайме. Печать офсетная. Печ.л. 42.
Доп. тираж 3000 экз. Заказ № 216

Отпечатано с готовых диапозитивов в ФГУП ордена Трудового Красного Знамени «Техническая книга» Министерства Российской Федерации по делам печати, телерадиовещания и средств массовых коммуникаций
198005, Санкт-Петербург, Измайловский пр., 29.

Authorized translation from the English language edition, entitled «PHP and MySQL Web Development», published by Sams, Copyright © 2000

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher,

Russian language edition published by DiaSoft Publishing.
Copyright © 2003

Лицензия предоставлена издательством Sams Corporation, подразделение Macmillan Computer Publishing.
Все права зарезервированы, включая право на полное или частичное воспроизведение в какой бы то ни было форме.

ISBN 5-93772-090-3 (рус.)
ISBN 0-672-31784-2 (англ.)

© Перевод на русский язык. ООО «ДиаСофтЮП», 2003
© Sams Corporation, 2000
© Оформление. ООО «ДиаСофтЮП», 2003

Гигиеническое заключение № 77.99.6.953.П.438.2.99 от 04.02.1999

Краткое оглавление

Часть 1. Использование PHP **25**
Глава 1. Краткий обзор PHP 26
Глава 2. Хранение и получение данных 59
Глава 3. Использование массивов 75
Глава 4. Манипулирование строками и регулярные выражения 95
Глава 5. Повторное использование кода и создание функций 115
Глава 6. Объектно-ориентированное программирование на PHP 139

Часть 2. Использование MySQL **157**
Глава 7. Проектирование Web-баз данных 158
Глава 8. Создание Web-базы данных 168
Глава 9. Работа с базой данных MySQL 187
Глава 10. Доступ к базе данных MySQL из Web с помощью PHP 203
Глава 11. Дополнительные возможности MySQL 216

Часть 3. Системы электронной торговли и безопасность .. **232**
Глава 12. Эксплуатация сайта электронной коммерции 233
Глава 13. Вопросы безопасности в электронной коммерции 245
Глава 14. Аутентификация с помощью PHP и MySQL 265
Глава 15. Реализация безопасных транзакций в PHP и MySQL 283

Часть 4. Усложненные технологии применения PHP **301**
Глава 16. Взаимодействие с файловой системой и сервером 302
Глава 17. Использование функций работы с сетью и протоколами .. 316
Глава 18. Управление датой и временем 333
Глава 19. Создание изображений 340
Глава 20. Управление сеансами в PHP 361
Глава 21. Другие полезные свойства PHP 374

Часть 5. Разработка практических приложений на PHP и MySQL 380

Глава 22. Применение PHP и MySQL при разработке крупных проектов 381

Глава 23. Отладка 396

Глава 24. Аутентификация и персонализация пользователей 411

Глава 25. Создание покупательской тележки 441

Глава 26. Построение системы управления содержимым 477

Глава 27. Построение почтовой службы, основанной на Web 500

Глава 28. Создание менеджера списков рассылки 529

Глава 29. Создание Web-форумов 571

Глава 30. Генерация персонифицированных документов в формате переносимых документов (PDF) 596

Часть 6. Приложения 626

Приложение А. Инсталляция PHP 4 и MySQL 627

Приложение В. Ресурсы Internet 645

Предметный указатель 648

Оглавление

Часть 1. Использование PHP 25

Глава 1. Краткий обзор PHP 26

Использование PHP 27
Пример приложения: Bob's Auto Parts 27
Форма заказа 27
Обработка формы 29
Встраивание PHP в HTML 29
Использование PHP-дескрипторов 30
Стили PHP-дескрипторов 30
PHP-операторы 31
Пробелы 31
Комментарии 32
Добавление динамического содержимого ... 33
Вызов функций 33
Функция date() 33
Доступ к переменным формы 34
Переменные формы 34
Конкатенация строк 35
Переменные и литералы 35
Идентификаторы 36
Переменные, объявляемые пользователем .. 36
Присвоение значений переменным 36
Типы переменных 36
Типы данных PHP 37
Преимущества типов 37
Приведение типов 37
Переменные переменных 38
Константы 38
Область действия переменных 39
Знаки операций 39
Арифметические операции 39
Строковые операции 40
Операции присваивания 40
Операции сравнения 43
Логические операции 44
Поразрядные операции 44
Другие операции 45
Использование операций: вычисление итога по форме 46
Приоритет и ассоциативность: вычисление выражений 47
Функции для работы с переменными 48
Проверка и установка типов переменных 48
Проверка состояния переменных 49
Повторная интерпретация переменных 50
Управляющие структуры 50
Принятие решений с помощью условных операторов 50
Операторы if 50
Блоки кода 51
Примечание: выравнивание кода при помощи отступов 51
Операторы else 51
Операторы elseif 52
Операторы switch 52
Сравнение различных условных операторов . 54
Итерация: повторение действий 54
Циклы while 55
Циклы for 56
Циклы do..while 57
Выход из управляющей структуры или сценария 58

Глава 2. Хранение и получение данных 59

Сохранение данных с целью дальнейшего использования 60
Сохранение и получение заказов в компании Боба 60
Обзор обработки файлов 61
Открытие файла 61
Режимы файлов 61
Использование функции fopen() для открытия файла 61
Открытие удаленных файлов через FTP или HTTP 63

Проблемы, возникающие при открытии файлов....64
Запись в файл....65
Параметры функции fwrite()....65
Форматы файлов....65
Закрытие файла....66
Считывание из файла....67
Открытие файла для чтения: fopen()....68
Определение конца файла: feof()....68
Построчное считывание: fgets(), fgetss() и fgetcsv()....68
Считывание всего файла: readfile(), fpassthru(), file()....69
Считывание символа: fgetc()....70
Считывание строк произвольной длины: fread()....70
Другие полезные файловые функции....70
Проверка существования файла: file_exists() 70
Выяснение размера файла: filesize()....71
Удаление файла unlink()....71
Перемещение внутри файла: rewind(), fseek и ftell()....71
Блокирование файлов....72
Более рациональный способ обработки: системы управления базами данных....73
Проблемы, связанные с использованием двумерных файлов....73
Как эти проблемы решаются с помощью СУРБД....74

Глава 3. Использование массивов .. 75

Что такое массив....76
Численно индексированные массивы....76
Инициализация численно индексированных массивов....76
Доступ к содержимому массива....77
Использование циклов для доступа к массиву 78
Ассоциативные массивы....78
Инициализация ассоциативного массива....78
Доступ к элементам массива....78
Организация циклов с использованием each() и list()....79
Многомерные массивы....80
Сортировка массивов....83
Использование функции sort()....83
Использование функций asort() и ksort() для сортировки ассоциативных массивов....83
Сортировка в обратном порядке....83
Сортировка многомерных массивов....84
Определяемые пользователем функции сортировки....84
Определяемые пользователем функции сортировки в обратном порядке....85
Изменение порядка следования элементов в массивах....86
Использование функции shuffle()....86
Использование функции array_reverse()....87
Загрузка массивов из файлов....88
Другие манипуляции с массивами....90
Перемещение внутри массива: функции each, current(), reset(), end(), next(), pos() и prev()....90
Применение любой функции к каждому элементу массива: array_walk()....91
Подсчет элементов в массиве: функции count(), sizeof() и array_count_values()....92
Преобразование массивов в скалярные переменные: функция extract()....93

Глава 4. Манипулирование строками и регулярные выражения....95

Пример приложения: Smart Form Mail....95
Форматирование строк....97
Усечение строк: функции chop(), ltrim() и trim()....97
Форматирование строк для представления ... 98
Форматирование строк для хранения: функции AddSlashes() и StripSlashes()....100
Объединение и разделение строк с помощью строковых функций....101
Использование функций explode(), implode() и join()....102
Использование функции strtok()....102
Использование функции substr()....103
Сравнение строк....104
Упорядочение строк: функции strcmp(), strcasecmp() и strnatcmp()....104

Проверка длины строки с помощью функции strlen() 104
Сопоставление и замена подстрок с помощью строковых функций 105
Поиск строк в строках: функции strstr(), strchr(), strrchr(), stristr() 105
Определение позиции подстроки: функции strpos(), strrpos() 106
Замена подстрок; str_replace(), substr_replace() 107
Введение в регулярные выражения 108
Наборы символов и классы 108
Повторение 110
Подвыражения 110
Подсчитываемые подвыражения 110
Привязка к началу или концу строки 110
Ветвление 111
Сопоставление с литеральными значениями специальных символов 111
Краткое описание специальных символов . 111
Использование регулярных выражений в приложении Smart Form 111
Поиск подстрок с помощью регулярных выражений 112
Замена подстрок с помощью регулярных выражений 113
Разделение строк с помощью регулярных выражений 113
Сравнение строковых функций и функций регулярных выражений 114

Глава 5. Повторное использование кода и создание функций 115

Для чего требуется повторное использование кода? 116
Стоимость 116
Надежность 116
Единообразие 116
Использование операторов require() и include() 116
Использование оператора require() 117
Расширения имен файлов и оператор require() 118
PHP-дескрипторы и оператор require() 118
Использование оператора require() для шаблонов Web-сайта 118
Использование директив auto_prepend_file и auto_append_file 122
Использование оператора include() 123
Использование функций в PHP 125
Вызов функций 125
Обращение к неопределенной функции 126
Регистр и имена функций 127
Для чего нужно определять собственные функции? 127
Базовая структура функции 128
Именование функций 128
Параметры 129
Область действия 131
Передача по ссылке и передача по значению 133
Возврат из функций 134
Возврат значений из функций 135
Блоки кода 136
Рекурсия 137

Глава 6. Объектно-ориентированное программирование на PHP 139

Концепции объектно-ориентированного программирования 139
Классы и объекты 140
Полиморфизм 141
Наследование 141
Создание классов, атрибутов и операций в PHP 142
Структура класса 142
Конструкторы 142
Создание экземпляров класса 143
Использование атрибутов класса 143
Вызов операций класса 145
Реализация наследования в PHP 146
Перекрытие 146
Множественное наследование 148
Разработка классов 148
Написание кода класса 149

Часть 2. Использование MySQL .. 157

Глава 7. Проектирование Web-баз данных ... 158

Общее представление о реляционных базах данных ... 159
Таблицы ... 159
Столбцы ... 159
Строки ... 159
Значения ... 160
Ключи ... 160
Схемы ... 161
Отношения ... 161
Как спроектировать собственную Web-базу данных ... 162
Подумайте о реальных объектах, которые вы моделируете ... 162
Избегайте хранения избыточной информации 163
Используйте атомарные значения столбцов . 164
Выбирайте подходящие ключи ... 164
Подумайте над вопросами, которые потребуется задать базе данных ... 165
Избегайте проектов с большим количеством пустых атрибутов ... 165
Типы таблиц ... 165
Архитектура Web-баз данных ... 166
Архитектура ... 166

Глава 8. Создание Web-базы данных ... 168

Замечания к использованию монитора MySQL ... 169
Вход в систему MySQL ... 170
Создание баз данных и подключение пользователей ... 171
Создание базы данных ... 171
Пользователи и привилегии ... 171
Знакомство с системами привилегий MySQL ... 171
Принцип наименьших привилегий ... 172
Установка пользователей: команда GRANT. 172
Типы и уровни привилегий ... 173
Команда REVOKE 175
Установка пользователя для доступа из Web ... 176
Выход из системы пользователя root ... 176
Использование требуемой базы данных ... 176
Создание таблиц баз данных ... 177
Что означают другие ключевые слова ... 178
Что означают типы столбцов ... 178
Просмотр базы данных с помощью команд SHOW и DESCRIBE ... 180
Идентификаторы MySQL ... 181
Типы данных столбцов ... 182
Числовые типы ... 182

Глава 9. Работа с базой данных MySQL ... 187

Что такое SQL? ... 187
Вставка данных в базу данных ... 188
Извлечение данных из базы данных ... 189
Извлечение данных по определенному критерию ... 191
Извлечение данных из нескольких таблиц .. 192
Использование других имен таблиц: псевдонимы ... 196
Извлечение данных в определенном порядке 197
Группировка и агрегирование данных ... 198
Выбор количества отображаемый строк ... 199
Обновление записей в базе данных ... 200
Изменение таблиц после создания ... 200
Удаление записей из базы данных ... 202
Удаление таблиц ... 202
Удаление целой базы данных ... 202

Глава 10. Доступ к базе данных MySQL из Web с помощью PHP ... 203

Как работает архитектура Web-баз данных . 204
Основные шаги выполнения запросов к базе данных через Web ... 206
Проверка и фильтрация данных, исходящих от пользователя ... 207
Установка соединения ... 207
Выбор базы данных ... 209
Выполнение запроса к базе данных ... 209
Получение результатов запроса ... 210

Отсоединение от базы данных....211
Внесение новой информации в базу данных....211
Прочие полезные функции PHP-MySQL 214
Освобождение ресурсов....214
Создание и удаление баз данных....214
Другие интерфейсы PHP-баз данных....214

Глава 11. Дополнительные возможности MySQL....216

Детальное описание системы привилегий 216
Таблица user....217
Таблицы db и host....218
Таблицы tables_priv и columns_priv....219
Управление доступом: как MySQL использует таблицы привилегий....220
Обновление привилегий: когда изменения вступают в силу?....221
Обеспечение безопасности баз данных MySQL....221
MySQL с точки зрения операционной системы....221
Пароли....222
Привилегии пользователей....222
Тонкости Web....223
Получение дополнительной информации о базах данных....223
Получение информации с помощью SHOW 223
Получение информации о столбцах с помощью DESCRIBE....225
Как запросы работают с EXPLAIN....225
Ускорение использования базы данных при помощи индексов....229
Советы по оптимизации....229
Оптимизация проектирования....229
Права доступа....229
Оптимизация таблиц....229
Применение индексов....230
Использование значений, принятых по умолчанию....230
Применяйте постоянные соединения....230
Другие советы....230
Различные типы таблиц....230
Загрузка данных из файла....231

Часть 3. Системы электронной торговли и безопасность....232

Глава 12. Эксплуатация сайта электронной коммерции....233

Преследуемые цели....233
Типы коммерческих Web-сайтов....234
Сетевые брошюры....234
Прием заказов на товары и услуги....237
Предоставление услуг и цифровой продукции 240
Дополнительные товары и услуги....241
Снижение расходов....241
Риски и угрозы....242
Взломщики....242
Невозможность привлечения компаньонов. 243
Отказы оборудования....243
Сбои питания, коммуникационных линий или сети, отказы службы доставки....243
Интенсивная конкуренция....243
Сбои программного обеспечения....244
Изменения в политике и налогообложении. 244
Ограниченная пропускная способность системы....244

Глава 13. Вопросы безопасности в электронной коммерции....245

Насколь важна ваша информация?....246
Угрозы безопасности....246
Вскрытие конфиденциальных данных....247
Потеря или разрушение данных....248
Изменение данных....249
Отказ в обслуживании....250
Ошибки программного обеспечения....251
Отказ от обязательств....252
Достижение баланса между практичностью, производительностью, стоимостью и защитой....253
Разработка стратегии защиты....254
Принципы аутентификации....254
Использование аутентификации....255
Основы шифрования....256

Шифрование с закрытым ключом....257
Шифрование с открытым ключом....258
Цифровые подписи....258
Цифровые сертификаты....259
Безопасные Web-серверы....260
Аудит и регистрация....262
Брандмауэры....262
Резервное копирование данных....263
Резервное копирование общих файлов....263
Резервное копирование и восстановление баз данных MySQL....263
Физическая безопасность....264

Глава 14. Аутентификация с помощью PHP и MySQL....265

Идентификация пользователей....265
Реализация контроля доступа....266
Хранение паролей....268
Шифрование паролей....270
Защита множества страниц....271
Базовая аутентификация....272
Использование базовой аутентификации в PHP....273
Использование базовой аутентификации при помощи файлов .htaccess сервера Apache .. 274
Использование базовой аутентификации в IIS....277
Использование аутентификации через модуль mod_auth_mysql....279
Установка модуля mod_auth_mysql....279
Заработало?....280
Использование модуля mod_auth_mysql....281
Создание собственного метода аутентификации....281

Глава 15. Реализация безопасных транзакций в PHP и MySQL....283

Обеспечение безопасности транзакций....283
Пользовательская машина....284
Internet....285
Ваша система....286
Использование слоя безопасных сокетов . 287
Проверка данных, вводимых пользователем .. 290
Обеспечение безопасного хранения данных . 291
Зачем хранить номера кредитных карточек? 292
Использование шифрования в PHP....292
Инсталляция GPG....293
Тестирование GPG....295

Часть 4. Усложненные технологии применения PHP....301

Глава 16. Взаимодействие с файловой системой и сервером ...302

Основы загрузки файлов на сервер....302
HTML-код загрузки файлов на сервер....303
Реализация PHP-кода для работы с файлом 304
Часто возникающие затруднения....307
Использование функций работы с каталогами....307
Считывание из каталогов....307
Получение сведений о текущем каталоге ... 309
Создание и удаление каталогов....309
Взаимодействие с файловой системой....310
Считывание информации из файла....310
Изменение свойств файла....312
Создание, перемещение и удаление файлов....312
Функции запуска программ....313
Взаимодействие со средой: getenv() и putenv()....314

Глава 17. Использование функций работы с сетью и протоколами....316

Обзор доступных протоколов....316
Отправка и получение почты....317
Использование других Web-служб через NTTP....317
Применение функций сетевого поиска....320
Использование FTP....323
Использование FTP для создания резервной или зеркальной копии файла....323

Загрузка файлов на сервер....329
Как избежать тайм-аутов....329
Другие функции работы с FTP....329
Использование общих сетевых соединений с помощью библиотеки cURL....330

Глава 18. Управление датой и временем....333

Получение даты и времени средствами PHP....333
Использование функции date()....333
Работа с метками времени UNIX....335
Использование функции getdate()....335
Проверка правильности дат....336
Преобразования даты в форматах PHP и MySQL....336
Операции над датами....338
Использование календарных функций....338

Глава 19. Создание изображений....340

Настройка поддержки изображений в PHP....340
Форматы изображений....341
JPEG....341
PNG....341
WBMP....342
GIF....342
Создание изображений....343
Создание холста....344
Рисование или вывод текста в изображение....344
Вывод окончательного рисунка....346
Освобождение ресурсов....347
Использование автоматически создаваемых изображений на других страницах....347
Использование текста и шрифтов при создании изображений....347
Настройка холста....350
Подбор размера текста на кнопке....350
Позиционирование текста....352
Вывод текста на кнопку....353
Заключительные действия....353
Рисование фигур и построение графиков 353
Другие функции обработки изображений 359

Глава 20. Управление сеансами в PHP....361

Что такое управление сеансом....361
Основные функциональные средства управления сеансом....362
Что такое cookie-набор?....362
Установка cookie-наборов из PHP....363
Использование cookie-наборов в сеансах....363
Сохранение идентификатора сеанса....364
Реализация управления простым сеансом 364
Запуск сеанса....364
Регистрация переменных сеанса....365
Использование переменных сеанса....365
Отмена регистрации переменных и завершение сеанса....365
Пример простого сеанса....366
Конфигурирование управления сеанса....368
Выполнение аутентификации пользователей средствами управления сеансом....368

Глава 21. Другие полезные свойства PHP....374

Использование магических кавычек....374
Выполнение команд, содержащихся в строке, - функция eval()....375
Прерывание выполнения: die и exit....376
Сериализация....376
Получение информации о рабочей среде PHP....377
Определение загруженных расширений....377
Определение владельца сценария....378
Определение даты последнего изменения сценария....378
Динамическая загрузка расширений....378
Временное изменение среды исполнения 378
Выделение элементов исходного кода....379

Часть 5. Разработка практических приложений на PHP и MySQL ... 380

Глава 22. Применение PHP и MySQL при разработке крупных проектов . 381

Применение методов проектирования программного обеспечения при разработке Web-приложений ... 382
Планирование и сопровождение проекта Web-приложения ... 382
Повторное использование кода ... 383
Написание удобного для сопровождения кода ... 384
Стандарты написания кода ... 384
Фрагментирование кода ... 387
Использование стандартной структуры каталогов ... 387
Документирование и распределение функций собственной разработки ... 387
Управление версиями ... 388
Выбор среды разработки ... 389
Документирование проектов ... 390
Создание прототипов ... 391
Разделение логики и содержимого ... 392
Оптимизация кода ... 392
Использование простой оптимизации ... 392
Использование продуктов компании Zend ... 393
Тестирование ... 394
Дополнительная информация ... 395
Что дальше ... 395

Глава 23. Отладка ... 396

Программные ошибки ... 396
Синтаксические ошибки ... 397
Ошибки времени выполнения ... 398
Логические ошибки ... 402
Вспомогательное средство отладки переменных ... 404
Уровни сообщений об ошибках ... 405
Изменение настроек сообщений об ошибках ... 406
Генерация собственных ошибок ... 408
Эффективная обработка ошибок ... 408
Удаленная отладка ... 410

Глава 24. Аутентификация и персонализация пользователей ... 411

Задача ... 411
Компоненты решения ... 412
Идентификация и персонализация пользователей ... 412
Хранение закладок ... 413
Рекомендация закладок ... 413
Обзор проекта ... 413
Реализация базы данных ... 415
Реализация основы сайта ... 416
Аутентификация пользователей ... 418
Регистрация ... 418
Вход в систему ... 423
Выход из системы ... 426
Смена паролей ... 427
Переустановка забытых паролей ... 428
Хранение и извлечение закладок ... 432
Добавление закладок ... 432
Отображение закладок ... 434
Удаление закладок ... 434
Выработка рекомендаций ... 436
Заключение и возможные расширения ... 440

Глава 25. Создание покупательской тележки ... 441

Задача ... 442
Компоненты решения ... 442
Построение интерактивного каталога ... 442
Отслеживание выбираемого товара ... 442
Платежи ... 443
Интерфейс администрирования ... 443
Обзор решения ... 443
Реализация базы данных ... 447
Реализация интерактивного каталога ... 449
Список категорий ... 451
Список книг в категории ... 453
Отображение информации о книгах ... 454

Реализация покупательской тележки ... 455
Использование сценария show_cart.php ... 456
Просмотр содержимого тележки ... 458
Добавление элементов в тележку ... 460
Сохранение изменений содержимого тележки ... 462
Печать итоговых данных в строке заголовка ... 462
Выполнение расчета ... 463
Реализация платежа ... 467
Реализация интерфейса администрирования ... 469
Расширение проекта ... 475
Использование существующей системы ... 476

Глава 26. Построение системы управления содержимым ... 477

Задача ... 477
Требования к проекту ... 478
Редактирование содержимого ... 478
Ввод содержимого в систему ... 478
FTP ... 478
Преимущество баз данных перед файлами для хранения содержимого ... 479
Структура документов ... 479
Использование метаданных ... 480
Форматирование вывода ... 480
Управление изображениями ... 481
Обзор проекта ... 483
Создание базы данных ... 484
Реализация ... 486
Интерфейсная часть ... 486
Прикладная часть ... 488
Поиск статей ... 495
Окно редактора ... 498
Расширение проекта ... 499

Глава 27. Построение почтовой службы, основанной на Web ... 500

Задача ... 500
Компоненты решения ... 501
Обзор проекта ... 502
Создание базы данных ... 503
Архитектура сценария ... 505
Вход и выход из системы ... 509
Создание учетных записей ... 511
Создание новой учетной записи ... 513
Изменение существующей учетной записи ... 515
Удаление учетной записи ... 515
Чтение почтовых сообщений ... 516
Выбор учетной записи ... 516
Просмотр содержимого почтового ящика ... 518
Чтение почтовых сообщений ... 521
Просмотр заголовков сообщений ... 523
Удаление почтовых сообщений ... 524
Отправка почты ... 525
Отправка нового сообщения ... 525
Ответ на сообщение или его переадресация ... 527
Расширение проекта ... 528

Глава 28. Создание менеджера списков рассылки ... 529

Задача ... 530
Компоненты решения ... 530
Определение базы данных списков и подписчиков ... 530
Загрузка файлов ... 531
Отправка сообщений электронной почты с присоединениями ... 531
Обзор решения ... 531
Создание базы данных ... 533
Архитектура сценария ... 535
Реализация регистрации ... 542
Создание новой учетной записи ... 542
Регистрация ... 544
Реализация функций пользователя ... 547
Просмотр списков рассылки ... 547
Просмотр сведений о списке рассылки ... 551
Просмотр архивов списков рассылки ... 553
Подписка и отмена подписки ... 554
Изменение параметров настройки учетной записи ... 555
Изменение паролей ... 555
Выход из системы ... 557

Реализация функций администратора 557
Создание нового списка рассылки 558
Загрузка нового информационного бюллетеня .. 559
Выполнение загрузки нескольких файлов ... 562
Предварительный просмотр информационного бюллетеня 565
Отправка сообщения 566
Расширение проекта 570

Глава 29. Создание Web-форумов 571

Компоненты решения 572
Обзор решения 573
Разработка базы данных 575
Просмотр дерева статей 577
Раскрытие и свертывание 579
Отображение статей 581
Использование класса treenode 582
Просмотр отдельных статей 588
Добавление новых статей 589
Расширение проекта 595
Использование существующих систем 595

Глава 30. Генерация персонифицированных документов в формате переносимых документов (PDF) 596

Задача 596
Оценка форматов документов 597
Бумажный документ 597
ASCII 598
HTML 598
Форматы текстовых процессоров 598
Расширенный текстовый формат 599
PostScript 600
Переносимый формат документов 600
Компоненты решения 601
Система вопросов и ответов 601
Программное обеспечение для генерации документов 602
Обзор решения 604
Задание вопросов 605
Оценка ответов 606
Генерация RTF-сертификата 608
Генерация PDF-сертификата из шаблона ... 611
Генерация PDF-документа с использованием PDFlib 614
Сценарий Hello World для PDFlib 614
Генерация сертификата с помощью PDFlib 618
Проблемы, связанные с заголовками 624
Расширение проекта 625

Часть 6. Приложения 626

Приложение А. Инсталляция PHP 4 и MySQL 627

Запуск PHP в качестве интерпретатора CGI или модуля 628
Установка Apache, PHP и MySQL под UNIX . 629
Apache и mod_SSL 632
Фрагменты файла httpd.conf 634
Работает ли поддержка PHP? 635
Работает ли SSL? 635
Установка Apache, PHP и MySQL под Windows 636
Установка MySQL под Windows 637
Установка Apache под Windows 638
Различия между версиями Apache для Windows и UNIX 640
Установка PHP под Windows 641
Заметки по инсталляции для Microsoft IIS . 643
Замечания по инсталляции для Microsoft PWS . 644
Другие конфигурации 644

Приложение В. Ресурсы Internet ... 645

Ресурсы, посвященные PHP 645
Ресурсы, посвященные MySQL и SQL 647
Ресурсы, посвященные Apache 647
Разработка для Web 647

Предметный указатель 648

Об авторах

Лаура Томсон (Laura Thomson) читает лекции по программированию для Web на факультете компьютерных наук университета RMIT в Мельбурне, Австралия. Одновременно она является одним из партнеров известной в области Web-разработки фирмы Tangled Web Design. Ранее Лаура работала в компании Telstra и Бостонской консультационной группе (Boston Consulting Group). Она имеет степень бакалавра прикладных наук (компьютерные пауки) и степень с отличием бакалавра технических наук (разработка компьютерных систем), а в настоящее время завершает работу над диссертацией на соискание звания доктора философии по теме адаптивных Web-сайтов. В свободное время любит поспать. С Лаурой можно связаться по электронной почте: **laura@tangledweb.com.au.**

Люк Веллинг (Luke Welling) читает лекции по разработке программного обеспечения в школе электрических и компьютерных систем при университете RMIT в Мельбурне, Австралия. Он также является партнером фирмы Tangled Web Design. Люк имеет степень бакалавра прикладных наук (компьютерные науки) и в настоящее время завершает работу над диссертацией на соискание звания магистра по теме "Генетические алгоритмы для разработки коммуникационной сети". В свободное время пытается бороться с бессонницей. С Люком можно связаться по электронной почте: **luke@tangledweb.com.au.**

О соавторах

Исраэль Денис-мл. (Israel Denis Jr.) — внештатный консультант по системам электронной торговли, сотрудничающий с фирмами по всему миру. Он специализируется на интеграции таких пакетов планирования ресурсов предприятия, как SAP и Lawson, со специализированными решениями для Web. В 1998 г. Денис получил степень магистра по специальности электромашиностроения в техническом колледже Джорджии, Атланта, шт. Джорджия. Он является автором многочисленных статей по Linux, Apache, PHP и MySQL; с ним можно связаться по электронной почте: **idenis@ureach.com**.

Крис Ньюмен (Chris Newman) — программист-консультант, специализирующийся на разработке динамических приложений для Internet. Он обладает большим опытом в области использования PHP и MySQL для создания широкого множества приложений, используемых в международной клиентской базе данных. Он выпускник Кильского университета и живет в Сток-он-Тренте, в Англии, где руководит компанией Lightwood Consultancy Ltd. Более подробную информацию о компании Lightwood Consultancy Ltd. можно найти па Web-сайте по адресу **http://www.lightwood.net,** а непосредственно с Ньюмэном можно связаться по электронной почте: **chris@lightwood.net.**

Посвящение

Нашим мамам и папам

Благодарности

Мы хотели бы поблагодарить коллектив издательства Sams за проделанную ими тяжкую работу. В частности, мы хотели бы выразить признательность Шелли Джонстон Маркенди (Shelley Jonston Markandy), без чьей преданности и терпения издание этой книги было бы невозможным. Мы хотели бы также поблагодарить Исраэля Дениса-мл. и Криса Ньюмена за их значительный вклад.

Мы высоко ценим работу, проделанную командами разработки PHP и MySQL. Выполняемая ими работа облегчала нам жизнь в течение ряда минувших лет, и это продолжается до сих пор.

Мы благодарим Эдриана Клоуза (Adrian Close) за то, что в 1998 г. он сказал: "Вы можете построить это на PHP", а также Джеймса Вудса (James Woods) и всех сотрудников компании Law Parners, которые предоставили нам столь интересную работу по тестированию возможностей PHP.

И, наконец, мы хотели бы выразить благодарность семьям и друзьям, которые мирились с нашим затворничеством в разгар отпускного сезона. Особенно мы благодарим за поддержку, оказанную членами наших семей: Жюли, Роберту, Мартину, Лесли, Адаму, Полу, Сэнди, Джеймсу и Арчеру.

Введение

Вы держите в руках книгу *"Разработка Web-приложений на PHP и MySQL"*. На ее страницах вы найдете наиболее важные сведения, почерпнутые авторами из опыта использования PHP и MySQL — двух наиболее популярных инструментальных средств Web-разработки.

Во введении рассматриваются следующие вопросы:

- Для чего следует прочесть эту книгу
- Чего можно добиться, используя эту книгу
- Что собой представляют PHP и MySQL и чем они хороши
- Обзор новых свойств PHP 4
- Как построена эта книга

Итак, приступим.

Для чего следует прочесть эту книгу

Эта книга научит создавать интерактивные Web-сайты, начиная с простейшей формы заказа и заканчивая сложными безопасными сайтами систем электронной торговли. Более того, вы узнаете, как это делать, используя технологии программного обеспечения с открытым исходным кодом (Open Source).

Эта книга предназначается читателям, которые уже знакомы, по крайней мере, с основами HTML и ранее создавали программы на современных языках программирования, но, возможно, не занимались программированием для Internet или не использовали реляционные базы данных. Книга несомненно окажется полезной для начинающих программистов, однако им для более качественного усвоения изложенного материала может потребоваться несколько больше времени. Мы старались не оставить без внимания ни одну из базовых концепций, но освещаем их кратко. В основном, книга адресована тем читателям, которые стремятся овладеть PHP и MySQL для построения крупных коммерческих Web-сайтов. Эта книга должна помочь быстрее приступить к делу также и желающим перейти на другой язык Web-разработки.

Мы написали данную книгу, поскольку устали от книг по PHP, по существу являющихся справочниками по функциям. Эти книги полезны, но они не могут помочь решить конкретную задачу. Мы же приложили все усилия, чтобы каждый из приведенных примеров нес в себе практическую пользу. Многие из примеров кода могут использоваться в Web-сайте непосредственно, а множество других — с минимальными изменениями.

Чего можно добиться, используя эту книгу

Прочтя эту книгу, вы сможете строить реальные, динамические Web-сайты. Если вам доводилось строить Web-сайты с использованием обычного HTML, то вам ясны ограничения такого подхода. При использований статического содержимого, созданного на основе чистого HTML, Web-сайт таковым и остается — статическим. Он остается неизменным, если только не обновить его физически. Пользователи не могут взаимодействовать с таким сайтом никаким осмысленным образом.

Использование такого языка, как PHP, и такой базы данных, как MySQL, позволяет делать сайты динамическими: настраиваемыми и содержащими информацию, изменяемую в реальном времени.

В данной книге, даже во вводных главах, мы намеренно основное внимание уделили реальным приложениям. Мы начнем с рассмотрения простой интерактивной системы заказов, а затем ознакомимся с различными составными частями PHP и MySQL.

Затем мы рассмотрим аспекты электронных систем продажи и безопасности во взаимосвязи с построением реального Web-сайта и покажем, как реализовать эти аспекты в среде PHP и MySQL.

В заключительном разделе книги мы поговорим о подходе к реальным проектам и ознакомим читателей с разработкой, планированием и построением следующих семи проектов:

- Аутентификация и персонификация пользователей
- Электронные покупательские тележки
- Системы управления содержимым
- Электронная почта Web
- Диспетчеры списков рассылки
- Web-форумы
- Генерация документов

Любой из этих проектов может использоваться в готовом виде или же модифицироваться в соответствие с конкретными потребностями. Мы выбрали их потому, что по нашему мнению они представляют семь наиболее широко используемых Web-приложений, создаваемых программистами. Если вам требуются другие приложения, эта книга должна помочь в достижении поставленной цели.

Что собой представляет PHP?

PHP — это серверный язык создания сценариев (или стороны сервера), разработанный специально для Web. В HTML-страницу можно внедрить код PHP, который будет выполняться при каждом ее посещении. Код PHP интерпретируется Web-сервером и генерирует HTML или иной вывод, наблюдаемый посетителем страницы.

Разработка PHP была начата в 1994 г. и вначале выполнялась одним человеком, Расмусом Лердорфом (Rasmus Lerdorf). Этот язык был принят рядом талантливых людей и претерпел три основных редакции, пока не стал широко используемым и зрелым продуктом, с которым мы имеем дело сегодня. К январю 2001 г. он использовался почти в пяти миллионах доменов во всем мире и их число продолжает быстро расти. Количество доменов, в которых в настоящее время используется PHP, можно выяснить на странице **http://www.php.net/usage.php.**

PHP — это продукт с открытым исходным кодом (Open Source). У пользователя имеется доступ к исходному коду. Его можно использовать, изменять и свободно распространять другим пользователям или организациям.

Первоначально PHP являлось сокращением от *Personal Home Page (Персональная начальная страница),* но затем это название было изменено в соответствии с рекурсивным соглашением по наименованию GNU (GNU = Gnu's Not Unix) и теперь означает *PHP Hypertext Preprocessor (Препроцессор гипертекста PHP).*

В настоящее время основной версией PHP является четвертая. Эта версия характеризуется несколькими существенными усовершенствованиями языка, которые рассматриваются в следующем разделе.

Адрес начальной страницы для PHP — **http://www.php.net**

Адрес начальной страницы для Zend — **http://www.zend.com.**

Что нового в PHP 4?

Если вы ранее использовали PHP, то несложно будет заметить ряд важных усовершенствований 4 версии:

- PHP 4 работает значительно быстрее предшествующих версий, поскольку в нем используется новый механизм Zend Engine. Если требуется еще более высокая производительность, по адресу **http://www.zend.com** можно получить модули Zend Optimizer, Zend Cache или Zend Compiler.
- PHP всегда можно было использовать как эффективный модуль сервера Apache. С появлением новой версии PHP можно устанавливать и в виде модуля ISAPI для Internet Information Server компании Microsoft.
- Теперь поддержка сеансов является встроенной. В предшествующих версиях для управления сеансом или создания собственного сеанса требовалось устанавливать дополнительный модуль PHPLib.

Что собой представляет MySQL?

MySQL (произносится *май-эс-кю-эл)* — очень быстрая, надежная *система управления реляционными базами данных (СУРБД).* База данных позволяет эффективно хранить, искать, сортировать и получать данные. Сервер MySQL управляет доступом к данным, позволяя работать с ними одновременно нескольким пользователям, обеспечивает быстрый доступ к данным и гарантирует предоставление доступа только имеющим на это право пользователям. Следовательно, MySQL является многопользовательским, многопотоковым сервером. Он применяет *SQL (Structured Query Language — язык структурированных запросов),* используемый по всему миру стандартный язык запросов в базы данных. MySQL появился на рынке в 1996 г., но его разработка началась еще в 1979 г. В настоящее время, по прошествии трех лет своего существования, эта система завоевала приз читательских симпатий журнала Linux Journal.

В настоящее время пакет MySQL доступен как программное обеспечение с открытым исходным кодом, но в случае необходимости можно получить и коммерческие лицензии.

Для чего следует использовать PHP и MySQL?

Приступая к созданию сайта системы электронной торговли, можно использовать множество различных продуктов.

Потребуется выбрать аппаратное обеспечение для Web-сервера, операционную систему, программное обеспечение Web-сервера, систему управления базами данных и язык программирования или создания сценариев.

Выбор некоторых из этих компонентов будет зависеть от уже произведенных выборов. Например, не все операционные системы могут работать на любом оборудовании, не все языки создания сценариев могут обеспечить подключение ко всем базам данных и т.д.

В этой книге не уделяется особое внимание аппаратному обеспечению, операционным системам и программному обеспечению Web-сервера. Нам это не требуется. Одно из замечательных свойств PHP в том, что он доступен для Microsoft Windows, для многих версий UNIX и выполняется на любых полнофункциональных Web-серверах. Система MySQL обладает такой же гибкостью.

Чтобы продемонстрировать это, примеры в книге написаны и протестированы на двух популярных вариантах установки:

- Linux с использованием Web-сервера Apache
- Microsoft Windows 2000 с использованием сервера Microsoft Internet Information Server (IIS)

Какие бы аппаратное обеспечение, операционная система и Web-сервер не были бы выбраны, мы надеемся, что вы серьезно задумаетесь об использовании PHP и MySQL.

Некоторые преимущества PHP

К числу конкурентов PHP относятся Perl, Active Server Pages (ASP) от Microsoft, Java Server Pages (JSP) и Allaire Cold Fusion.

PHP обладает множеством преимуществ по сравнению с этими продуктами, в числе которых:

- Высокая производительность
- Наличие интерфейсов ко многим различным системам баз данных
- Встроенные библиотеки для выполнения многих общих задач, связанных с Web
- Низкая стоимость
- Простота изучения и использования
- Переместимость
- Доступность исходного кода

Эти преимущества более подробно рассматриваются далее.

Производительность

PHP исключительно эффективен. Используя единственный недорогой сервер, можно обслуживать миллионы обращений в день. Результаты тестирования, опубликованные компанией Zend Technologies (http://www.zend.com), подтверждают более высокую производительность PHP по сравнению с конкурирующими продуктами.

Интеграция с базами данных

PHP обладает встроенной связностью со многими системами баз данных. В дополнение к MySQL, в числе прочих можно непосредственно подключаться к базам данных PostgreSQL, mSQL, Oracle, dbm, Hyperware, Informix, InterBase и Sybase.

Используя *Open Database Connectivity Standard (Стандарт открытого интерфейса связи с базами данных, ODBC),* можно подключаться к любой базе данных, для которых существует ODBC-драйвер. Это распространяется на продукты Microsoft и многих других компаний.

Встроенные библиотеки

Поскольку PHP был разработан для использования в Web, он имеет множество встроенных функций для выполнения широкого разнообразия полезных, связанных с

Web, задач. С его помощью можно "на лету" генерировать GIF-изображения, подключаться к другим сетевым службам, отправлять сообщения электронной почты, работать с cookie-наборами и генерировать PDF-документы — и все это посредством всего нескольких строк кода.

Стоимость

Пакет PHP является бесплатным. Наиболее новую версию можно в любой момент совершенно бесплатно выгрузить из **http://www.php.net.**

Изучение PHP

Синтаксис PHP основывается на других языках программирования, в первую очередь на C и Perl. Если вы уже знакомы с C, Perl или C-подобным языком, таким как C++ или Java, то почти сразу сможете эффективно использовать PHP.

Переносимость

Пакет PHP можно использовать под управлением многих различных операционных систем. Код PHP можно создавать в среде таких бесплатных Unix-подобных операционных систем, как Linux и FreeBSD, коммерческих версий Unix типа Solaris и IRIX или различных версий Microsoft Windows.

Как правило, программы будут работать без каких-либо изменений в различных средах с установленным PHP.

Исходный код

Пользователь имеет доступ к исходному коду PHP. В отличие от коммерческих закрытых программных продуктов, если нужно что-либо изменить или добавить в этом языке, то это всегда можно сделать.

Не следует дожидаться, пока фирма-изготовитель выпустит правки (патчи). Нет необходимости беспокоиться о том, что изготовитель собирается покинуть рынок или перестанет поддерживать продукт.

Некоторые преимущества MySQL

К конкурентам MySQL, помимо прочих, относятся PostgreSQL, Microsoft SQL Server и Oracle.

MySQL обладает многими преимуществами, в том числе высокой производительностью, низкой стоимостью, простотой конфигурирования и изучения, переносимостью и доступностью исходного кода.

Более подробно упомянутые преимущества рассматриваются ниже.

Производительность

MySQL без сомнений работает очень быстро. Результаты сравнительных тестов производительности, выполненных фирмой-изготовителем, можно посмотреть на странице **http://web.mysql.com/benchmark.html**. Многие из этих сравнительных тестов показывают, что MySQL работает на порядок быстрее конкурирующих продуктов.

Низкая стоимость

Пакет MySQL доступен бесплатно в соответствии с лицензией на программное обеспечение с открытым исходным кодом или, если это необходимо для приложения, за небольшую сумму можно приобрести коммерческую лицензию.

Простота использования

В большинстве современных баз данных используется SQL. Если ранее вы работали с другими СУРБД, переход к этой системе не должен вызывать какие-либо затруднения. Установка MySQL столь же проста, как и установка многих аналогичных продуктов.

Переносимость

MySQL может использоваться в среде многих различных систем UNIX, а также в среде Microsoft Windows.

Исходный код

Как и в случае PHP, исходный код MySQL можно выгружать и изменять.

Как построена эта книга

Книга разделена на пять основных разделов.

В части I, "Использование PHP", приводится обзор основных составляющих языка PHP с примерами. Каждый из примеров — не просто "игрушечный" код, а образец реальной программы, используемой при построении сайта системы электронной торговли. Этот раздел начинается с главы 1, "Краткий обзор PHP". Если вы уже использовали PHP, можете бегло просмотреть ее. Если же вы лишь начинаете знакомиться с PHP или вообще с программированием, возможно, потребуется изучить эту главу более внимательно.

В части II, "Использование MySQL", рассматриваются концепции и особенности разработки, связанные с использованием систем реляционных баз данных типа MySQL, использование SQL, подключение базы данных MySQL к внешним приложениям с помощью PHP и дополнительные вопросы использования MySQL, такие как безопасность и оптимизация.

В части III, "Системы электронной торговли и безопасность", освещаются некоторые из основных вопросов, связанных с разработкой сайта электронной торговли при использовании любого языка программирования. Безопасность является наиболее важной из них. Затем будет показано, как задействовать PHP и MySQL для аутентификации пользователей и для безопасного сбора, передачи и хранения информации.

В части IV, "Усовершенствованные технологии использования PHP". подробно рассматриваются некоторые из основных встроенных функций PHP. Мы остановили свой выбор на тех функциях, которые наверняка окажутся полезными при построении сайта электронной торговли. Читатели узнают о взаимодействии с сервером, взаимодействии с сетью, о генерировании изображений, манипулировании датой и временем и о переменных сеанса.

В части V, "Разработка практических приложений на PHP и MySQL", исследуются такие практические вопросы, как управление большими проектами и отладка; здесь же приводятся примеры проектов, демонстрирующие возможности и гибкость PHP и MySQL.

Заключение

Мы надеемся, что читатели получат такое же удовольствие от прочтения этой книги и ознакомления с PHP и MySQL, какое мы получили, начав применять эти продукты. Их действительно приятно использовать. Со временем вы сможете примкнуть к тысячам разработчиков Web-приложений, использующих эти надежные, обладающие большими возможностями инструментальные средства для простого и быстрого создания динамических, работающих в реальном времени Web-сайтов.

Часть 1

Использование PHP

В этой части:

1 Краткий обзор PHP

2 Хранение и получение данных

3 Использование массивов

4 Манипулирование строками и регулярные выражения

5 Повторное использование кода и написание функций

6 Объектно-ориентированный PHP

Глава 7

Краткий обзор PHP

В этой главе приводится краткий обзор синтаксиса и языковых конструкций PHP. Программистам PHP она может восполнить некоторые пробелы в знаниях. Если у вас имеется некоторый опыт использования C, ASP или другого языка программирования, она поможет быстрее начать работать с максимальной эффективностью.

Как следует изучив эту книгу, читатели научатся использовать PHP. Этому поспособствует множество реальных примеров, которые взяты из нашего опыта построения сайтов систем электронной торговли. Часто в учебниках по программированию описывается основы синтаксиса и приводятся очень простые примеры. Мы решили отказаться от такого подхода. Мы понимаем, что часто для понимания принципов использования языка читателям требуется получить какую-либо работающую программу, а не просто просмотреть еще один справочник по синтаксису и функциям, который ничем не лучше интерактивного руководства.

Постарайтесь разобраться в примерах — наберите их или загрузите из CD-ROM, измените, разбейте на модули и научитесь снова собирать их в единое целое.

Чтобы узнать, как переменные, операции и выражения используются в PHP, в этой главе начнем с примера интерактивной формы заказа товаров. В ней будут также описываться типы переменных и приоритеты операций. Вычисляя общую сумму и налог в заказе клиента, читатели узнают, как получить доступ к переменным формы и манипулировать ими.

Затем мы разработаем пример интерактивной формы заказа, используя созданный сценарий на PHP для проверки вводимых данных. Мы исследуем концепцию булевых значений и приведем примеры использования операторов **if, else, switch** и операции ?:.

В заключение, написав несколько строк PHP-кода для генерации повторяющихся HTML-таблиц, мы разберемся с применением циклов.

В этой главе освещаются следующие основные темы:

- Встраивание PHP в HTML
- Добавление динамического содержимого
- Доступ к переменным формы
- Идентификаторы
- Переменные, объявляемые пользователем
- Типы переменных
- Присвоение значений переменным
- Константы
- Область действия переменных
- Знаки операций и приоритеты
- Выражения
- Функции для работы с переменными
- Принятие решений с помощью **if, else и switch**
- Итерация: циклы **while, do и for**

Использование PHP

Чтобы можно было проработать примеры, приведенные в этой главе и во всей книге, потребуется иметь доступ к Web-серверу с установленным пакетом PHP. Для максимально эффективного использования приведенных примеров их следует запустить и попытаться изменить. Для этого необходимо располагать испытательным стендом, на котором можно выполнять эксперименты.

Если на используемом компьютере PHP не установлен, его следует установить или попросить об этом системного администратора. Инструкции по установке можно найти в приложении A, "Установка PHP 4 и MySQL". Все программное обеспечение, необходимое для установки PHP в среде UNIX или Windows NT, можно найти на сопровождающем книгу CD-ROM.

Пример приложения: Bob's Auto Parts

Одно из наиболее распространенных приложений любого языка создания серверных сценариев — обработка HTML-форм. Изучение PHP начнем с реализации формы заказа для вымышленной компании Bob's Auto Parts (Автозапчасти от Боба), торгующей запчастями. Все исходные тексты примеров, использованных в этой главе, находятся в каталоге **chapterl** на CD-ROM.

Форма *заказа*

В настоящий момент программист на HTML в компании Bob's Auto Parts занимается созданием формы для продаваемых Бобом запчастей. Форма заказа показана на рис. 1.1. Это сравнительно простая форма, аналогичная множеству других, которые можно встретить в Web. Прежде всего, Боб хотел бы иметь возможность выяснять, *что* заказал его клиент, вычислять общую сумму заказа и сумму налога с продаж, которую потребуется уплатить по выполнению заказа.

Фрагмент HTML-кода для создания этой формы приведен в листинге 1.1. В этом коде следует отметить два важных момента.

Во-первых, действию, выполняемому формой, присвоено имя PHP-сценария, который будет обрабатывать заказ клиента. (Этот сценарий будет разрабатываться несколько позже.) В общем случае значением атрибута **ACTION** является URL-адрес, который будет загружаться при нажатии пользователем кнопки отправки (submit). Данные, введенные пользователем в форму, будут отправляться по этому URL-адресу с использованием метода, указанного в атрибуте **METHOD:** либо GET (данные присоединяются в конец URL-адреса), либо **POST** (данные отправляются в виде отдельного пакета).

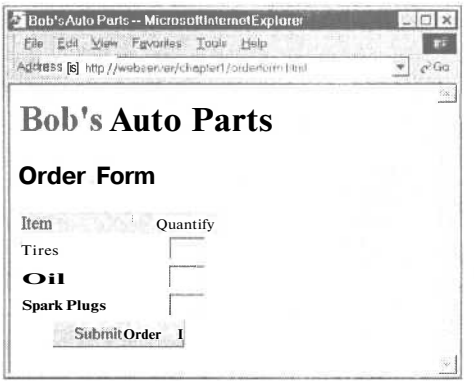


РИСУНОК 1.1 *В первоначальной форме Боба отражены только товары и их количества.*

Во-вторых, следует обратить внимание на имена полей формы — **tireqty, oilqty и sparkqty.** Эти имена будут снова использоваться в PHP-сценарии. Поэтому полям формы важно присваивать осмысленные имена, которые легко запомнить при написании PHP-сценария. Некоторые HTML-редакторы по умолчанию будут генерировать имена полей типа **field23.** Подобные имена трудно запомнить. Ваша задача как программиста на PHP существенно упростится, если эти имена будут отражать вводимые в поле данные.

Возможно, имеет смысл принять стандарт кодирования имен полей, чтобы для всех имен полей во всем сайте применялся один и тот же формат. Это облегчает запоминание того, например, сокращаются ли слова в именах полей или же для разделения слов в именах используются символы подчеркивания.

Листинг 1.1 orderform.html **— HTML-код для базовой формы заказа Боба.**

```
<form action="processorder.php" method=post>
<table border=0>
<tr bgcolor=#cccccc>
   <td width=150>Item</td>
   <td width=15>Quantity</td>
</tr>
<tr>
   <td>Tires</td>
   <td align=center><input type="text" name="tireqty" size=3 maxlength=3></td>
</tr>
<tr>
   <td>Oil</td>
   <td align=center><input type="text" name="oilqty" size=3 maxlength=3></td>
</tr>
<tr>
   <td>Spark Plugs</td>
   <td align=center><input type="text" name="sparkqty" size=3 maxlength=3></td>
</tr>
<tr>
   <td colspan=2 align=center><input type=submit value="Submit Order"></td>
</tr>
</table>
</form>
```

Обработка формы

Для обработки формы потребуется создать сценарий, упомянутый в атрибуте **ACTION** дескриптора **FORM** и названный **processorder.php**. Откройте текстовый редактор и создайте этой файл. Для этого введите следующий код:

```
<html>
<head>
   <title>Bob's Auto Parts - Order Results</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2>Order Results</h2>
</body>
</html>
```

Обратите внимание, что все введенное до сих пор представляет собой обычный HTML-текст. Теперь пора добавить в сценарий немного простого PHP-кода.

Встраивание PHP в HTML

Под заголовком <h2> файла введите следующие строки:

```
<?
     echo "<p>Order processed.";
?>
```

Сохраните файл и загрузите его в свой браузер, затем заполните форму и щелкните на кнопке Submit (Отправить). На экране должно отобразиться что-то похожее на изображение, показанное на рис. 1.2.

Обратите внимание, как написанный PHP-код внедряется в обычный HTML-файла. Попытайтесь просмотреть его в браузере. Вы должны увидеть следующие строки кода:

```
<html>
<head>
   <title>Bob's Auto Parts - Order Results</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2X3rder Results</h2>
<p>Order processed.</p></body>
</html>
```

Ни одной исходной строки PHP-кода не видно. Это происходит потому, что интерпретатор PHP просмотрел сценарий и заменил его строками вывода. Следовательно, из среды PHP можно создать чистый код HTML, пригодный для просмотра в любом браузере — иначе говоря, применяемый пользователем браузер не обязательно должен понимать PHP.

Это служит хорошей иллюстрацией концепции создания серверных сценари-

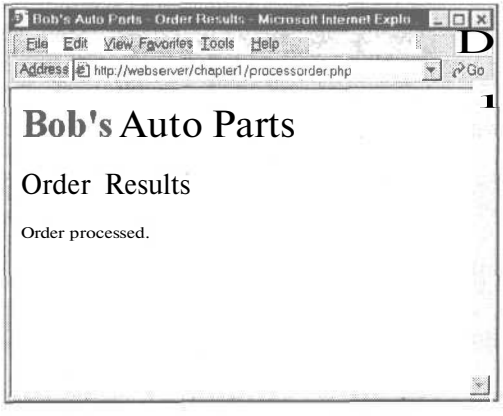


РИСУНОК 1.2 *Текст, переданный в PHP-конструкцию echo, повторяется в окне браузера.*

ев. PHP-код интерпретируется и выполняется на Web-сервере, в отличие от JavaScript и других технологий клиентской стороны, которые интерпретируются и выполняются в среде Web-браузера на компьютере пользователя.

Теперь код в рассматриваемом файле состоит из четырех частей:

- HTML
- Дескрипторы PHP
- Операторы PHP
- Пробелы

В него можно добавить также еще одну часть

- Комментарии

Большинство строк в приведенном примере — всего лишь простой HTML-код.

Использование PHP-дескрипторов

PHP-код из предыдущего примера начинается с последовательности <? и завершается последовательностью ?>. Это аналогично всем HTML-дескрипторам, поскольку все они начинаются с символа "меньше" (<) и завершаются символом "больше" (>). Эти символы называются PHP-дескрипторами, поскольку они указывают Web-серверу, где начинается, а где завершается PHP-код. Любой текст, расположенный между дескрипторами, будет интерпретироваться как PHP-код. Любой текст вне этих дескрипторов будет обрабатываться как обычный HTML-код. PHP-декрипторы позволяют *выполнить выход из* HTML.

Существуют различные стили дескрипторов. В приведенном примере используется сокращенный стиль. Возникновение каких-либо проблем при выполнении приведенного сценария может быть связано с тем, что сокращенные дескрипторы не активизированы в установках PHP. Давайте подробнее рассмотрим стили дескрипторов.

Стили PHP-дескрипторов

Фактически существует четыре различных стиля PHP-дескрипторов. Все приведенные ниже фрагменты кода эквивалентны.

- Сокращенный стиль

```
<? echo "<p>Order processed."; ?>
```

Именно этот стиль дескрипторов будет использоваться в данной книге. Этот же стиль дескрипторов используется по умолчанию разработчиками PHP.

Данный стиль дескрипторов является самым простым и соответствует стилю инструкций обработки SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки). Чтобы использовать этот тип дескрипторов (который является самым коротким), потребуется активизировать применение сокращенных дескрипторов в файле конфигурации или откомпилировать код самого PHP при активизированных сокращенных дескрипторах. Более подробную информацию можно найти в приложении А.

- XML-стиль

```
<?php echo "<p>Order processed."; ?>
```

Этот стиль дескрипторов может использоваться в документах XML (Extensible Markup Language — расширяемый язык разметки). Если планируется обслужи-

вание на сайте XML-документов, следует применять именно этот стиль дескрипторов.

- SCRIPT-стиль

```
<SCRIPT LANGUAGE='php'> echo "<p>Order processed."; </SCRIPT>
```

Этот стиль дескрипторов является самым длинным и будет знакомым тем, кому приходилось работать с JavaScript или VBScript. Он может использоваться в случае применения HTML-редактора, у которого возникают проблемы с другими стилями дескрипторов.

- ASP-стиль

```
<% echo "<p>Order processed."; %>
```

Этот стиль дескриптора совпадает с используемым в Active Server Pages (ASP). Он может применяться, если включен параметр настройки конфигурации asp_tags. Использование этого стиля дескрипторов может потребоваться в случае работы с редактором, ориентированным на ASP, либо если основное кодирование выполняется в ASP.

PHP-операторы

Действия, которые должен выполнить PHP-интерпретатор, указываются PHP-операторами, помещаемыми между открывающим и закрывающим дескрипторами. В следующем примере используется только один тип оператора:

```
echo "<p>Order processed.";
```

Как легко догадаться, конструкция **echo** приводит к очень простому результату; она выводит (или повторяет) в окне браузера переданную ей строку. Из рис. 1.2 видно, что в результате в окне браузера отображается текст **"Order processed.**".

Легко заметить, что в конце оператора **echo** присутствует точка с запятой. Этот символ используется для разделения PHP-операторов подобно тому, как точка используется для разделения предложений в обычном языке. Тем, кто ранее программировал на языке C или Java, подобное применение точки с запятой должно выглядеть знакомо.

Пропуск точки с запятой — это часто встречающаяся синтаксическая ошибка, которую совершенно легко допустить. Тем не менее, ее столь же легко выявить и исправить.

Пробелы

Символы пропусков, такие как пустые строки (возвраты каретки), пробелы между словами и символы табуляции обобщенно называют пробелами. Они игнорируются в PHP и HTML.

Возможно, вам уже известно, что браузеры игнорируют пробелы в HTML-коде. Механизм PHP действует точно так же. Рассмотрим следующих два фрагмента HTML-кода:

```
<h1>Welcome to Bob's Auto Parts!</h1><p>What would you like to order
   today?
```

и

```
<h1>Welcome to Bob's
Auto Parts!</h1>
<p>What would you like
 to order today?
```

Эти два фрагмента HTML-кода создают одинаковый вывод, поскольку для браузера они выглядят одинаково. Пробелы в HTML-кода использовать можно и нужно, поскольку они упрощают чтение самого HTML-кода. Это же справедливо по отношению к PHP. Пробелы между PHP-операторами не требуются, однако размещение каждого оператора в отдельной строке облегчает чтение кода. Например, фрагменты

```
echo "hello";
echo "world";
```

и

```
echo "hello";echo "world";
```

эквивалентны, но первая версия более читабельна.

Комментарии

Комментарии — это именно то, что следует из их названия: комментарии в коде служат примечаниями для людей, читающих текст кода. Комментарии могут использоваться для пояснения назначения сценария, сообщения информации о его создателе, пояснения, почему он написан именно таким образом, сообщения о времени его последнего изменения и т.п. Как правило, комментарии присутствуют во всех PHP-сценариях, за исключением простейших.

PHP-интерпретатор будет игнорировать любой текст, находящийся в комментарии. По существу, программа синтаксического анализа PHP попросту пропускает комментарии, которые для нее равнозначны пробелам.

PHP поддерживает комментарии в стилях C, C++ и сценариев оболочки.

Вот как выглядит многострочный комментарий в стиле C, который может появляться в начале PHP-сценария:

```
/* Автор: Боб Смит
     Дата последнего изменения: 10 апреля
     Этот сценарий обрабатывает заказы клиента.
*/
```

Многострочные комментарии должны начинаться с символов /* и завершаться символами */. Как и в C, многострочные комментарии не могут быть вложенными.

Можно также использовать однострочные комментарии в стиле C++:

```
echo "<p>Order processed."; // Начало вывода заказа
```

или в стиле сценариев оболочки:

```
echo "<p>Order processed."; # Начало вывода заказа
```

При использовании обоих этих стилей все, следующее за символом комментария (# или //) вплоть до конца строки или до завершающего дескриптора PHP. в зависимости от того, что встретится раньше, рассматривается как комментарий.

Добавление динамического содержимого

До сих пор мы не использовали PHP для выполнения каких-либо действий, которые нельзя было бы реализовать с помощью обычного HTML.

Основная побудительная причина использования языка создания серверных сценариев — желание иметь возможность предоставления пользователям сайта динамического содержимого. Это важно, поскольку содержимое, изменяющееся в соответствии с потребностями пользователя или по истечении некоторого времени, будет заставлять посетителей вновь и вновь возвращаться к сайту. PHP позволяет легко делать это.

Давайте начнем с рассмотрения простого примера. Замените PHP-код в файле **processorder.php** на следующий код:

```
<?
   echo "<p>Order processed at ";
   echo date("H:i, jS F");
   echo "<br>";
?>
```

В этом коде встроенная PHP-функция **date()** используется для сообщения клиенту даты и времени обработки его заказа. Эта значение будет изменяться при каждом выполнении сценария. Вывод, полученный в результате одного такого выполнения сценария, показан на рис. 1.3.

Вызов функций

Взгляните на вызов функции **date().** Это общая форма вызова функции. PHP имеет обширную библиотеку функций, которые можно использовать при разработке Web-приложений. Большинство из этих функций принимают какие-либо передаваемые им данные и возвращают какие-либо данные в качестве результатов.

Взгляните на вызов функции:

```
date("H:i, jS F")
```

Обратите внимание, что передаваемая функции строка (текстовые данные) заключена в круглые скобки. Это значение называется аргументом, или параметром, функции. Аргументы являются входными значениями, которые используются функцией для вывода каких-либо конкретных результатов.

Функция date()

Аргумент, передаваемый в функцию **date(),** должен быть строкой формата, задающей требуемый стиль вывода. Каждая буква в строке представляет часть строки даты и времени. H представляет часы в 12-часовом формате, i — минуты с ведущим нулем, когда это требуется, j — день месяца без ведущего нуля, S представляет обычный суффикс (в данном случае "th"), а F -- год, представленный четырьмя цифрами.

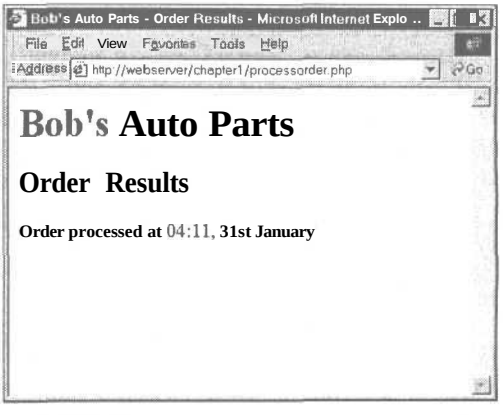


РИСУНОК 1.3 *PHP-функция date() возвращает отформатированную строку даты.*

(Полный список форматов, поддерживаемых функцией **datc(),** приводится в главе 18.)

Доступ к переменным формы

Весь смысл использования формы заказа заключается в получении информации о заказе клиента. Получение подробной информации о том, что ввел клиент, реализуется в PHP очень просто.

Внутри PHP-сценария к каждому из полей формы можно получить доступ как к переменной, имеющей то же имя, что и у поля формы. Давайте рассмотрим пример.

Добавьте следующие строки в нижнюю часть PHP-сценария:

```
echo "<p>Your order is as follows:";
echo "<br>";
echo $tireqty." tires<br>";
echo $oilqty." bottles of oil<br>";
echo $sparkqty." spark plugs<br>";
```

После обновления окна браузера вывод сценария должен выглядеть подобно показанному на рис. 1.4. Конечно же, фактические значения будут зависеть от того, *что* введено в форму.

Несколько примечательных особенностей этого примера описываются в следующих подразделах.

Переменные формы

В конечном счете данные из сценария попадают в PHP-переменные. Имена переменных в PHP легко узнать, поскольку все они начинаются с символа доллара ($). (Пропуск символа доллара — еще одна распространенная ошибка профаммирования.)

Существуют два способа доступа к данным формы через переменные.

В этом примере и во всей книге для ссылки на переменные формы используется сокращенный стиль. В данном случае имена переменных в сценарии совпадают с именами в HTML-форме. Так всегда происходит при сокращенном стиле. Переменные не обязательно объявлять в сценарии, поскольку они передаются в сценарий по существу так же, как аргументы передаются в функцию. При использовании этого стиля можно, например, просто начать работать с переменной, скажем, **$tireqty**, как и было сделано перед этим

Второй стиль заключается в получении переменных формы через один из двух массивов, хранящихся в переменных **$HTTP_POST_VARS** и **$HTTP_GET_VARS**. Один из этих массивов будет содержать подробную информацию о всех переменных формы. Выбор используемого массива зависит от метода отправки формы: **POST** или **GET.**

В рамках этого стиля для доступа к данным, введенным в поле формы **tireqty** в предыдущем примере, следовало бы использовать выражение

```
$HTTP_POST_VARS["tireqty"]
```

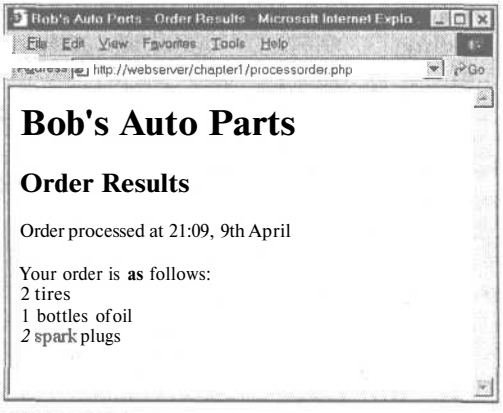


РИСУНОК 1.4 *Переменные формы, введенные пользователем, легко доступны в файле processorder.php.*

Сокращенный стиль можно применять только при установке в значение "On" директивы **register_globals** в файле **php.ini**. Это — настройка по умолчанию в стандартном файле **php.ini.**

Если же необходимо, чтобы директива **register_globals** была установлена в значение "Off", придется воспользоваться вторым стилем. При этом придется также установить в значение "On" директиву **track_vars**.

Более длинный стиль обеспечит более быстрое выполнение и предотвратит создание переменных, которые могут оказаться лишними. С другой стороны, сокращенный стиль проще читать и применять; к тому же он совпадает со стилем, используемым в предыдущих версиях PHP.

Оба эти метода аналогичны методам, используемым в других языках создания сценариев, например, Perl, и могут выглядеть знакомо.

Несложно было заметить, что на этом этапе не проверяется содержимое переменных на предмет корректного ввода важных данных в каждом из полей формы. Попытайтесь ввести заведомо неверные данные и посмотрите, что при этом произойдет. После изучения всей главы, возможно, вы сумеете включить в сценарий код, выполняющий определенную проверку данных.

Конкатенация строк

В сценарии оператор **echo** применялся для вывода значений, введенных пользователем в каждом из полей формы, за которыми следовал некоторый пояснительный текст. Если внимательно присмотреться к операторам **echo,** можно заметить, что между именем переменной и следующим за ним текстом содержится точка (.), например:

```
echo $tireqty. " tires<br>";
```

Это операция конкатенации строк, которая используется для объединения строк (фрагментов текста). Она будет часто применяться при пересылке вывода в браузер с помощью оператора **echo.** Эта операция позволяет избегать записи нескольких команд **echo.**

Иначе можно было бы записать

```
echo "$tireqty tires<br>";
```

Этот оператор эквивалентен первому. Оба формата допустимы и использование каждого из них — личное предпочтение каждого.

Переменные и литералы

Переменные и строки, объединяемые в каждом из операторов **echo,** — суть различные понятия. Переменные — это символы (обозначения) для данных. Строки же — это собственно данные. Фрагмент неструктурированных данных в программе наподобие рассматриваемой называется литералом, в отличие от переменной. **$tireqty — это** переменная, т.е. символ, который представляет введенные клиентом данные. С другой стороны, **" tires"** — это литерал. Он принимается так, как выглядит.

Ну что ж, мы почти закончили с этим вопросом. Помните второй приведенный ранее пример? PHP заменяется в строке имя переменной **$tireqty** на значение, хранящееся в переменной.

Фактически, в PHP существует два вида строк — с двойными кавычками и одинарными кавычками. PHP будет пытаться оценить строки, заключенные в двойные кавычки, что приводит к поведению, которое наблюдалось ранее. Строки, заключенные в одинарные кавычки, будут обрабатываться как истинные литералы.

Идентификаторы

Идентификаторы — это имена переменных. (Имена функций и классов также являются идентификаторами; функции и классы будут рассматриваться в главах 5 и 6.) Идентификаторы подчиняются некоторым простым правилам:

- Идентификаторы могут иметь любую длину и состоять из букв, цифр, символов подчеркивания и знаков доллара. Однако при использовании в идентификаторах знаков доллара следует проявлять внимательность. Причина сказанного станет понятна из раздела "Переменные переменных".
- Идентификаторы не могут начинаться с цифры.
- В PHP идентификаторы чувствительны к регистру. **$tireqty** и **$TireQty** — далеко не равнозначны. Попытка использования строчных символов вместо прописных и наоборот — очередная часто встречающаяся ошибка программирования. Исключение из этого правила составляют встроенные PHP-функции — их имена могут вводиться в любом регистре.
- Идентификаторы переменных могут совпадать с именами встроенных функций. Однако это обычно вызывает путаницу, потому подобных ситуаций следует избегать. Нельзя также создавать функции, идентификаторы которых совпадают с идентификаторами встроенных функций.

Переменные, объявляемые пользователем

В дополнение к переменным, передаваемым из HTML-формы, можно объявлять и использовать свои собственные переменные.

Одна из особенностей PHP заключается в том, что переменные не обязательно объявлять прежде, чем их можно будет использовать. Переменная будет создаваться при первом присвоении ей значения; подробнее об этом — в следующем разделе.

Присвоение значений переменным

Значения переменным присваиваются при помощи операции присваивания =. На сайте компании Боба требуется подсчитать общее количество деталей и общую сумму оплаты. Для хранения этих чисел можно создать две переменные. Для начала они инициализируются нулевыми значениями.

Добавьте следующие строки в нижнюю часть PHP-сценария:

```
$totalqty = 0;
$totalamount = 0.00;
```

Каждая из двух приведенных строк создает переменную и присваивает ей литеральное значение. Переменным можно присваивать также значения других переменных, например:

```
$totalqty = 0;
$totalamount = $totalqty;
```

Типы переменных

Тип переменной связан с видом хранящихся в ней данных.

Типы данных PHP

PHP поддерживает следующие типы данных:

- Integer (целый) — Используется для целых чисел
- Double (двойной точности) — Используется для действительных чисел
- String (строковый) — Используется для строк символов
- Array (массив) — Используется для хранения нескольких элементов данных одного типа (см. главу 3)
- Object (объект) — Используется для хранения экземпляров классов (см. главу 6)

PHP поддерживает также типы **pdfdoc и pdfinfo**, если он был установлен с поддержкой PDF (Portable Document Format — формат переносимых документов). Использование PDF в PHP рассматривается в главе 29.

Преимущества типов

Язык PHP весьма либерален по отношению к типам. В большинстве языков программирования переменные могут содержать только один тип данных, и этот тип должен быть объявлен прежде, чем переменную можно будет использовать, как это имеет место в C. В PHP тип переменной определяется присвоенным ей значением.

Например, при создании переменных **$totalqty** и **$totalamount** их начальные типы были определены следующим образом:

```
$totalqty = 0;
$totalamount = 0.00;
```

Поскольку переменной **$totalqty** было присвоено целочисленное значение 0, эта переменная имеет тип integer. Аналогично, переменная **$totalamount** имеет тип double.

Как ни странно, в сценарий вполне можно было бы добавить следующую строку:

```
$totalamount = "Hello";
```

В этом случае переменная **$totalamount** получила бы тип string. PHP в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней.

Подобная возможность изменения типов "на лету" может оказаться исключительно полезной. Помните, что PHP "автоматически" распознает тип данных, помещаемых в переменные. При чтении данных из переменной возвращаются данные в точности хранимого типа.

Приведение типов

Используя приведение типов, можно имитировать, будто переменная или значение имеет другой тип. Приведение выполняется так же, как и в C. Для этого достаточно перед переменной, тип которой требуется преобразовать, поместить в круглых скобках временный тип.

Например, две созданные выше переменные можно было бы объявить с использованием приведения типов.

```
$totalqty = 0;
$totalamount = (double)$totalqty;
```

Вторая строка означает "Взять значение, хранящееся в переменой **$totalqty**, интерпретировать как значение типа double и сохранить в переменной **$totalamount**". Пере-

менная $totalamount будет иметь тип double. Приведение типов не изменяет типы, поэтому типом переменной $totalqty остается integer.

Переменные переменных

PHP предоставляет еще один тип переменных — т.н. переменные переменных. Переменные переменных позволяют динамически изменять имена переменных.

(Как видите, PHP допускает очень большую свободу в этом вопросе — все языки разрешают изменять значение переменной, но лишь немногие позволяют изменять тип переменной, а уж совсем немногие — имя переменной.)

Способ достижения этого заключается в использовании значения одной переменной в качестве имени другой. Например, можно было бы определить

```
$varname = "tireqty";
```

Затем вместо $tireqty можно использовать $$varname, например, так:

```
$$varname = 5;
```

Это может показаться несколько запутанным, однако позже мы еще вернемся к этому вопросу. Вместо того чтобы перечислять и использовать каждую переменную формы отдельно, можно организовать цикл и переменную для автоматической обработки всех из них. Пример, иллюстрирующий такой метод, приводится в разделе, посвященном циклам for.

Константы

Как было показано ранее, значение, хранящееся в переменной, можно изменять. Кроме того, допускается также объявление констант. Как и переменная, константа хранит значение, но ее значение устанавливается однажды и впоследствии в сценарии изменяться не может.

В нашем примере приложения цены всех продаваемых запчастей можно было бы хранить в виде констант. Такие константы определяются с использованием функции define:

```
define("TIREPRICE", 100);
define("OILPRICE", 10);
define("SPARKPRICE", 4);
```

Добавьте эти строки в код сценария.

Вы должны были заметить, что все имена констант записываются прописными. Это соглашение заимствовано из языка C и оно упрощает зрительное различение переменных и констант. Соблюдать соглашение вовсе не обязательно, тем не менее, следует помнить, что оно облегчает чтение и работу с кодом.

Теперь у нас имеются три константы, которые можно использовать для вычисления общей суммы заказа клиента.

Важное различие между константами и переменными заключается в том, что обращение к константе не требует присутствия перед ней знака доллара. Если требуется использовать значение константы, указывайте только ее имя. Например, для вывода одной из только что созданных констант применяется код:

```
echo TIREPRICE;
```

Наряду с константами, определяемыми пользователем, PHP определяет большое количество собственных констант. Эти константы можно легко просмотреть, выполнив команду **phpinfo():**

```
phpinfo();
```

В результате выводится список предопределенных переменных и констант PHP, а также другая полезная информация. Некоторые из этих вопросов будут рассматриваться по мере изложения материала.

Область действия переменных

Термин *область действия (scope)* относится к разделам сценария, внутри которых видна (доступна) конкретная переменная. В PHP используются следующие три основных типа областей действия:

- Глобальные переменные, объявленные в сценарии, видны во всем сценарии, но *не внутри функций.*
- Переменные, использованные внутри функции, являются локальными для данной функции.
- Переменные, использованные внутри функции, которая объявлена как глобальная, относятся к глобальным переменным с такими же именами.

Области действия переменных более подробно изучаются при рассмотрении функций. Пока достаточно заявить, что все используемые переменные по умолчанию будут глобальными.

Знаки операций

Знаки операций — это символы, которые можно использовать для манипуляции значениями и переменными путем выполнения над ними операций. Некоторые из этих операций потребуются для вычисления общей суммы и налога для заказа клиента.

Мы уже упоминали две операции: операцию присваивания (=) и операцию конкатенации строк (.) Теперь давайте ознакомимся с полным списком операций.

В общем случае операции могут выполняться над одним, двумя и тремя аргументами, причем большинство из них выполняется над двумя аргументами. Например, операция присваивания принимает два аргумента — адрес, указываемый слева от символа =, и выражение, указываемое справа. Эти аргументы называются *операндами,* т.е. элементами, с которыми должна выполняться операция.

Арифметические операции

Арифметические операции очень просты — это обычные математические операции. Они перечисляются в табл. 1.1.

Таблица 1.1 Арифметические операции PHP

Знак операции	*Название*	*Пример*
+	**Сложение**	**$a + $b**
-	**Вычитание**	**$a - $b**
*	**Умножение**	**$a * $b**
/	**Деление**	**$a / $b**
%	**Взятие модуля**	**$a % $b**

Для каждой из этих операций можно сохранять результат выполнения операции, например:

```
$result = $a + $b;
```

Сложение и вычитание работают так, как и ожидается. Результатом их выполнения является, соответственно, сумма и разность значений, хранящихся в переменных $a и $b.

Символ вычитания — можно использовать и в качестве унарной операции (т.е. операции, которая принимает один аргумент, или операнд) для указания отрицательных чисел. Например,

```
$a = -1;
```

Умножение и деление также работают как обычно. Обратите внимание на использование звездочки вместо традиционного математического символа умножения и наклонной черты вместо обычного символа деления.

Операция взятия модуля возвращает остаток от деления переменной Sa на переменную $b. Рассмотрим следующий фрагмент кода:

```
$a = 27;
$b = 10;
$result = $a%$b;
```

Значение, сохраненное в переменной Sresult, представляет собой остаток от деления 27 на 10, т.е. 7. Следует обратить внимание, что арифметические операции обычно применяются к целым числам или значениям с двойной точностью. В случае применения их к строкам PHP все же попытается их выполнить, преобразуя строки в числа. При наличии символов "e" или "E" строка преобразуется в значение двойной точности. В противном случае она преобразуется в целочисленное значение. PHP выполняет поиск цифр в начале строки и найденные цифры использует в качестве значения; если строка не содержит цифр, ее значением будет нуль.

Строковые операции

Мы уже встречали и использовали одну строковую операцию. Операцию конкатенации строк можно использовать для сложения двух строк и генерации и сохранения результата во многом подобно применению операции сложения для получения суммы двух чисел.

```
$a = "Bob's ";
$b = "Auto Parts";
$result = $a.$b;
```

Теперь переменная $result содержит строку "Bob's Auto Parts".

Операции присваивания

Мы уже знакомы с операцией =, основной операцией присваивания. Этот символ всегда означает операцию присваивания и читается как "устанавливается равным". Например,

```
$totalqty = 0;
```

Эта запись должна читаться как "значение переменной **$totalqty** устанавливается равным нулю". Причина этого станет понятна при рассмотрении операций сравнения далее в этой главе.

Возврат значений операций присваивания

В результате использования операции присваивания возвращается итоговое значение, как и в других операциях. Если записать

```
$a + $b
```

то значением этого выражения будет результат сложения переменных $a и **$b**. Аналогично, можно записать

```
$a = 0;
```

Значение всего этого выражения равно 0.

Это позволяет выполнять действия наподобие следующего:

```
$b = 6 + ($a = 5);
```

В результате значение переменной **$b** устанавливается равным 11. Это справедливо для всех операторов присваивания: значением всего оператора присваивания является значение, присвоенное левому операнду.

При оценке значения выражения скобки применяются с целью повышения приоритета подвыражения, как это делалось в приведенном примере. Скобки работают точно так же, как и в математике.

Комбинация операций присваивания

Кроме простых операций присваивания существует набор комбинированных операций присваивания. Каждая из них представляет собой сокращенную форму записи какой-либо другой операции с переменной и присвоения результата этой переменной. Например

```
$a += 5;
```

Это эквивалентно записи

```
$a = $a + 5;
```

Объединенные операции присваивания существуют для каждой из арифметических операций и для операции конкатенации строк.

Перечень всех объединенных операций присваивания и результата их действия приведен в табл. 1.2.

Таблица 1.2 Объединенные операции присваивания PHP

Символ операции	*Использование*	*Эквивалентная операция*
+=	$a += $b	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b	$a = $a * $b
/=	$a /= $b	$a = $a / $b
%=	$a %= $b	$a = $a % $b
.=	$a .= $b	$a = $a . $b

Префиксный и суффиксный инкремент и декремент

Операции префиксного и суффиксного инкремента (++) и декремента (--) аналогичны операциям += и -=, но с несколькими отличиями.

Все операции инкремента оказывают двойное действие — они увеличивают и присваивают значение. Давайте рассмотрим следующее:

```
$a=4;
echo ++4a;
```

Во второй строке используется операция префиксного инкремента, называемая так потому, что символ ++ записывается перед $a. В результате сначала значение $a увеличивается на 1, а затем возвращается увеличенное значение. В данном случае значение $a увеличивается до 5, а затем 5 возвращается и выводится. Значением всего этого выражения будет 5. (Обратите внимание, что фактическое значение, хранящееся в переменной $a, изменяется: результат выполненных действий не ограничивается простым возвратом значения выражения $a + 1.)

Однако, если символ ++ записывается после $a, используется операция суффиксного инкремента, которая приводит к другому результату. Рассмотрим следующие строки:

```
$a=4;
echo $a++;
```

В этом случае действия выполняются в обратном порядке. То есть, вначале значение $a возвращается и выводится, а затем увеличивается на 1. Значением всего этого выражения является 4. Именно это значение и будет выведено. Однако после выполнения этого оператора значение переменной $a равно 5.

Несложно догадаться, что операция — действует аналогично, только здесь значение $a уменьшается на 1, а не увеличивается.

Ссылки

Новой в PHP 4 является операция ссылки & (амперсанд), которая может использоваться в сочетании с операцией присваивания. Обычно, когда переменная присваивается другой переменной, создается копия первой переменной, которая сохраняется где-либо в памяти. Например,

```
$a = 5;
$b = $a;
```

Эти строки кода создают вторую копию значения $a и сохраняют ее в переменной $b. Если впоследствии значение $a изменить, значение $b не изменится:

```
$a = 7; // значение $b будет по-прежнему равно 5
```

Создания копии можно избежать, используя операцию ссылки &, например:

```
$a = 5;
$b = &$a;
$a = 7; // Теперь оба значения $a и $b равны 7
```

Операции сравнения

Операции сравнения используются для сравнения двух значений. Выражения, в которых используются эти операции, возвращают в зависимости от результата сравнения логические значения **true (истинно)** или **false (ложно).**

Операция равенства

Операция равенства == (два знака равенства) позволяет проверить равенство двух значений. Например, выражение

```
$a == $Ь
```

можно было использовать для проверки равенства значений, хранящихся в переменных $a и $Ь. Результатом этого выражения будет **true**, если они равны, или **false,** если они не равны.

Эту операцию легко спутать с операцией присваивания. Это не приведет к выводу сообщения об ошибке, но в общем случае не даст желаемого результата. В общем случае ненулевые значения интерпретируются как **true,** а нулевые — как **false.** Предположим, что две переменных были инициализированы следующим образом:

```
$a = 5;
$Ь = 7;
```

Если затем выполнить проверку $a = **$b**, результатом будет значение **true.** Почему? Да потому что значением выражения $a = **$b** является значение, присвоенное левому операнду, которое в данном случае равно 7. Это — ненулевое значение, поэтому выражение интерпретируется как **true.** Если же в действительности нужно было выполнить проверку $a == $Ь, которая дает в результате **false,** значит, в коде появилась логическая ошибка, которую исключительно трудно обнаружить. В этой связи всегда следует проверять правильность использования этих двух операций.

Подобного рода ошибку очень легко допустить, и, вероятно, читатели не будут исключением. Посему будьте внимательными!

Другие операции сравнения

PHP поддерживает также ряд других операций сравнения, которые перечислены в табл. 1.3.

Таблица 1.3 Операции сравнения PHP

Символ операции	*Название*	*Использование*
==	равно	$a == $Ь
===	идентично	$a === $Ь
!=	не равно	$a != $Ь
O	не равно	$a O $Ь
<	меньше	$a < $Ь
>	больше	$a > $Ь
<=	меньше или равно	$a <= $Ь
>=	больше или равно	$a >= $Ь

Следует обратить внимание на операцию идентичности, появившуюся в PHP 4, которая возвращает значение **true** только в том случае, если оба операнда равны и имеют один и тот же тип.

Логические операции

Логические операции используются для объединения результатов логических условий. Например, нас может интересовать случай, когда значение переменной $a находится в диапазоне между 0 и 100. В этом случае следовало бы проверить условия $a >=0 и $a <= 100, используя операцию **AND (И),** как в следующем примере:

```
$a >= 0 && $a <=100
```

PHP поддерживает логические операции **AND (И), OR (ИЛИ), XOR (исключающее ИЛИ) и NOT (НЕ).**

Перечень логических операций и описание их применения приводится в табл. 1.4.

Таблица 1.4 Логические операции PHP

Символ операции	*Название*	*Использование*	*Результат*
!	НЕ	!$b	Возвращается **true,** если значение $a равно **false,** и наоборот
&&	И	$a && $Ь	Возвращается **true,** если обе переменные $a и $Ь имеют значения **true;** в противном случае возвращается **false**
||	ИЛИ	$a || $b	Возвращается **true,** если любая из переменных $a или $Ь или обе имеют значение **true;** иначе возвращается **false**
and	И	$a and $Ь	**Та же операция, что и &&, но с меньшим приоритетом**
or	ИЛИ	$a or $Ь	**Та же операция, что и ||, но с меньшим приоритетом**

Операции and и or обладают меньшим приоритетом, нежели операции && и ||. Приоритеты более подробно рассматриваются далее в этой главе.

Поразрядные операции

Поразрядные операции позволяют обрабатывать целые числа как последовательности представляющих их разрядов.

Вероятно, в PHP эти операции придется использовать не особенно часто, тем не менее, их перечень приведен в табл. 1.5.

Таблица 1.5 Поразрядные операции PHP

Символ операции	*Название*	*Использование*	*Результат*
&	поразрядное И	$a & $b	Разряды, установленные в единичные состояния $a и $b, устанавливаются в единичные состояния в результате
|	поразрядное ИЛИ	$a | $b	Разряды, установленные в единичные состояния в $a или $b, устанавливаются в единичные состояния в результате
~	поразрядное НЕ	~$a	Разряды, установленные в единичные состояния $a, устанавливаются в нулевые состояния в результате, и наоборот
^	поразрядное исключающее ИЛИ	$a ^ $b	Разряды, установленные в единичные состояния в $a или $b, но не в обеих переменных, устанавливаются в единичные состояния в результате
<<	сдвиг влево	$a << $b	Разряды в переменной $a сдвигаются влево на $b позиций
>>	сдвиг вправо	$a >> $b	Разряды в переменной $a сдвигаются вправо на $b позиций

Другие операции

Помимо описанных, существует также ряд других операций.

Символ запятой (,) используется для разделения аргументов функций и элементов других списков. Обычно он применяется по мере необходимости.

Две специальных операции new и -> используются, соответственно, для создания экземпляра класса и для доступа к членам класса. Подробнее эти операции рассматриваются в главе 6.

Операции работы с массивами [] позволяют получить доступ к элементам массива. Они будут рассматриваться в главе 3.

Существуют еще три операции, которые кратко упоминаются в этой главе.

Тернарная операция

Эта операция, ?:, работает точно так же, как в C. Она записывается в форме

```
условие ? значение, если условие истинно : значение, если условие ложно
```

Тернарная операция аналогична исследуемому далее в главе оператору **if-else**, записанному в виде выражения.

Ниже приведен простой пример:

```
($grade > 50 ? "Выдержал" : "Не выдержал");
```

Это выражение интерпретирует результаты сдачи экзаменов студентом ($grade) как "Выдержал" или "Не выдержал".

Операция подавления ошибки

Операция подавления ошибки @ может использоваться перед любым *выражением*, т.е. любой записью, которая генерирует или имеет значение.

Например,

```
$a = @(57/0);
```

Без символа операции @ эта строка будет генерировать предупреждение о делении на нуль (можете это проверить). При использовании операции @ вывод сообщения об ошибке подавляется.

При таком подавлении сообщений об ошибках необходимо создать некоторый код обработки ошибок с целью проверки на предмет генерации предупреждений. Если PHP установлен с активизированной функцией **track_errors,** сообщение об ошибке будет сохраняться в глобальной переменой **$php_errormsg**.

Операция запуска на выполнение

В действительности символ операции выполнения представляет собой пару символов операций — пару обратных одинарных кавычек (``). Их не следует путать с одинарными кавычками — обычно они вводятся нажатием той же клавиши, что и символ ~ (тильда).

Все, введенное между обратными одинарными кавычками, PHP будет пытаться запустить как команду, вводимую в командной строке сервера. Вывод команды будет значением выражения.

Например, в среде UNIX-подобных операционных систем можно использовать

```
$out = `ls -la`;
echo "<pre>".$out."</pre>";
```

На сервере Windows этим строкам эквивалентны строки

```
$out = `dir c:';
echo "<pre>".$out."</pre>";
```

Любая из версий этого кода получит листинг каталога и сохранит его в переменной **$out**. Затем его можно вывести в окне браузера или обработать любым требуемым образом.

Существуют и другие способы выполнения команд на сервере. Они будут рассматриваться в главе 16.

Использование операций: вычисление итога по форме

Теперь, когда читатели узнали, как использовать операции PHP, можно вычислить итоговую сумму и налог для формы заказа компании Боба.

Для этого в нижнюю часть сценария PHP потребуется добавить следующий код:

```
$totalqty = $tireqty + $oilqty + $sparkqty;
$totalamount =  $tireqty * TIREPRICE + $oilqty * OILPRICE
              + $sparkqty * SPARKPRICE;
$totalamount = number_format($totalamount, 2);
echo "<br>\n";
echo "Items ordered:          ".$totalqty."<br>\n";
echo "Subtotal:               $".$totalamount."<br>\n";
$taxrate = 0.10;  // местный налог с продаж составляет 10%
$totalamount = $totalamount * (1 + $taxrate);
$totalamount = number_format($totalamount, 2);
echo "Total including tax:  $".$totalamount."<br>\n";
```

После обновления окна браузера оно должно выглядеть подобно показанному на рис. 1.5.

Как видите, в этом фрагменте кода задействованы несколько операций. Операции сложения (+) и умножения (*) используются для вычисления числовых значений, а операция конкатенации строк (.) — для форматирования вывода в окне браузера.

Кроме того, при помощи функции **number_format()** выполнялось форматирование итоговых сумм в виде строк с двумя десятичными знаками. Эта функция входит в состав библиотеки математических функций PHP.

При внимательном рассмотрении произведенных вычислений может возникнуть вопрос, почему они были организованы именно в таком порядке. Например, рассмотрим следующую строку:

```
$totalamount = $tireqty * TIREPRICE
             + $oilqty * OILPRICE
             + $sparkqty * SPARKPRICE;
```

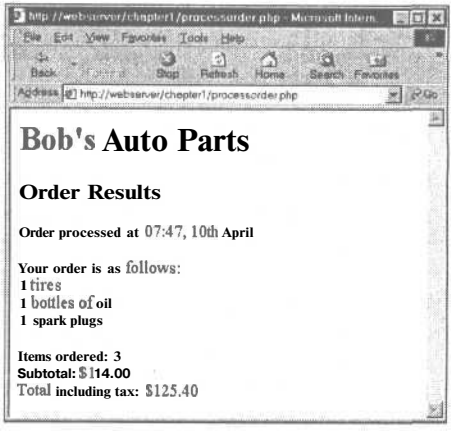


РИСУНОК 1,5 *Общая сумма заказа клиента вычислена и отображена в сформатированном виде.*

Итог кажется правильным, но почему умножение выполнилось перед сложением? Это обусловлено *приоритетом* операций, т.е. порядком их выполнения.

Приоритет и ассоциативность: вычисление выражений

В общем случае операции обладают приоритетами, или порядком их вычислений.

Кроме того, операциям присуща ассоциативность, определяющая порядок выполнения операций с одинаковым приоритетом. В общем случае операции могут выполняться слева направо, справа налево, либо же их порядок не имеет значения.

Приоритеты и ассоциативность операций в PHP перечислены в табл. 1.6.

Таблица 1.6 Приоритеты операций в PHP

Ассоциативность	*Операции*
слева направо	,
слева направо	or
слева направо	xor
слева направо	and
справа налево	prinf
слева направо	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
слева направо	? :
слева направо	||
слева направо	&&
слева направо	|
слева направо	^
слева направо	&
не определена	== != ===

Ассоциативность	Операции
не определена	< <= > >=
слева направо	<< >>
слева направо	+— .
слева направо	* / %
справа налево	! ~ ++— (int) (double) (string) (array) (object) @
справа налево	[]
не определена	new
не определена	()

В этой таблице операции, имеющие самый низкий приоритет, приводятся в верхней части, *а* приоритеты возрастают сверху вниз.

Обратите внимание, что наивысшим приоритетом обладает операция, которую мы еще не рассматривали: старые добрые круглые скобки. Они повышают приоритет любого заключенного в них выражения. Именно с их помощью в случае необходимости можно изменять правила приоритета.

Вспомните следующий фрагмент из последнего примера:

```
$totalamount = $totalamount * (1 + $taxrate);
```

Если бы мы записали

```
$totalamount = $totalamount * 1 + $taxrate;
```

то операция умножения, имеющая более высокий приоритет по сравнению с операцией сложения, выполнялась бы первой, что привело бы к неверному результату. Используя круглые скобки, можно добиться, чтобы вначале вычислялось подвыражение **1** + **$taxrate**.

В выражении можно использовать любое требуемое количество наборов круглых скобок. При этом первым будет вычисляться выражение, заключенное в самые внутренние скобки.

Функции для работы с переменными

Прежде чем покинуть мир переменных и операций, давайте рассмотрим функции для работы с переменными PHP. Это библиотека функций, позволяющая различными способами манипулировать переменными и проверять их.

Проверка и установка типов переменных

Большинство из этих функций связано с проверкой типов.

Двумя наиболее общими функциями являются **gettype()** и **settype()**. Они имеют следующие прототипы, определяющие ожидаемые аргументы и возвращаемые значения.

```
string gettype(mixed var);
int settype(string var, string type);
```

При вызове функции **gettype()** ей необходимо передать переменную. Функция определяет тип переменой и возвращает строку, содержащую имя типа или **"unknown type"**, если тип переменной не является одним из стандартных типов: **integer**, **double**, **string**, **array** или **object**.

При вызове функции **settype()** ей необходимо передать переменную, тип которой требуется изменить, и строку, содержащую новый тип переменной из приведенного ранее списка.

Пример применения этих функций показан ниже:

```
$a = 56;
echo gettype($a)."<br>";
settype ($a, "double");
echo gettype($a)."<br>";
```

Перед первым вызовом функции **gettype()** переменная $a имеет тип **integer.** После вызова функции **settype()** ее тип изменится на **double.**

PHP предоставляет также ряд функций проверки типов. Каждая из этих функций принимает переменную в качестве аргумента и возвращает значение **true** или **false.** Вот их перечень:

- **is_array()**
- **is_double(), is_float(), is_real()** (Это одна и та же функция)
- **is_long(), is_int(), is_integer()** (Это одна и та же функция)
- **is_string()**
- **is_object()**

Проверка состояния переменных

PHP имеет несколько функций, предназначенных для проверки состояния переменных.

Первая из них — **isset()**, имеющая следующий прототип:

```
int isset(mixed var);
```

Эта функция принимает в качестве аргумента имя переменной и возвращает значение **true,** если переменная существует, и **false** в противном случае.

Переменную можно удалить, используя сопутствующую функцию **unset()**. Она имеет следующий прототип:

```
int unset(mixed var);
```

Эта функция удаляет переменную и возвращает значение **true.**

И наконец, существует функция **empty()**. Она проверяет существование переменной и наличие у нее непустого, ненулевого значения, возвращая, соответственно, значение **true** или **false.** Эта функция имеет следующий прототип:

```
int empty(mixed var);
```

Рассмотрим пример использования этих трех функций.

Попытайтесь временно добавить в сценарий следующие строки кода:

```
echo isset($tireqty);
echo isset($nothere);
echo empty($tireqty);
echo empty($nothere);
```

Обновите страницу, чтобы увидеть результат.

Функция **isset()** должна возвращать для переменной **$tireqty** значение **true** вне зависимости от того, введено или нет значение в поле формы. Возвращаемое значение функции **empty()** зависит от введенного значения.

Переменная **$nothere** не существует, поэтому для нее функция **isset()** будет генерировать значение **false,** а функция **empty() — true**.

Эти функции могут оказаться полезными при проверке заполнения пользователем соответствующих полей формы.

Повторная интерпретация переменных

Можно достичь эффекта, эквивалентного приведению переменной, при помощи функции. Для этого применяются три функции:

```
int intval(mixed var);
double doubleval(mixed var);
string strval(mixed var);
```

Каждая из них принимает в качестве аргумента переменную и возвращает значение переменной, преобразованное к соответствующему типу.

Управляющие структуры

Управляющие структуры — это языковые структуры, которые позволяют управлять потоком выполнения программы или сценария. Их можно разделить на условные структуры (или структуры ветвления) и структуры повторения, или циклы. Далее рассматриваются характерные реализации каждого вида структур в PHP.

Принятие решений с помощью условных операторов

Дабы обеспечить своевременную реакцию на вводимую пользователем информацию, код должен быть в состоянии принимать решения. Конструкции, которые указывают программе о необходимости принятия решений, называются *условными операторами.*

Операторы if

Для принятия решений применяется оператор **if.** Чтобы его использовать, ему необходимо передать условие. Если условие имеет значение **true,** будет выполняться следующий за ним блок кода. Условие в операторе **if** должно заключаться в круглые скобки ().

Например, если форма заказа у Боба не содержит ни шин, ни бутылок с маслом, ни свечей, вероятно, это связано со случайным нажатием кнопки Submit. Вместо сообщения "Order processed" ("Заказ обработан"), на странице стоило бы отобразить более полезное сообщение.

Если посетитель вообще не заказывает запчастей, вероятно, имеет смысл вывести сообщение "You did not order anything on previous page!" ("Вы ничего не заказали на предыдущей странице!"). Это легко достигается с помощью следующего оператора **if:**

```
if( $totalqty == 0 )
      echo "You did not order anything on previous page!<br>";
```

В этом операторе используется условие **$totalqty == 0.** Помните, что операция равенства (==) ведет себя иначе, нежели операция присваивания (=).

Условие **$totalqty == 0** будет иметь значение **true,** если значение переменной **$totalqty** равно нулю. Если значение переменной **$totalqty** не равно нулю, значение

условия будет равно **false.** Когда значением условия будет **true,** оператор **echo** выполнится.

Блоки кода

Часто внутри такого условного оператора, как **if,** требуется выполнить более одного оператора. Перед каждым из них вовсе не обязательно помещать новый оператор **if.** Вместо этого последовательность операторов можно сгруппировать в блок. Для объявления блока операторы должны помещаться в фигурные скобки:

```
if( $totalqty == 0 )
{
   echo "<font color=red>";
   echo "You did not order anything on the previous page!<br>";
   echo "</font>";
}
```

Теперь три строки кода, заключенные в фигурные скобки, являются блоком. Когда значением условия будет **true,** выполняются все три строки. Если значением условия будет **false**, все три строки будут игнорироваться.

Примечание: выравнивание кода при помощи отступов

Как уже отмечалось, выравнивание кода в PHP значения не имеет. Отступы требуются только для повышения читабельности. В основном, они используются для того, чтобы с первого взгляда было видно, какие строки будут выполняться при каких условиях, какие операторы сгруппированы в блоки и какие операторы являются частью циклов или функций. В приведенных ранее примерах операторы, выполнение которых зависит от оператора if, и операторы, образующие блок, были записаны с отступами.

Операторы else

Часто требуется принимать решение не только о выполнении того или иного действия, но и выбирать определенный набор возможных действий в противном случае.

Оператор **else** позволяет определить альтернативное действие, которое должно выполняться, если значением условия в операторе if окажется **false.** В рассматриваемом примере необходимо предупреждать клиентов, когда они ничего не заказывают. С другой стороны, если они делают заказ, вместо предупреждения требуется вывести список заказанного.

Если немного изменить код и добавить в него оператор **else,** можно отображать либо предупреждение, либо итоговую информацию.

```
if( $totalqty = 0 )
{
   echo "You did not order anything on the previous page!<br>";
}
else
{
   echo $tireqty." tires<br>";
   echo $oilqty." bottles of oil<br>";
   echo $sparkqty." spark plugs<br>";
}
```

Вкладывая операторы **if** один в другой, можно строить более сложные логические процессы. Следующий код не только обеспечит отображение итоговой информации, когда значение условия **$totalqty == 0** равно **true,** но и отображение каждой из итоговых строк только при выполнении ее собственного условия.

```
if( $totalqty == 0)
{
   echo "You did not order anything on the previous page!<br>";
}
else
{
   if ( $tireqty>0 )
      echo $tireqty." tires<br>";
   if ( $oilqty>0 )
      echo $oilqty." bottles of oil<br>";
   if ( $sparkqty>0 )
      echo $sparkqty." spark plugs<br>";
}
```

Операторы *elseif*

Для многих принимаемых решений может существовать более двух возможностей. Последовательность множества вариантов действий создается с использованием оператора **elseif,** который представляет собой комбинацию операторов **else** и **if.** При наличии последовательности условий программа может проверять каждое из них до тех пор, пока не отыщет то, значением которого является **true**.

Боб предоставляет скидки при заказе большого количества автопокрышек. Схема скидок выглядит следующим образом:

- Приобретение менее 10 автопокрышек — без скидки
- Приобретение 10-49 автопокрышек — скидка 5%
- Приобретение 50-99 автопокрышек — скидка 10%
- Приобретение 100 и более автопокрышек — скидка 15%

Можно создать код для вычисления скидок с использованием условий и операторов **if** и **elseif.** Для объединения двух условий в одно применяется операция И (&&).

```
if( $tireqty < 10 )
   $discount = 0;
elseif( $tireqty >= 10 &S $tireqty <= 49 )
   $discount = 5;
elseif( $tireqty >= 50 SS $tireqty <= 99 )
   $discount = 10;
elseif( $tireqty > 100 )
   $discount = 15;
```

Обратите внимание, что можно применять как **elseif, так и else if** — оба варианта правильны.

При использовании каскадных наборов операторов **elseif** следует помнить, что будет выполняться только один из блоков операторов. В данном примере это не важно, поскольку все условия являются взаимоисключающими — в каждый момент времени может выполняться только одно из них. Если бы условия были записаны так, что одновременно могло бы выполняться несколько условий, выполнялся бы только блок или оператор, следующий за первым истинным условием.

Операторы switch

Оператор **switch** работает аналогично оператору **if,** но позволяет условному выражению иметь в качестве результата более двух значений. В операторе **if** условие принимает значение **true** или **false.** В операторе **switch** условие может принимать любое количество

различных значений до тех пор, пока оно имеет простой тип (integer, string или double). Для обработки каждого из значений, на которые требуется реагировать, необходимо записать оператор **case,** а также (необязательно) можно определить случай для обработки по умолчанию любых значений, для которых специально не задан оператор **case.**

Боб желает знать, какие формы рекламы работают на него наиболее эффективно. Для этого в форму заказа можно добавить вопрос.

Добавьте в форму следующий HTML-код, после чего она должна выглядеть так, как показано на рис. 1.6:

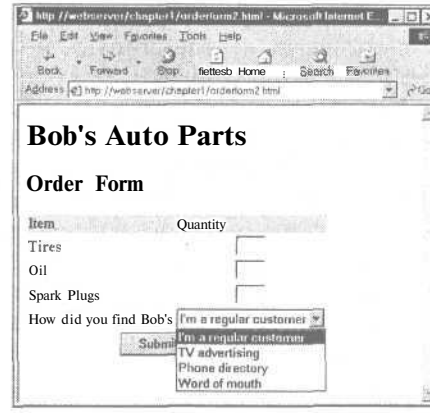


РИСУНОК 1.6 *Теперь форма заказа запрашивает посетителей о том, каким образом они нашли сайт Bob's Auto Parts.*

```
<tr>
  <td>How did you find Bob's</td>
  <tdxselect name="find">
        <option value = "a">I'm a regular customer
        <option value = "b">TV advertising
        <option value = "c">Phone directory
        <option value = "d">Word of mouth
      </select>
  </td>
</tr>
```

Приведенный HTML-код добавил новую переменную формы, значением которой будет "a", "b", "c" или "d". Эту новую переменную можно было бы обработать с помощью последовательности операторов **if** и **elseif:**

```
if($find == "a")
  echo "<P>Regular customer.";
elseif($find == "b")
  echo "<P>Customer referred by TV advert.";
elseif($find == "c")
  echo "<P>Customer referred by phone directory.";
elseif($find == "d")
  echo "<P>Customer referred by word of mouth.";
```

Либо же можно было бы записать оператор **switch:**

```
switch($find)
{
  case "a" :
    echo "<P>Regular customer.";
    break;
  case "b" :
    echo "<P>Customer referred by TV advert.";
    break;
  case "c" :
    echo "<P>Customer referred by phone directory.";
    break;
  case "c" :
    echo "<P>Customer referred by word of mouth.";
    break;
  default :
    echo "<P>We do not know how this customer found us." ;
    break;
}
```

Этот оператор **switch** ведет себя несколько иначе, нежели оператор **if** или **elseif.** Оператор **if** оказывает влияние только на один оператор, если только специально не использовать фигурные скобки для создания блока операторов. Оператор **switch** действует по-другому. Когда оператор **case** в операторе **switch** активизируется, операторы выполняются до тех пор, пока не встретится оператор **break.** Без него оператор **switch** выполнял бы весь код, следующий за оператором **case,** условие которого было истинным. При достижении оператора break будет выполняться строка кода, следующая за оператором **switch.**

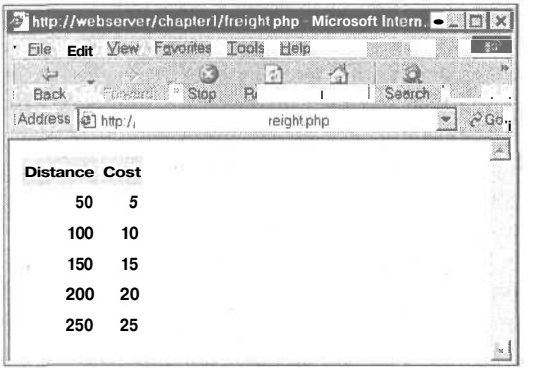


РИСУНОК 1.7 *В этой таблице показана стоимость доставки в зависимости от расстояния.*

Сравнение различных условных операторов

У читателей, не знакомых с этими операторами, может возникнуть вопрос "Какой же из них наилучший?".

На этот вопрос невозможно дать однозначный ответ. Один или несколько операторов **else, elseif** или **switch** не позволяют сделать ничего такого, чего нельзя было бы сделать с помощью набора операторов if. В каждой конкретной ситуации следует использовать именно те условные операторы, которые выглядят наиболее читабельно. Утверждение станет более понятным по мере приобретения практического опыта.

Итерация: повторение действий

Одна из задач, с которыми компьютеры всегда справлялись очень успешно — автоматизация повторяющихся действий. Если что-то требуется выполнять многократно и одинаково, для повторения определенных частей программы стоит применить цикл.

Бобу требуется таблица, отображающая стоимость доставки, которая будет добавляться к стоимости заказа клиента. Стоимость доставки зависит от расстояния и может быть вычислена с помощью простой формулы.

Таблица стоимости доставки показана на рис. 1.7.

HTML-код, отображающий эту таблицу, приведен в листинге 1.3. Как видите, он достаточно длинен, причем многие его фрагменты повторяются.

Листинг 1.3 freight.html — HTML-код для таблицы стоимости доставки в компании Боба

```
<html>
<body>
<table border = 0 cellpadding = 3>
<tr>
   <td bgcolor = "#CCCCCC" align = center>Distance</td>
   <td bgcolor = "#CCCCCC" align = center>Cost</td>
</tr>
<tr>
   <td align = right>50</td>
   <td align = right>5</td>
</tr>
```

```
<tr>
   <td align = right>100</td>
   <td align = right>10</td>
</tr>
<tr>
   <td align = right>150</td>
   <td align = right>15</td>
</tr>
<tr>
   <td align = right>200</td>
   <td align = right>20</td>
</tr>
<tr>
   <td align = right>250</td>
   <td align = right>25</td>
</tr>
</table>
</body>
</html>
```

Вместо того чтобы поручать ввод HTML-кода человеку, которому выполнение подобной задачи быстро надоедает и которому, к тому же, необходимо платить за затраченное время, было бы хорошо, если бы это мог делать компьютер.

Операторы цикла указывают PHP о необходимости многократного выполнения оператора или блока операторов.

Циклы while

Простейший вид цикла в PHP — цикл **while.** Подобно оператору **if,** этот оператор основан на проверке условия. Различие между циклом **while** и оператором **if** состоит в том, что если условие принимает значение **true,** оператор **if** выполняет следующий за ним блок кода только один раз. Цикл **while** выполняет блок операторов многократно до тех пор, пока условие принимает значение **true.**

В общем случае цикл **while** используется, когда не известно, сколько итераций потребуется для выполнения условия. Если же требуется фиксированное количество итераций, стоит подумать об использовании цикла **for.**

Основная структура цикла **while** имеет вид

```
while( условие ) выражение;
```

Следующий цикл **while** отображает числа от 1 до 5.

```
$num = 1;
while ($num <= 5 )
{
   echo $num."<BR>";
   $num++;
}
```

Условие проверяется в начале каждой итерации. Если оно принимает значение **false,** блок не будет выполняться и цикл завершается. После этого выполняется оператор, следующий за циклом.

Цикл **while** можно использовать для выполнения чего-то более полезного, например для отображения повторяющейся таблицы стоимости доставки, показанной на рис. 1.7.

В листинге 1.4 цикл **while** используется для генерации таблицы стоимости доставки.

Листинг 1.4 freight.php — генерация таблицы стоимости доставки компании Боба с помощью PHP

```
<bodyx
<table border = 0 cellpadding = 3>
<tr>
   <td bgcolor = "#CCCCCC" align = center>Distance</td>
   <td bgcolor = "#CCCCCC" align = center>Cost</td>
</tr>
<?
$distance = 50;
while  ($distance <= 250  )
{
   echo "<tr>\n  <td align = right>$distance</td>\n";
   echo "   <td align = right>". $distance / 10  ."</td>\n</tr>\n";
   $distance += 50;
}
?>
</table>
</body>
</html>
```

Циклы for

Описанный выше способ использования циклов **while** очень распространен. Сначала устанавливается начальное значение счетчика. Перед каждой итерацией значение счетчика проверяется в условии. В конце каждой итерации значение счетчика изменяется.

При помощи цикла **for** такого рода цикл можно записать в более компактной форме.

Основная структура цикла **for** имеет вид

```
for( выражение1; условие; выражение2)
      выражение3;
```

- *выражение1* выполняется один раз в начале цикла. Обычно в нем устанавливается начальное значение счетчика.
- Выражение *условие* проверяется перед каждой итерацией. Если выражение возвращает значение **false,** итерация прекращается. Обычно в нем значение счетчика сравнивается с предельным значением.
- *выражение2* выполняется в конце каждой итерации. Обычно в нем изменяется значение счетчика.
- *выражение3* выполняется один раз для каждой итерации. Обычно это выражение представляет собой блок кода и содержит собственно тело цикла.

Пример цикла **while,** приведенный в листинге 1.4, можно переписать с применением цикла **for.** PHP-код примет следующий вид:

```
<?
for ($distance = 50;  $distance <= 250;  $distance += 50)
{
   echo "<tr>\n  <td align = right>$distance</td>\n";
   echo "   <td align = right>". $distance / 10  ."</td>\n</tr>\n";
}
?>
```

В функциональном смысле циклы **while** и **for** идентичны. Однако цикл **for** несколько компактней и содержит на две строки меньше.

Оба эти типа циклов эквивалентны — ни один из них ничем не лучше и не хуже другого. В конкретной ситуации можно использовать тот, который кажется более подходящим.

Попутно отметим, что можно объединять переменные переменных и циклы **for** для организации итераций по последовательности повторяющихся полей формы. Например, при наличии полей формы с именами наподобие **name1**, **name2**, **name3** и т.д. их можно обрабатывать следующим образом:

```
for ($i=1; $i <= $numnames; $i++)
{
   $temp= "name$i";
   echo $$temp."<br>"; // любая требуемая обработка
}
```

Динамически создавая имена переменных, можно обращаться к каждому из полей по очереди.

Циклы do..while

Последний тип циклов, который мы рассмотрим, действует несколько иначе. Общая структура оператора **do..while** имеет вид

```
do
   выражение;
while( условие );
```

Цикл **do..while** отличается от цикла **while** тем, что в нем условие проверяется в конце. Это означает, что в цикле **do..while** оператор или блок внутри цикла выполняется всегда не менее одного раза.

Даже в этом примере, в котором с самого начала условие имеет значение **false** и никогда не может принять значение **true,** цикл выполнится один раз, прежде чем условие будет проверено и цикл завершится.

```
$num = 100;
do
{
   echo $num."<BR>";
}
while ($num< 1 ) ;
```

Выход из управляющей структуры или сценария

В зависимости от эффекта, который требуется получить, для выхода из фрагмента кода можно воспользоваться одним из трех подходов.

Если необходимо прекратить выполнение цикла, можно задействовать оператор **break,** как было описано ранее в разделе, посвященном оператору **switch.** При использовании оператора **break** в цикле выполнение сценария будет продолжаться, начиная со строки сценария, следующей за циклом.

Если требуется перейти к следующей итерации цикла, можно воспользоваться оператором **continue.**

Если требуется завершить выполнение всего PHP-сценария, необходимо применить оператор **exit.** Обычно этот оператор используется при проверке на ошибки. Например, ранее приведенный пример можно было бы изменить следующим образом:

```
if( $totalqty = 0 >
{
   echo "You did not order anything on the previous page!<br>";
   exit;
}
```

Вызов оператора **exit** прекращает выполнение оставшейся части PHP-сценария.

Что дальше: сохранение заказа клиента

Теперь уже известно, как получить и манипулировать заказом клиента. В следующей главе рассматриваются вопросы сохранения заказа, дабы впоследствии его можно было прочитать и выполнить.

Глава 2

Хранение и получение данных

Теперь, когда мы научились получать доступ и манипулировать данными, введенными в HTML-форму, можно рассмотреть способы хранения этой информации с целью дальнейшего использования. В большинстве случаев, включая и пример, рассмотренный в предыдущей главе, данные необходимо сохранять и загружать впоследствии. В данном случае потребуется записывать формы заказов клиентов на устройство хранения так, чтобы затем их можно было окончательно заполнить.

В этой главе мы рассмотрим, как созданную в предшествующем примере форму заказа клиента можно записать в файл и затем прочитать ее из файла. Мы рассмотрим также, почему это решение не всегда является наилучшим. При наличии большого количества заказов стоит воспользоваться системой управления базами данных, подобную MySQL.

В этой главе рассматриваются следующие основные темы:

- Сохранение данных с целью дальнейшего использования
- Открытие файла
- Создание файла и запись в файл
- Закрытие файла
- Считывание из файла
- Блокирование файла
- Удаление файлов
- Другие полезные файловые функции
- Более рациональный способ обработки: системы управления базами данных
- Дополнительная информация

Сохранение данных с целью дальнейшего использования

Существуют два основных способа хранения данных: в двумерных (обычных) файлах и в базах данных.

Двумерный файл может иметь множество форматов, но в общем случае под *двумерным (flat) файлом* будем понимать простой текстовый файл. В рассматриваемом ниже примере заказы клиента записываются в текстовый файл, по одному заказу в каждой строке.

Этот способ столь же прост, сколь и ограничен, как будет показано далее в главе. Если приходится иметь дело с достаточно большим объемом информации, вероятно, лучше воспользоваться базами данных. Однако, двумерные файлы находят достаточно широкое применение, поэтому в ряде ситуаций необходимо владеть технологией их применения.

Запись в файлы и считывание из них в среде PHP совершенно идентичны реализации этих задач в среде C. Если ранее доводилось программировать на языке C или создавать сценарии для оболочки UNIX, этот процесс должен показаться достаточно знакомым.

Сохранение и получение заказов в компании Боба

В этой главе будет использоваться несколько измененная форма заказа, основанная на рассмотренной в предыдущей главе. Мы начнем с этой формы и с PHP-кода, созданного для обработки данных формы.

ПРИМЕЧАНИЕ

Используемые в этой главе HTML- и PHP-сценарии можно найти в папке **chapter2** на сопровождающем CD-ROM.

Мы изменили форму, включив в нее адрес доставки клиента (см. рис. 2.1).

Поле формы, предназначенное для ввода адреса доставки, носит название **address.** В результате мы располагаем переменной, к которой при обработке формы в PHP можно обращаться как к **$address** при условии, что используется сокращенный стиль переменных формы. Помните, что в случае применения длинной формы следовало бы использовать **$HTTP_GET_VARS["address"] или $HTTP_POST_VARS["address"]** (см. главу 1).

Каждый из поступающих заказов записывается в один и тот же файл. Затем создается Web-интерфейс, чтобы служащие компании Боба смогли просматривать полученные заказы.

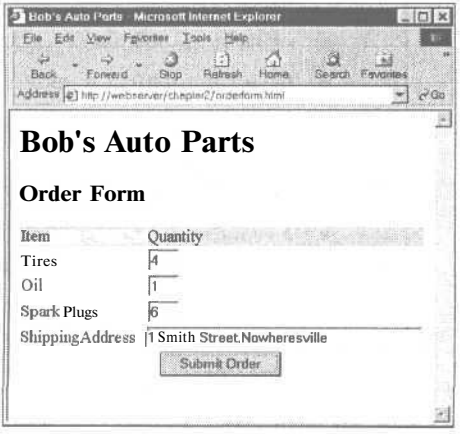


РИСУНОК 2.1
Эта версия формы заказа получает адрес доставки клиента.

Обзор обработки файлов

Запись данных в файл реализуется в три шага:

1. Открытие файла. Если файл еще не существует, его потребуется создать.
2. Запись данных в файл.
3. Закрытие файла.

Аналогично, считывание данных из файла также связано с выполнением трех шагов:

1. Открытие файла. Если файл не может быть открыт (например, он не существует), эта ситуация должна быть распознана и следует предусмотреть корректный выход из нее.
2. Считывание данных из файла.
3. Закрытие файла.

При необходимости считывания данных из файла можно выбирать, какая часть файла должна считываться за один раз. Чуть позже будут подробно рассматриваться все доступные возможности.

Пока давайте исследуем шаг открытия файла.

Открытие файла

Для открытия файла в среде PHP используется функция **fopen()**. При открытии файла необходимо указать, как его предполагается использовать. Это называется *режимом файла*.

Режимы файлов

Серверная операционная система должна знать, что нужно делать с открываемым файлом. Ей требуется знать, может ли файл быть открыт и обработан другим сценарием в то время, когда он является открытым, если владелец сценария имеет право на подобное его использование. По существу, режимы файла предоставляют операционной системе механизм для определения способа обработки запросов на доступ, поступающих от других пользователей или сценариев, а также метод проверки наличия доступа и прав для работы с конкретным файлом.

При открытии файла следует принять три решения:

1. Файл можно открыть только для чтения, только для записи или для чтения и записи.
2. При выполнении записи в файл можно перезаписать любое существующее содержимое файла либо же дописать новые данные в конец файла.
3. При попытке выполнения записи в файл в системе, которая различает двоичные и текстовые файлы, может потребоваться указать тип файла.

Функция **fopen()** поддерживает любые имеющие смысл комбинации этих трех вариантов.

Использование функции fopen() для открытия файла

Давайте предположим, что требуется записать заказ клиента в файл заказов Боба. Этот файл можно открыть для записи с помощью следующего оператора:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "w");
```

Функция **fopen** ожидает двух или трех входных параметров. Обычно используются два параметра, как показано в приведенной выше строке кода.

Первым параметром должен быть файл, который необходимо открыть. При этом можно указать путь к файлу, как было сделано в приведенной выше строке кода - **orders.txt** находится в каталоге **orders.** Мы использовали встроенную переменную **$DOCUMENT_ROOT** PHP. Эта переменная указывает на основание дерева документов Web-сервера. Кроме того, мы использовали символ "..", означающий "родительский каталог каталога **$DOCUMENT_ROOT**. В целях повышения безопасности этот каталог находится вне дерева документов. Нежелательно, чтобы этот файл был доступен в Web помимо предоставляемого нами интерфейса. Этот путь называется относительным, поскольку он описывает позицию в файловой системе относительно **$DOCUMENT_ROOT.**

Можно было бы задать и абсолютный путь к файлу — путь от корневого каталога (/ в системе UNIX и, как правило, C:\ в системе Windows). На сервере UNIX, который используют авторы, такой путь выглядит как **/home/book/orders**. Проблема, связанная с подобным указанием пути, особенно в случае размещения своего сайта на чужом сервере, заключается в том, что абсолютный путь может изменяться. Мы убедились в этом на собственном горьком опыте после того, как пришлось изменять абсолютные пути в большом количестве сценариев, когда системные администраторы без предупреждения "сочли необходимым" изменить структуру каталогов.

Если путь вообще не указан, файл будет создаваться или отыскиваться в том же каталоге, в котором находится собственно сценарий. Упомянутое поведение будет иным при запуске PHP через какую-то CGI-оболочку и зависит от конфигурации сервера.

В среде UNIX в качестве разделителя каталогов используется символ прямой (с уклоном вправо) косой черты (/). На платформах Windows можно применять символы прямой или обратной косой черты. При использовании символа обратной черты они должны быть помечены как специальные (т.е. отменены), чтобы функция **fopen** смогла их корректно интерпретировать. Для отмены перед символом следует просто поместить дополнительный символ обратной косой черты, как показано в следующей строке:

```
$fp = fopen("..\\..\\orders\\orders.txt", "w");
```

Второй параметр функции **fopen()** — это режим файла, который должен иметь строковый тип. Этот параметр определяет, что необходимо делать с файлом. В данном случае в функцию **fopen()** передается параметр "w" — это означает открытие файла для записи. Режимы файла перечислены в табл. 2.1.

Таблица 2,1 Режимы файла для функции fopen

Режим	*Значение*
r	**Режим чтения — Открытие файла для чтения, начиная с начала файла.**
r+	**Режим чтения — Открытие файл для чтения и записи, начиная с начала файла.**
w	**Режим записи — Открытие файла для записи, начиная с начала файла. Если файл уже существует, его содержимое удаляется. Если файл не существует, предпринимается попытка его открытия и в результате файл создается.**
w+	**Режим записи — Открытие файла для записи и чтения, начиная с начала файла. Если файл уже существует, его содержимое удаляется. Если файл не существует, предпринимается попытка его открытия и в результате файл создается.**

Режим	*Значение*
a	**Режим добавления — Открытие файла только для добавления (записи), начиная с конца существующего содержимого, если оно имеется. Если файл не существует, предпринимается попытка его открытия и в результате файл создается.**
a+	**Режим добавления — Открытие файла для добавления (записи) и чтения, начиная с конца существующего содержимого, если оно имеется. Если файл не существует, предпринимается попытка его открытия и в результате файл создается.**
b	**Двоичный режим — Используется в сочетании с одним из остальных режимов. Его указание может потребоваться, если файловая система различает двоичные и текстовые файлы. Операционная система Windows различает эти файлы, а UNIX - нет.**

Режим файла, который необходимо использовать в рассматриваемом примере, зависит от того, как будет использоваться система. Параметр **"w"** позволяет сохранить в файл только один заказ. При приеме каждого нового заказа он будет перезаписывать ранее записанный заказ. Вероятно, это не очень разумно, поэтому лучше указать режим добавления:

```
$fp = fopen("../../orders/orders.txt", "a");
```

Третий параметр функции **fopen()** не является обязательным. Его можно использовать, если файл необходимо искать в пути **include_path** (определенном в конфигурации PHP; см. приложение А). Если это требуется, установите параметр равным 1. В этом случае не следует задавать имя каталога или путь:

```
$fp = fopen("orders.txt", "a", 1);
```

В случае успешного открытия файла функция **fopen()** возвращает указатель на файл и сохраняет его в переменой, в данном случае **$fp**. Эта переменная будет использоваться для доступа к файлу, когда из него потребуется выполнить считывание либо что-то записать в него.

Открытие удаленных файлов через FTP или HTTP

Используя функцию **fopen(),** можно открывать для чтения или записи не только локальные файлы, но и удаленные с использованием протоколов FTP и HTTP.

Если используемое имя файла начинается с ftp://, открывается FTP-соединение с указанным сервером в пассивном режиме и возвращается указатель на начало файла.

Если используемое имя файла начинается с **http://**, открывается HTTP-соединение с указанным сервером и возвращается указатель на ответ от сервера. В случае применения режима HTTP обязательно следует указывать завершающие символы косой черты в именах каталогов, как показано в следующем примере:

```
http://www.server.com/
```

но не

```
http://www.server.com
```

При второй форме указания адреса (без завершающей косой черты) Web-сервер, как правило, будет использовать перенаправление HTTP с целью обращения по первому адресу (с косой чертой). Проверьте это в своем браузере.

Функция **fopen()** не поддерживает перенаправление HTTP, поэтому необходимо указывать URL-адреса (унифицированный локатор ресурсов), которые ссылаются на каталоги с завершающими символами косой черты.

Помните, что имена доменов в URL-адресах не зависят от регистра, однако пути и имена файлов зависят.

Проблемы, возникающие при открытии файлов

Обычная ошибка, связанная с открытием файла — попытка открыть файл, для которого отсутствуют права на чтение или запись. В этом случае PHP выводит соответствующее предупреждение (см. рис. 2.2).

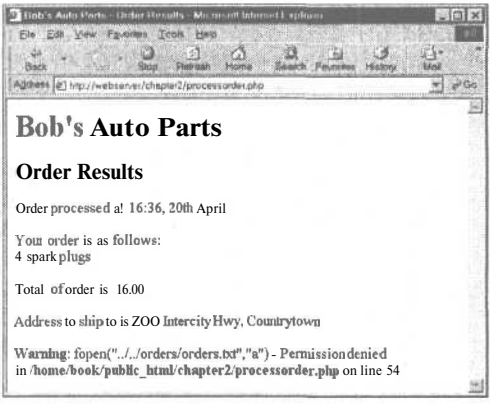


РИСУНОК 2.2 *PHP предупреждает, если файл не может быть открыт.*

В случае получения подобного сообщения об ошибке необходимо убедиться, что пользователь, выполняющий сценарий, обладает правами доступа к файлу, попытка использования которого предпринимается. В зависимости от того, как установлен данный сервер, сценарий может выполняться как пользователь Web-сервера или как владелец каталога, в котором размещается сценарий.

В большинстве систем сценарий будет выполняться как пользователь Web-сервера. Если бы сценарий находился в системе UNIX в каталоге **~/public_html/chapter2/**, общедоступный для записи каталог для хранения заказов можно было бы создать, набрав следующие команды:

```
mkdir ~/orders
chmod 777 ~/orders
```

Имейте в виду, что каталоги, в которых любой пользователь может выполнить запись, представляют опасность. В системе не должно быть каталогов, которые доступны для записи непосредственно из Web. Именно поэтому наш каталог **orders** размещается на два подкаталога выше каталога **public_html**. Подробнее вопросы безопасности рассматриваются в главе 13.

Некорректные настройки прав доступа — вероятно, наиболее часто встречающаяся ошибка во время открытия файла, однако она далеко не единственная. Если файл не может быть открыт, об этом действительно нужно знать, дабы не пытаться считывать или записывать в него данные.

Если обращение к функции **fopen()** оказывается безуспешным, функция возвращает значение false. Обработку ошибок можно сделать более удобной для пользователя, подавив сообщение об ошибке от PHP и реализовав собственное сообщение:

```
@ $fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "a", 1);

  if (!$fp)
  {
    echo "<p><strong> Your order could not be processed at this time. "
      ."Please try again later.</strong></p></body></html>";
    exit;
  }
```

Символ @ перед обращением к функции **fopen()** указывает PHP на необходимость подавления любых сообщений об ошибках, генерируемых после вызова функции. Обычно важно только знать, когда что-то выполняется неправильно, но в данной ситуации в любом случае следует разобраться с подобного рода проблемой. Обратите внимание, что символ @ должен располагаться в самом начале строки. Подробнее сообщения об ошибках описываются в главе 23.

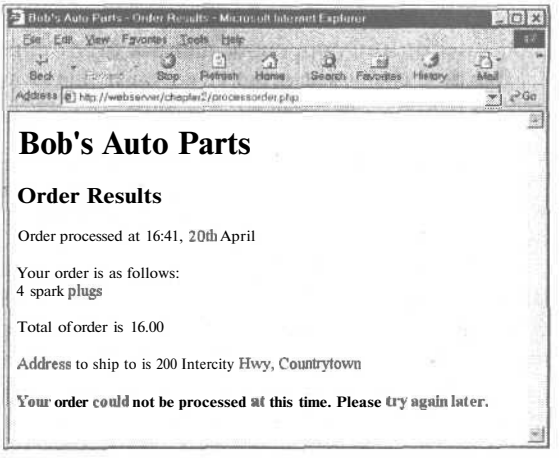


РИСУНОК 2.3 *Использование собственных сообщений об ошибке вместо сообщений, генерируемых PHP, может оказаться более удобным для пользователя.*

Оператор **if** проверяет переменную $fp с целью выяснения, возвращался ли из функции **fopen** допустимый указатель файла, и если нет, выводится сообщение об ошибке и выполнение сценария завершается. Поскольку здесь завершается и страница, обратите внимание на генерацию закрывающего дескриптора </html>, что обеспечивает допустимость HTML-кода.

Вывод, получаемый в результате использования изложенного выше подхода, показан на рис. 2.3.

Запись в файл

Запись в файл в PHP выполняется сравнительно просто. Для этого можно воспользоваться любой из функций **fwrite()** (file write — запись в файл) или **fputs()** (file put string — запись строки в файл); **fputs()** — это псевдоним функции **fwrite()**. Функцию **fwrite()** можно вызвать следующим образом:

```
fwrite($fp, $outputstring);
```

Это указывает PHP на необходимость записи строки из переменной **$outputstring** в файл, указанный **$fp.** Рассмотрим функцию **fwrite()** более подробно, прежде чем приступать к исследованию содержимого переменной **$outputstring**.

Параметры функции fwrite()

В действительности функция **fwrite()** принимает три параметра, однако третий из них не является обязательным. Прототип функции **fwrite()** имеет следующий вид

```
int fputs(int fp, string str, int [length]);
```

Третий параметр ***length*** представляет собой максимальное количество байтов, которые требуется записать. При передаче этого параметра функция **fwrite()** будет записывать строку *str* в файл, указанный параметром ***fp,*** пока не встретит конец строки или не запишет ***length*** байтов, в зависимости от того, что произойдет раньше.

Форматы файлов

При создании файла данных, подобного используемому в рассматриваемом примере, формат хранения данных полностью зависит от программиста. (Однако, если плани-

руется использование файла данных в другом приложении, возможно, придется учесть особенности интерпретации данных этого приложения.)

Давайте создадим строку, которая представляет одну запись в файле данных. Это можно сделать следующим образом:

```
$outputstring = $date."\t".$tireqty." tires \t".$oilqty." oil\t"
                      .$sparkqty." spark plugs\t\$".$total
                      ."\t". $address."\n";
```

В этом простом примере каждая запись заказа сохраняется в отдельной строке файла. Подобное решение обусловлено тем, что позволяет в качестве простого разделителя строк использовать символ новой строки. Поскольку символы новой строки невидимы, они представляются с помощью управляющей последовательности "\n".

Поля данных будут записываться в одном и том же порядке, а в качестве разделителя полей будет использоваться символ табуляции. Опять-таки, поскольку этот символ невидим, он представляется управляющей последовательностью "\t". В качестве разделителя можно использовать любой легко читаемый символ.

Разделителем, или ограничителем, должен быть любой символ, который наверняка не будет встречаться в исходных данных, иначе придется подвергнуть исходные данные дополнительной обработке с целью удаления или отмены всех вхождений ограничителя. Обработка ввода рассматривается в главе 4. Пока предположим, что никто не будет выводить символы табуляции в форму заказа. Помещение пользователем символов табуляции или новой строки в однострочное поле ввода HTML маловероятно, но не так уж невозможно.

Использование специального разделителя полей упрощает разделение данных на отдельные переменные во время считывания. Подробнее этот вопрос рассматривается в главах 3 и 4. Пока каждый заказ будет обрабатываться как отдельная строка.

После обработки нескольких заказов содержимое файла будет выглядеть подобно примеру, приведенному в листинге 2.1.

Листинг 2.1 orders.txt — Пример содержимого файла заказов

```
15:42, 20th April        4 tires          1 oil    6 spark plugs
   $434.00
    22 Short St, Smalltown
15:43, 20th April        1 tires          0 oil    0 spark plugs
   $100.00
    33 Main Rd, Newtown
15:43, 20th April        0 tires          1 oil    4 spark plugs
   $26.00
    127 Acacia St, Springfield
```

Закрытие файла

По завершении использования файла его следует закрыть при помощи функции **fclose(),** как показано ниже:

```
fclose($fp);
```

Эта функция возвращает значение **true** в случае успешного закрытия файла и **false,** если файл не был закрыт. Ошибка при этом значительно менее вероятна, чем при открытии файла, поэтому в данном случае проверка выполнения функции не выполняется.

Считывание из файла

Уже сейчас клиенты Боба могут отправлять свои заказы через Web, однако если сотрудники компании Боба захотят взглянуть на заказы, им придется открывать файлы самостоятельно.

Давайте создадим Web-интерфейс, который позволит служащим компании Боба легко читать файлы. Код этого интерфейса приведен в листинге 2.2.

Листинг **2.2** **vieworders.php** — интерфейс для просмотра файла заказов

```
<html>
<head>
  <title>Bob's Auto Parts - Customer Orders</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2>Customer Orders</h2>
<?

 @  $fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "r");

   if (!$fp)
   {
     echo "<p><strong>No orders pending."
          ."Please try again later.</strong></p></body></html>";
     exit;
   }

   while (!feof($fp))
   {
       $order= fgets($fp, 100);
       echo $order."<br>";
   }

   fclose($fp);
?>
</body>
</html>
```

В этом сценарии выполняется ранее описанная последовательность действий: открытие файла, считывание из файла, закрытие файла. Вывод, генерируемый этим сценарием при использовании файла данных из листинга 2.1, показан на рис. 2.4.

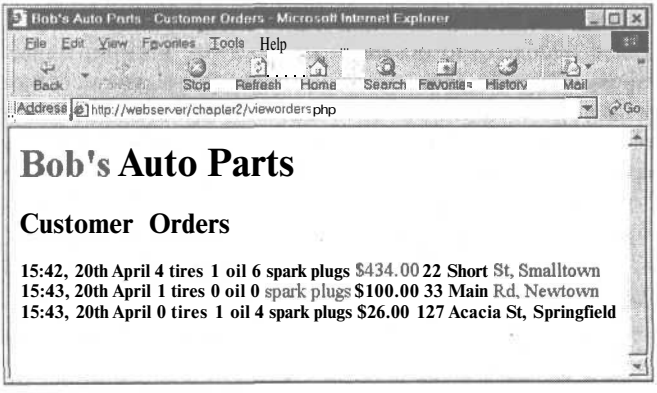


РИСУНОК 2.4
Сценарий vieworders.php отображает в окне браузера все заказы, которые в данный момент записаны в файле orders.txt.

Давайте подробнее рассмотрим функции, используемые в этом сценарии.

Открытие файла для чтения: fopen()

Как и ранее, мы открываем файл с помощью функции **fopen().** На этот раз файл открывается только для чтения, поэтому используется режим файла "r":

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "r");
```

Определение конца файла: feof()

В этом примере используется цикл **while** для считывания из файла до тех пор, пока не будет достигнут конец файла. Проверка на наличие конца файла осуществляется при помощи функции **feof():**

```
while (!feof($fp))
```

Функция **feof()** принимает в единственном параметре указатель файла. Она будет возвращать значение **true,** если указатель файла находится в конце файла. Имя функции легко запомнить, если знать, что **feof** означает File End Of File (Файл: конец файла).

В данном случае (и вообще при считывании) считывание из файла выполняется до тех пор, пока не будет достигнут **EOF.**

Построчное считывание: fgets(), fgetss() и fgetcsv()

В рассматриваемом примере для считывания из файла используется функция **fgets()**:

```
$order= fgets($fp, 100);
```

Эта функция используется для считывания из файла по одной строке за один раз. В данном случае считывание будет выполняться до тех пор, пока не встретится символ новой строки (**\n**), EOF или из файла не будут прочитаны 99 байт. Максимальная длина считываемой строки равна указанной длине минус один байт.

Существует много различных функций, которые используют для считывания из файлов. Функция **fgets()** полезна при работе с файлами, содержащими обычный текст, который требуется обрабатывать по кускам.

Интересным вариантом функции **fgets()** является функция **fgetss(),** имеющая следующий прототип:

```
string fgetss(int fp, int length, string [allowable_tags]);
```

Эта функция во многом подобна функции **fgets()** за исключением того, что она будет избавляться от любых дескрипторов PHP и HTML, найденных в строке. Если в файле необходимо оставить конкретные дескрипторы, они должны быть включены в строку **allowable_tags**. Функцию **fgetss()** следует использовать для обеспечения безопасности при считывании файла, записанного кем-либо другим или содержащего данные, введенные пользователем. Отсутствие ограничений на наличие в файле HTML-кода может привести к нарушению тщательно спланированного форматирования. Отсутствие ограничений на наличие в файле PHP-кода может предоставить злонамеренному пользователю почти полную свободу действий на сервере.

Функция **fgetcsv()** — еще одна вариация функции **fgets().** Она имеет следующий прототип:

```
array fgetcsv(int fp, int length, string [delimiter]);
```

Эта функция используется для разделения строк файлов при использовании в качестве разделительного символа табуляции, как предлагалось ранее, или запятой, которая обычно применяется в электронных таблицах и других приложениях. Если требуется восстановить переменные заказа отдельно одна от другой, а не в виде строки текста, следует прибегнуть к функции **fgetcsv().** Она вызывается подобно функции **fgets(),** но ей необходимо передать разделитель, используемый для разделения полей. Например,

```
$order = fgetcsv($fp, 100 "\t");
```

получает строку из файла и разбивает ее при каждом обнаружении символа табуляции (\t). Результирующие данные помещаются в массив (в этом примере — в **$order**). Подробнее массивы рассматриваются в главе 3.

Параметр *length* должен быть больше длины самой длинной строки считываемого файла, выраженной в символах.

Считывание всего файла: readfile(), fpassthru(), file()

Вместо считывания по одной строке из файла за один проход можно считывать весь файл. Существуют три различных способа.

Первый заключается в использовании функции **readfile()**. Весь ранее созданный сценарий можно заменить одной строкой:

```
readfile("$DOCUMENT_ROOT/../orders/orders.txt");
```

Функция **readfile()** открывает файл, повторяет его содержимое в стандартном выводе (окне браузера), а затем закрывает файл. Прототип этой функции имеет вид

```
int readfile(string имя_файла, int [use_include_path]);
```

Необязательный второй параметр указывает, должен ли PHP искать файл в пути **use_include_path**, и действует так же, как в функции **fopen().** Функция возвращает общее количество байтов, считанных из файла.

Во-вторых, можно использовать функцию **fpassthru()**. Вначале необходимо открыть файл с помощью функции **fopen().** Затем указатель файла можно передать в функцию **fpassthru(),** которая загрузит содержимое файла, начиная с позиции, заданной указателем, в стандартный вывод. По завершении этого процесса функция закрывает файл.

Ранее приведенный сценарий можно заменить функцией **fpassthru()** следующим образом:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "r");
fpassthru($fp);
```

Функция **fpassthru()** возвращает значение **true**, если считывание было выполнено успешно, и **false** — в противном случае.

Третья возможность считывания всего файла — использование функции **file()**. Эта функция идентична функции **readfile()** за исключением того, что вместо повторения файла в стандартном выводе она преобразует его в массив. Подробнее это исследуется при рассмотрении массивов в главе 3. А пока, просто для сведения, отметим, что эту функцию следовало бы вызывать так:

```
$filearray = file($fp);
```

Эта строка приведет к считыванию всего файла в массив, названный **$filearray**. Каждая строка файла сохраняется в отдельном элементе массива.

Считывание символа: fgetc()

Еще одна возможность обработки файлов — считывание из файла по одному символу. Это выполняется с помощью функции **fgetc().** В качестве своего единственного параметра она принимает указатель файла и возвращает следующий символ файла. Цикл **while** в нашем первоначальном сценарии можно заменить циклом, в котором используется функция **fgetc()**:

```
while (!feof($fp))
{
   $char = fgetc($fp);
   if (!feof($fp))
      echo ($char=="\n" ? "<br>": $char);
}
```

Используя функцию **fgetc(),** этот код считывает из файла по одному символу за раз и сохраняет его в переменной **$char**, пока не будет достигнут конец файла. Затем выполняется небольшая дополнительная обработка с целью замещения текстовых символов конца строки \n HTML-разделителями строк **
**. Это делается лишь для приведения в порядок форматирования. Поскольку без этого кода браузеры не распознают новые строки, весь файл был бы выведен в виде единой строки. (Попытайтесь сделать это и посмотрите, что получится.) Для выполнения этой задачи используется тернарная операция.

Побочный эффект использования функции **fgetc()** вместо функции fgets() заключается в том, что она будет возвращать символ EOF, в то время как fgets() не делает этого. После считывания символа приходится снова выполнять проверку с помощью функции **feof()**, поскольку символ **EOF** не должен отображаться в окне браузера.

В общем случае считывание файла символ за символом не находит особого применения, если только по какой-либо причине не требуется посимвольная обработка файла.

Считывание строк произвольной длины: fread()

Последний способ считывания из файла, который мы рассмотрим — использование функции **fread()** для считывания из файла произвольного количества байтов. Эта функция имеет следующий прототип:

```
string fread(int fp, int length);
```

Функция считывает *length* байтов или все байты до конца файла, в зависимости от того, что произойдет раньше.

Другие полезные файловые функции

Существует ряд других файловых функций, которые временами могут оказаться полезными.

Проверка существования файла: file_exists()

Если необходимо проверить файл на предмет существования без его открытия, можно воспользоваться функцией **file_exists(),** как показано в следующем примере:

```
if (file_exists("$DOCUMENT_ROOT/../orders/orders.txt"))
   echo "There are orders waiting to be processed.";
else
   echo "There are currently no orders.";
```

Выяснение размера файла: filesize()

Размер файла можно проверить с помощью функции **fflesize().** Она возвращает размер файла, выраженный в байтах:

```
echo filesize("$DOCUMENT_ROOT/../orders/orders.txt");
```

Эта функция может применяться в сочетании с функцией **fread()** для одновременного считывания всего файла (или определенной его части). Весь первоначальный сценарий можно заменить следующим кодом:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "r");
echo fread( $fp, filesize("$DOCUMENT_ROOT/../orders/orders.txt" ));
fclose( $fp );
```

Удаление файла unlink()

Если после обработки заказов файл заказов необходимо удалить, это выполняется с помощью функции **un!ink().** (Нет ни одной функции с именем **delete.)** Например:

```
unlink("$DOCUMENT_ROOT/../orders/orders.txt");
```

Эта функция возвращает значение **false,** если файл не может быть удален. Как правило, это будет происходить при недостаточном уровне прав доступа к файлу или если файл не существует.

Перемещение внутри файла: rewind(), fseek и ftell()

Выяснять позицию указателя файла внутри файла и изменять ее можно с помощью функций **rewind(), fseek() и ftell().**

Функция **rewind()** переустанавливает указатель файла на начало файла. Функция **ftellQ** сообщает в байтах позицию указателя относительно начала файла. Например, в нижнюю часть первоначального сценария (перед командой **fclose())** можно добавить следующие строки:

```
echo "Final position of the file pointer is ". (ftell ($fp));
echo "<br>";
rewind($fp);
echo "After rewind, the position is ". (ftell($fp));
echo "<br>";
```

Вывод в окне браузера будет выглядеть аналогично показанному на рис. 2.5.

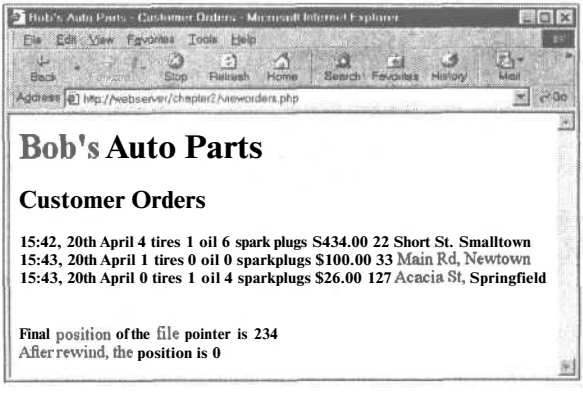


РИСУНОК 2.5
После считывания заказов указатель файла указывает на конец файла, имеющий смещение, равное 234 байтам. В результате вызова функции rewind указатель снова устанавливается на О позицию — начало файла.

Функция **fseek()** может использоваться для установки указателя файла в некоторую точку внутри файла. Ее прототип имеет вид

```
int fseek(int fp, int offset);
```

В результате вызова функции **fseek()** указатель файла *fp* устанавливается в точку файла, имеющую смещение *offset* байтов относительно начала файла. Вызов функции **rewind()** эквивалентен вызову функции **fseek()** со смещением, равным нолю. Например, функцию **fseek()** можно использовать для нахождения средней записи в файле или для выполнения бинарного поиска. Часто, когда подобные задачи требуется решать применительно к достаточно сложному файлу данных, имеет смысл использовать базу данных.

Блокирование файлов

Представьте себе ситуацию, когда два клиента одновременно пытаются заказать товар. (Эта ситуация возникает не столь уж редко, особенно когда Web-сайт начинает обрабатывать значительные информационные потоки.) Что произойдет, если один клиент вызовет функцию **fopen()** и начнет запись, а затем второй клиент также вызовет функцию **fopen()** и тоже попытается выполнить запись? Каким в результате будет содержимое файла? Будет ли вначале записан первый заказ, а затем второй, или наоборот? Будет ли записан первый заказ или второй? Либо же содержимое будет представлять собой нечто менее полезное, наподобие двух произвольно чередующихся заказов? Ответ на эти вопросы зависит от конкретной используемой операционной системы, но часто точно ответить на них невозможно.

Во избежание подобных проблем используется блокирование файлов. В PHP блокирование реализуется с помощью функции **flock().** Эта функция должна вызываться после открытия файла, но перед считыванием данных из файла или их записью в файл.

Прототип функции **flock()** выглядит так:

```
bool flock(int fp, int operation);
```

В функцию необходимо передать указатель на открытый файл и число, представляющее вид требуемой блокировки. Функция возвращает значение **true,** если блокировка была успешно выполнена, и **false** — в противном случае.

Возможные значения параметра *operation* перечислены в табл. 2.2.

Таблица 2.2 Значения параметра operation функции flock()

Значение параметра operation	*Описание*
\	**Блокировка чтения. Это означает, что файл может использоваться совместно с другими читающими приложениями.**
2	**Блокировка записи. Это монопольный режим. Файл не доступен для совместного использования.**
3	**Снятие существующей блокировки.**
+4	**Добавление 4 к текущему значению параметра operation предотвращает другие попытки блокирования во время выполнения текущего блокирования.**

Если решено использовать функцию **flock()**, ее следует включить во все сценарии, в которых используется данный файл; в противном случае ее применение лишено смысла.

Для использования блокирования в рассматриваемом примере программу **processorder.php** необходимо изменить следующим образом:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "a", 1);
flock($fp, 2); // блокирование файла для записи
fwrite($fp, $outputstring);
flock($fp, 3); // снятие блокировки записи
fclose($fp);
```

Следует также добавить блокировки в файл **vieworders.php**:

```
$fp = fopen("$DOCUMENT_ROOT/../orders/orders.txt", "r");
flock($fp, 1); // блокирование файла для чтения
// считывание из файла
flock($fp, 3); // снятие блокировки записи
fclose($fp);
```

Теперь код более надежен, но все еще не идеален. Что произойдет, если два сценария попытаются одновременно выполнить блокирование? Это привело бы к конфликту, когда процессы соперничают за установку блокировки, но не известно, какому из них это удастся, что, в свою очередь, могло бы породить новые проблемы. Задачу можно решить значительно успешнее, используя СУБД.

Более рациональный способ обработки: системы управления базами данных

До сих пор во всех рассмотренных примерах использовались двумерные файлы. В следующем разделе книги будет рассматриваться применение MySQL — системы управления реляционными базами данных. У читателей может возникнуть вопрос: "Для чего это нужно?"

Проблемы, связанные с использованием двумерных файлов

При работе с двумерными файлами возникает ряд проблем:

- Когда двумерные файлы становятся большими, работа с ними существенно замедляется.
- Поиск конкретной записи или группы записей в двумерном файле затруднен. Если записи упорядочены, для поиска в ключевом поле можно использовать какой-либо из видов бинарного поиска в сочетании с применением записей фиксированной длины. Если нужно найти информацию, соответствующую определенному шаблону (например, найти всех клиентов, проживающих в Киеве), придется прочесть и проверить каждую из записей в отдельности.
- Конкурирующий доступ может порождать проблемы. Уже было показано, как блокируются файлы, но это может привести к возникновению описанной ранее конфликтной ситуации. Кроме того, это может привести к образованию "узкого места" в сети. При достаточно интенсивном информационном потоке в сайте большой группе пользователей может потребоваться ожидать разблокирования файла, прежде чем они смогут разместить свои заказы. Если ожидание продлится слишком долго, люди обратятся за покупкой куда-либо в другое место.

- Вся до сих пор рассмотренная обработка файлов сводилась к последовательной обработке — т.е. считывание начиналось с начала файла и выполнялось до его конца. При необходимости вставить записи или удалить их из середины файла (т.е. при необходимости произвольного доступа), это может оказаться затруднительным — в конце концов, придется считать весь файл в память, выполнить изменения и снова записать весь файл. При работе с большими файлами данных этот процесс сопряжен со значительной перегрузкой системы.
- Кроме ограничений, налагаемых правами доступа к файлам, не существует никакого способа обеспечения различных уровней доступа к данным.

Как эти проблемы решаются с помощью СУРБД

Системы управления реляционными базами данных (СУРБД) решают все эти проблемы:

- СУРБД могут обеспечить более быстрый доступ к данным, чем двумерные файлы. А MySQL, система управления базами данных, используемая в этой книге, обладает одними из самых высоких показателей производительности среди всех СУРБД.
- В СУРБД можно легко отправлять запрос для извлечения наборов данных, соответствующих определенным критериям.
- СУРБД обладают встроенными механизмами обработки конкурирующих обращений, что позволяет программисту не беспокоиться об этом.
- СУРБД обеспечивают произвольный доступ к данным.
- СУРБД обладают встроенными системами определения прав доступа. MySQL обладает особенно большими возможностями в этой области.

Вероятно, главная побудительная причина использования СУРБД заключается в том, что все (или, по меньшей мере, большинство) функциональные возможности, требуемые от системы хранения данных, в ней уже реализованы. Конечно, можно было бы создать собственную библиотеку PHP-функций, но зачем же заново изобретать колесо?

В части II, "Использование MySQL", мы рассмотрим работу реляционных баз данных в целом и то, как, в частности, можно установить и использовать MySQL для создания Web-сайтов, основанных на использовании баз данных.

Дополнительная информация

Для получения более подробной информации по взаимодействию с файловой системой можно обратиться непосредственно к главе 16. В этой главе мы рассмотрим изменение прав доступа, прав владения и имен файлов; мы научимся работать с каталогами и узнаем, как взаимодействовать со средой файловой системы.

Кроме того, можно прочесть раздел интерактивного руководства по PHP, посвященный файловой системе, который находится по адресу **http://www.php.net.**

Что дальше

В следующей главе мы рассмотрим, что представляют собой массивы и как их можно задействовать для обработки данных в PHP-сценариях.

Глава 3

Использование массивов

Эта глава посвящена использованию важной программной конструкции — массивов. Переменные, рассмотренные в предшествующих главах, являются *скалярными,* в них хранится единственное значение. *Массив* — это переменная, в которой хранится набор, или последовательность, значений. Один массив может содержать много элементов. Каждый элемент может содержать единственное значение, такое как текст или число, или другой массив. Массив, который содержит другие массивы, называется многомерным массивом.

PHP поддерживает как численно индексированные, так и ассоциативные массивы. Читатели, которые уже работали с каким-либо языком программирования, возможно, знакомы с численно индексированными массивами, но если ранее не приходилось использовать PHP или Perl, то, скорее всего, с ассоциативными массивами сталкиваться не доводилось. Ассоциативные массивы позволяют использовать значения, которые находят более широкое применение, чем индексы. Вместо численных индексов каждый элемент такого массива может иметь слова или другую полезную информацию.

Мы продолжим разработку примера сайта компании автозапчастей Боба, используя массивы, что должно упростить работу с такой повторяющейся информацией, как заказы клиентов. Кроме того, использование массивов позволит создать более короткий и гармоничный код для выполнения некоторых действий с файлами, выполнявшихся в предыдущей главе.

В этой главе рассматриваются следующие основные вопросы:

- Что такое массив?
- Численно индексированные массивы
- Ассоциативные массивы

- Многомерные массивы
- Сортировка массивов
- Дополнительная информация

Что такое массив

В главе 1 рассматривались скалярные переменные. Скалярная переменная — это именованная ячейка памяти, в которой хранится значение; по аналогии, массив — это именованная ячейка памяти, в которой хранится *набор* значений, что позволяет группировать обычные скалярные значения.

Список товаров, поставляемых компанией Боба, будет представлять собой массив. На рис. 3.1 показан список трех из этих товаров, хранящихся в форме массива, и переменная **$products**, в которой хранятся три значения. (Вскоре мы рассмотрим создание таких переменных.)



РИСУНОК 3.1 *Товары, поставляемые компанией Боба, могут храниться в массиве.*

После того, как информация сохранена в виде массива, с ней можно выполнять ряд полезных действий. Используя конструкции циклов, описанные в главе 1, работу можно сохранять, выполняя одни и те же действия над каждым элементом массива. Весь объем информации можно перемещать как единый блок. Таким образом, все значения могут быть переданы в функцию с помощью одной строки кода. Например, может требоваться упорядочение товаров по алфавиту. Для этого весь массив можно было бы передать в PHP-функцию **sort()**.

Хранящиеся в массиве значения называются *элементами* массива. Каждый элемент массива имеет связанный с ним *индекс* (называемый также *ключом),* который используется для доступа к элементу.

В большинстве языков программирования массивы имеют численные индексы, которые, как правило, начинаются с нуля или единицы. PHP также поддерживает такой тип массивов.

Но кроме того, PHP поддерживает также *ассоциативные* массивы, которые должны быть знакомы программистам на Perl. В качестве индексов в ассоциативных массивах могут использоваться практически любые значения, но, как правило, таковыми являются строки.

Рассмотрение начнем с численно индексированных массивов.

Численно индексированные массивы

Инициализация численно индексированных массивов

Для создания массива, показанного на рис. 3.1, можно использовать следующую строку кода:

```
$products = array( "Tires", "Oil", "Spark Plugs" );
```

В результате создается массив **products,** содержащий три заданных значения: "Tires", **"Oil"** и **"Spark Plugs".** Обратите внимание, что подобно инструкции **echo,** **array()** в действительности является скорее языковой конструкцией, нежели функцией.

В зависимости от требуемого содержимого массива, возможно, его не придется инициализировать вручную, как показано в предыдущем примере.

При наличии данных, которые требуются в другом массиве, можно просто копировать один массив в другой с помощью операции =.

Если в массиве необходимо хранить возрастающую последовательность чисел, для автоматического его создания можно использовать функцию **range()**. Следующая строка кода создает массив **numbers,** содержащий элементы, которые являются числами от 1 до 10:

```
$numbers = range(1,10);
```

Если информация хранится в файле на диске, содержимое массива можно загрузить непосредственно из файла. Этот процесс будет рассматриваться в разделе "Загрузка массивов из файлов" далее в главе.

Если данные массива хранятся в базе данных, содержимое массива можно загрузить непосредственно из базы данных. Этот процесс исследуется в главе 10.

Можно также использовать различные функции для извлечения части массива или для изменения порядка следования элементов массива. Некоторые из этих функций будут рассматриваться в разделе "Другие манипуляции с массивами" этой главы.

Доступ к содержимому массива

Для доступа к содержимому переменной используется ее имя. Если переменная является массивом, доступ к ее содержимому осуществляется через имя переменной и ключ, или индекс. Ключ, или индекс, указывает, каким хранимым значениям реализуется доступ. Индекс указывается в квадратных скобках после имени.

Например, чтобы использовать содержимое массива **products,** необходимо ввести **Sproducts[0], $products[1] и $products[2].**

Нулевой элемент является первым элементом массива. Эта же схема нумерации используется в C, C++, Java и ряде других языков программирования, но чтобы привыкнуть к ней, потребуется некоторое время.

Как и в случае других переменных, содержимое элементов массива изменяется с помощью операции =. Так следующая строка заменит первый элемент массива **"Tires"** элементом **"Fuses".**

```
$products[0] = "Fuses";
```

Для отображения содержимого массива можно было бы ввести

```
echo "$products[0] $products[1] $products[2] $products[3]";
```

Подобно другим переменным PHP, массивы не нужно инициализировать или создавать заранее. Они автоматически создаются при первом использовании.

Следующий код создаст этот же массив **$products**:

```
$products[0] = "Tires";
$products[1] = "Oil";
$products[2] = "Spark Plugs";
```

Если массив **$products** еще не существует, первая строка создает новый массив с только одним элементом. Последующие строки добавляют значения в массив.

Использование циклов для доступа к массиву

Поскольку массив индексируется по последовательным номерам, для упрощения отображения его содержимого можно использовать цикл for:

```
for ( $i = 0; $i<3; $i++ )
   echo "$products[$i] ";
```

Этот цикл создаст такой же вывод, как и предыдущий код, но для вывода содержимого каждого из элементов большого массива потребуется ввод меньшего объема кода вручную. Возможность использования простого цикла для доступа к каждому элементу — замечательное свойство численно индексированных массивов. Выполнить циклический просмотр в ассоциативных массивах не столь легко, но зато они позволяют присваивать индексам осмысленные значения.

Ассоциативные массивы

При создании массива товаров мы предоставили PHP возможность присвоить каждому элементу индекс, определяемый по умолчанию. Это означает, что первый добавленный элемент стал 0 элементом, второй -- 1 и т.д. PHP поддерживает также ассоциативные массивы. В ассоциативным массиве с каждым значением можно связать любой ключ, или индекс.

Инициализация ассоциативного массива

Следующий код создает ассоциативный массив, в котором названия товаров используются в качестве ключей, а их цены — в качестве значений.

```
$prices = array( "Tires"=>100, "Oil"=>10, "Spark Plugs"=>4 );
```

Доступ к элементам массива

Как и ранее, доступ к содержимому осуществляется через имя переменной и ключ, поэтому к информации, сохраненной в массиве **prices** можно обратиться как к **$prices["Tires"], $prices["Oil"] и $pices["Spark Plugs"].**

Подобно численно индексированным массивам, ассоциативные массивы могут создаваться и инициализироваться по одному элементу.

Следующий код создает этот же массив **$prices**. Вместо создания массива с тремя элементами эта версия создает массив только *с* одним элементом, а затем добавляет в него еще два элемента.

```
$prices = array( "Tires"=>100 );
$prices["Oil"] = 10;
$prices["Spark Plugs"] = 4 ;
```

Ниже приведен еще один, несколько отличающийся от предыдущего, однако эквивалентный ему фрагмент кода. В этой версии массив вообще не создается явно. Он создается при добавлении в него первого элемента.

```
$prices["Tires"] = 100;
$prices["Oil"] = 10;
$prices["Spark Plugs"] = 4;
```

Организация циклов с использованием each() и list()

Поскольку в ассоциативном массиве индексы не являются числами, для работы с массивом нельзя использовать простой счетчик в цикле **for**. Следующий код выводит содержимое нашего массива **$prices**:

```
while( $element = each( $prices ) )
{
   echo $element[ "key" ];
   echo " - ";
   echo $element[ "value" ] ;
   echo "<br>";
}
```

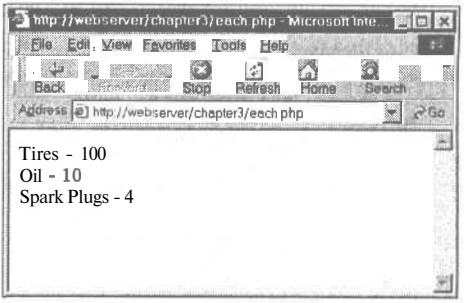


РИСУНОК 3.2 *Оператор each может использоваться для организации циклов в массивах.*

Вывод, создаваемый этим фрагментом сценария, показан на рис. 3.2.

В главе 1 были рассмотрены циклы **while** и оператор **echo.** В приведенном выше примере кода используется функция **each()**, которая ранее не встречалась. Эта функция возвращает текущий элемент массива и делает текущим следующий элемент. Поскольку функция **each()** вызывается внутри цикла **while,** она по очереди возвращает каждый из элементов массива и прекращает свое выполнение по достижении конца массива.

В этом примере кода переменная **$element** является массивом. При вызове функции **each()** она предоставляет массив с четырьмя значениями и четырьмя индексами ячеек массива. Ячейки key и 0 содержат ключ текущего элемента, а ячейки **value и 1** — значение текущего элемента. Хотя выбор ячеек не имеет значения, мы использовали именованные ячейки, а не нумерованные.

Это же можно сделать более изящным и привычным способом — использовать функцию **list()** для разделения массива на ряд значений. Два значения, передаваемые функцией **each(),** можно разделить следующим образом:

```
$list( $product, $price ) = each( $prices ) ;
```

Эта строка использует функцию **each()** для получения текущего элемента из массива **$prices**, возвращает его в виде массива и делает следующий элемент текущим. Кроме того, функция **list()** используется для преобразования элементов 0 и 1 массива, возвращаемого функцией **each(),** в две новых переменных: **$product и $price**.

Можно циклически просмотреть весь массив **$prises**, повторяя его содержимое, воспользовавшись следующим коротким сценарием.

```
while ( list( $product, $price ) = each( $prices ) )
   echo "$product - $price<br>";
```

Этот сценарий создает такой же вывод, как и предыдущий, но его легче читать, поскольку функция **list()** позволяет присваивать имена переменным.

При использовании функции **each()** следует помнить, что массив отслеживает текущий элемент. Если в одном и том же сценарии элемент необходимо использовать дважды, с помощью функции **reset()** потребуется снова установить текущий элемент на начало массива. Чтобы снова выполнить циклический просмотр массива **prices** следует ввести:

```
reset($prices);
while ( list( $product, $price ) = each( $prices ) )
   echo "$product - $price<br>";
```

В результате текущий элемент будет снова установлен на начало массива, что позволит вновь выполнить в нем просмотр.

Многомерные массивы

Массив не обязательно должен быть простым списком ключей и значений — каждая ячейка массива может содержать другой массив. Таким образом, можно создать двумерный массив. Двумерный массив можно представить себе в виде матрицы, или таблицы, ширина и высота которой выражается строками и столбцами.



РИСУНОК 3.3 *Более подробную информацию о товарах, поставляемых компанией Боба, можно хранить в двумерном массиве.*

Если бы для каждого товара, поставляемого компанией Боба, требовалось хранить более одного вида информации, для этого можно было бы воспользоваться двумерным массивом.

На рис. 3.3 товары, поставляемые компанией Боба, представлены в виде двумерного массива, каждая строка которого представляет отдельный вид товара, а каждый столбец — атрибут хранящегося товара.

Для определения данных в массиве, показанном на рис. 3.3, нужно было бы записать следующий PHP-код:

```
$products = array( array( "TIR", "Tires", 100 ),
                   array( "OIL", "Oil", 10 ),
                   array( "SPK", "Spark Plugs", 4 ) );
```

Из этого определения видно, что теперь массив товаров содержит три массива.

Вспомните, что для доступа к данным в одномерном массиве требуется имя массива и индекс элемента. Это же справедливо по отношению к двумерному массиву за исключением того, что каждый элемент имеет два индекса — строку и столбец. (Верхняя строка является строкой с номером 0, а крайний слева столбец — столбцом с номером 0.)

Для отображения содержимого этого массива можно было бы вручную обратиться к каждому из элементов в следующем порядке:

```
echo "|".$products[0][0]."|".$products[0][1]."|".$products[0][2]."|<BR>";
echo "|".$products[1][0]."|".$products[1][1]."|".$products[1][2]."|<BR>";
echo "|".$products[2][0]."|".$products[2][1]."|".$products[2][2]."|<BR>";
```

Или же для получения этого же результата можно было бы поместить цикл **for** внутрь другого цикла **for.**

```
for ( $row = 0; $row < 3; $row++ )
{
   for ( $column = 0; $column < 3; $column++ )
   {
      echo "|".$products[$row][$column];
   }
   echo "|<BR>";
}
```

Обе версии этого кода создают в окне браузера одинаковый вывод:

```
|TIR|Tires|100|
|OIL|Oil|10|
|SPK|Spark Plugs|4|
```

Единственное различие между двумя приведенными примерами состоит в том, что при использовании второй версии применительно к большому массиву код будет короче.

Возможно, вместо номеров столбцов вы предпочтете создать их имена, как показано на рис. 33. Для этого можно использовать ассоциативные массивы. Для сохранения этого же набора товаров при использовании имен столбцов, показанных на рис. 3.3, применяется следующий код:

```
$products = array ( array ( Code => "TIR",
                            Description => "Tires",
                            price => 100
                          ),
                    array ( Code => "OIL",
                            Description => "Oil",
                            price => 10
                          ),
                    array ( Code => "SPK",
                            Description => "Spark Plugs",
                            price =>4
                          )
                  );
```

С этим кодом проще работать, если нужно получить единственное значение. Проще запомнить, что описание хранится в столбце Description (Описание), чем что оно хранится в столбце 1. При использовании ассоциативных массивов, не приходится запоминать, что элемент хранится в ячейке [x][y]. Данные можно легко найти, обратившись к ячейке с осмысленными именами строки и столбца.

Однако при этом утрачивается возможность использования простого цикла **for** для поочередного просмотра всех столбцов. Ниже приведен один из вариантов кода для отображения этого массива:

```
for ( $row = 0; $row < 3; $row++ )
{
  echo "|".$products[$row]["Code"]."|".$products[$row]["Description"].
       "|".$products[$row]["Price"]."|<BR>";
}
```

С использованием цикла **for** можно просмотреть внешний численно индексированный массив **$products**. Каждая строка в массиве **$products** — ассоциативный массив. Используя функции **each() и list()** в цикле **while,** можно просмотреть ассоциативные массивы. Следовательно, внутри цикла **for** требуется цикл **while.**

```
for ( $row = 0; $row < 3; $row++ )
(
  while ( list( $key, $value ) = each( $products[ $row ] ) )
  {
    echo "|$value";
  }
  echo "|<BR>";
!
```

Не обязательно ограничиваться двумя измерениями — так же, как элементы массива могут содержать другие массивы, эти массивы, в свою очередь, могут содержать другие массивы.

Трехмерный массив имеет высоту, ширину и глубину. Если вам удобно представлять двумерный массив в виде таблицы, имеющей строки и столбцы, представьте себе стопку таких таблиц. Ссылка на каждый элемент такого массива будет осуществляться по его слою, строке и столбцу.

Если бы Боб разделил поставляемые им товары на категории, для их хранения можно было бы использовать трехмерный массив. Товары, поставляемые компанией Боба, хранящиеся в трехмерном массиве, показаны на рис. 3.4.

Из кода определения этого массива видно, что трехмерный массив — это массив, содержащий массив массивов.

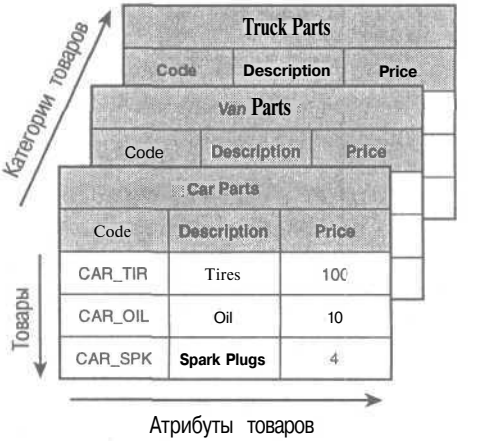


РИСУНОК 3.4 *Этот трехмерный массив позволяет разделить товары на категории.*

```
$categories = array( array (array( "TIR", "Tires", 100 ),
                            array( "OIL", "Oil", 10 ),
                            array( "SPK", "Spark Plugs", 4 )
                           ),
                     array (array( "TIR", "Tires", 100 ),
                            array( "OIL", "Oil", 10 ),
                            array( "SPK", "Spark Plugs", 4 )
                           ),
                     array (array( "TIR", "Tires", 100 ),
                            array( "OIL", "Oil", 10 ),
                            array( "SPK", "Spark Plugs", 4 )
                           )
                   );
```

Поскольку этот массив имеет только численные индексы, для отображения его содержимого можно использовать вложенные циклы **for**.

```
for ( $layer = 0; $layer < 3; $layer++ )
{
   echo "Layer $layer<BR>";
   for ( $row = 0; $row < 3; $row++ )
   {
      for ( $column = 0; $column < 3; $column++ )
      {
         echo "|".$categories[$layer][$row][$column];
      }
      echo "|<BR>";
   }
}
```

Способ создания многомерных массивов позволяет создавать четырех-, пяти-, или шестимерные массивы. Правила синтаксиса языка не налагают никаких ограничений на количество измерений, но человеку трудно себе представить конструкции, содержащие более трех измерений. Большинство реальных задач логически соответствует конструкциям с тремя или менее измерениями.

Сортировка массивов

Часто полезно сортировать связанные данные, хранящиеся в массиве. Сортировка одномерного массива достаточно проста.

Использование функции sort()

Следующий код приводит к упорядочению массива в алфавитном порядке:

```
$products = array( "Tires", "Oil", "Spark Plugs" );
sort($products);
```

Теперь элементы массива будут расположены в следующем порядке: **Oil, Spark Plugs, Tires.**

Значения можно упорядочивать также в цифровом порядке. При наличии массива, содержащего цены на товары, поставляемые компанией Боба, в нем можно выполнить сортировку в порядке возрастания численных значений:

```
$prices = array(100, 10, 4);
sort($prices);
```

Теперь цены будут приведены в порядке 4, 10, 100.

Следует отметить, что функция **sort** зависит от регистра, т.е. прописные буквы предшествуют строчным буквам. Так, 'A' меньше 'Z', но 'Z' меньше 'a'.

Использование функций asort() и ksort() для сортировки ассоциативных массивов

Если для хранения информации о товарах и их ценах используется ассоциативный массив, нужно использовать другие функции сортировки, обеспечивающее совместное сохранение ключей и значений при сортировке.

Следующий код создает ассоциативный массив, содержащий три товара и связанные с ними цены, а затем сортирует массив в порядке увеличения цен.

```
$prices = array( "Tires"=>100, "Oil"=>10, "Spark Plugs"=>4 );
asort($prices);
```

Функция **asort()** упорядочивает массив в соответствии со значениями элементов. В данном массиве значения — это цены, а ключи — текстовые описания. Если сортировку нужно выполнить не по ценам, а по описаниям, следует использовать функцию **ksort()**, которая выполняет сортировку не по значениям, а по ключам. Следующий код приведет к упорядочению ключей массива в алфавитном порядке — **Oil, Spark Plugs, Tires.**

```
$prices = array( "Tires"=>100, "Oil"=>10, "Spark Plugs"=>4 );
ksort($prices);
```

Сортировка в обратном порядке

Мы рассмотрели функции **sort()**, **asort()** и **ksort()**. Все эти функции выполняют сортировку массива в возрастающем порядке. Каждая из них имеет соответствующую ей функцию, выполняющую сортировку массива в порядке убывания. Обратные функции — **rsort()**, **arsort()** и **krsort()**.

Функции сортировки в обратном порядке используются так же, как функции сортировки. Функция **rsort()** выполняет сортировку одномерного численно индексирован-

ного массива в порядке убывания. Функция **arsort()** выполняет сортировку одномерного ассоциативного массива в порядке убывания значений элементов. Функция **krsort()** выполняет сортировку одномерного ассоциативного массива в порядке убывания значений ключей элементов.

Сортировка многомерных массивов

Сортировка массивов, имеющих более одного измерения, или в порядке, отличающемся от алфавитного или цифрового, более сложна. В PHP имеется возможность сравнения двух чисел или двух текстовых строк, но в многомерном массиве каждый элемент является массивом. В PHP отсутствует возможность сравнения двух массивов, поэтому для их сравнения необходимо создать метод. В большинстве случаев порядок слов или номеров очевиден — но в случае сложных объектов выполнение задачи становится более проблематичным.

Определяемые пользователем функции сортировки

Ниже приведено определение ранее использованного двумерного массива. В этом массиве хранятся названия товаров, поставляемых компанией Боба, их коды и цены.

```
$products = array( array( "TIR", "Tires", 100 ),
                   array( "OIL", "Oil", 10 ),
                   array( "SPK", "Spark Plugs", 4 ) );
```

Каков будет порядок значений, если выполнить сортировку в этом массиве? Поскольку известно, что именно представляет содержимое массива, существует, по меньшей мере, два полезных порядка сортировки. Товары можно упорядочить в алфавитном порядке по описаниям или в цифровом порядке по ценам. И то, и другое одинаково возможно, но нужно использовать функцию usort() и указать PHP, как следует сравнивать элементы. Для этого потребуется создать собственную функцию сравнения.

Следующий код выполняет сортировку этого массива в алфавитном порядке по значению второго столбца — описания.

```
function compare($x, $y)
{
   if ( $x[1] == $y[1] )
      return 0;
   else if ( $x[1] < $y[1] )
      return -1;
   else
      return 1;
}

usort($products, compare);
```

До сих пор мы использовали встроенные PHP-функции. Для сортировки этого массива мы определили свою собственную функцию. Создание функций будет подробно рассматриваться в главе 5, а пока приведем только краткое описание этого процесса.

Функция определяется с помощью ключевого слова function. Функции необходимо присвоить имя. Имена должны быть имеющими смысл, поэтому назовем нашу функцию compare(). Многие функции принимают параметры, или аргументы. Наша функция compare() принимает два аргумента: x и y. Ее назначение — принять два значения и определить их порядок.

Применительно к рассматриваемому примеру параметрами x и y будут два массива внутри основного массива, каждый из которых представляет один из товаров. Чтобы обратиться к элементу **Description** массива x, необходимо ввести **$x[1],** поскольку **Description** — второй элемент в этом массиве, а нумерация начинается с нуля. Для сравнения элементов **Description** из массивов, переданных в функцию, используются переменные **$x[1] и $y[1].**

Когда функция завершает свою работу, она может сообщить ответ вызвавшему ее коду. Это значение называется *возвращаемым.* Для возврата значения в функции используется ключевое слово **return.** Например, строка **return 1;** возвращает значение 1 коду, вызвавшему функцию.

Чтобы она могла использоваться функцией **usort(),** функция **compare()** должна сравнивать x и y. Она должна возвращать 0, если значение x равно значению y, отрицательное число, если оно меньше, и положительное — если оно больше. В зависимости от значений x и y, наша функция будет возвращать значения **0,** 1 или -1.

Заключительная строка кода вызывает встроенную функцию **usort()** с массивом, в котором нужно выполнить сортировку **($products)** и именем нашей функции сравнения **(compare()).**

Если требуется, чтобы массив был отсортирован в другом порядке, можно просто создать другую функцию сравнения. Для выполнения сортировки по ценам необходимо просмотреть третий столбец массива и создать следующую функцию сравнения:

```
function compare($x, $y)
{
  if ( $x[2] == $y[2] )
    return 0;
  else if ( $x[2] < $y[2] )
    return -1;
  else
    return 1;
}
```

При вызове функции **usort($products, compare)** массив будет упорядочен в порядке возрастания цен.

Символ "и" в имени **usort()** означает "user" ("пользовательская"), поскольку этой функции требуется определяемая пользователем функция сравнения. Версии **uasort()** и **uksort()** функций **asort()** и **ksort()** также требуют использования определяемых пользователем функций сравнения.

Аналогично функции **asort(),** функция **uasort()** должна использоваться при сортировке ассоциативного массива по значениям. Функцию **asort** следует использовать, если значения являются простыми числами или текстом. Если же значения являются более сложными объектами, такими как массивы, следует определить функцию сравнения и использовать функцию **uasortQ.**

Подобно функции **ksort(),** функция **uksort()** должна использоваться при сортировке ассоциативного массива по значениям. Функцию **ksort** следует использовать, если значения являются простыми числами или текстом. Если же значения являются более сложными объектами, такими как массивы, следует определить функцию сравнения и воспользоваться функцией **uksort().**

Определяемые пользователем функции сортировки в обратном порядке

Функции **sort(), asort() и ksort()** имеют соответствующие функции сортировки в обратном порядке, имена которых содержат символ "r". Определяемые пользователем

функции сортировки не имеют обратных версий, но сортировку многомерных массивов можно выполнять и в обратном порядке. Поскольку функция сравнения создается программистом, достаточно создать функцию сравнения, которая возвращает противоположные значения. Чтобы можно было выполнить сортировку в обратном порядке, функция должна будет возвращать значение 1, если x меньше y, и -1, если x больше y. Например,

```
function reverseCompare($x, $y)
{
  if ( $x[2] == $y[2] )
    return 0;
  else if ( $x[2] < $y[2] )
    return 1;
  else
    return -1;
)
```

Теперь вызов функции **usort($products, reverseCompare)** привел бы к упорядочению массива в порядке убывания цен.

Изменение порядка следования элементов в массивах

В некоторых приложениях может потребоваться манипулирование порядком следования элементов массива другими способами. Функция **shuffle()** располагает элементы массива в случайном порядке. Функция **array_reverse()** возвращает копию массива, в которой все элементы располагаются в обратном порядке.

Использование функции shuffle()

Бобу нужно, чтобы несколько из поставляемых его компанией товаров были представлены на титульной странице сайта. Его компания поставляет большую номенклатуру товаров, но он хотел бы, чтобы три произвольно выбранных товара отображались на титульной странице. Чтобы постоянные посетители не скучали, желательно, чтобы при каждом посещении сайта выбирались другие три вида товаров. Этой цели легко достичь, если информация о всех товарах хранится в массиве. Программа, представленная в листинге 3.1, отображает на экране три произвольно выбранных рисунка, сортируя элементы массива в произвольном порядке, а затем отображая первые три из них.

Листинг 3.1 bobs_front_page.php — Использование PHP для создания динамической титульной страницы компании Bob's Auto Parts

```
<?
  $pictures = array("tire.jpg", "oil.jpg", "spark_plug.jpg",
                    "door.jpg",   "steering_wheel.jpg",
                    "thermostat.jpg",   "wiper_blade.jpg",
                    "gasket.jpg",   "brake_pad.jpg");
   shuffle($pictures);
?>
<html>
<head>
  <title>Bob's Auto Parts</title>
</head>
<body>
  <center>
    <h1>Bob's Auto Parts</Hl>
    <table width = 100%>
      <tr>
```

```
<?
   for ( $i = 0; $i < 3; $i++ )
   {
      echo "<td align = center><img src=\"";
      echo $pictures[$i];
      echo "\" width = 100 height = 100></td>";
   }
?>
         </tr>
      </table>
   </center>
</body>
</html>
```

Поскольку программа выбирает произвольные рисунки, практически при каждой ее загрузке она создает другую страницу, как показано на рис. 3.5.

Использование функции array_reverse()

Функция **array_reverse()** принимает массив и создает новый массив, элементы которого располагаются в обратном порядке. Например, существует ряд способов создания массива, содержащего элементы, пронумерованные в обратном порядке от 10 до 1.

Поскольку сама по себе функция **range()** создает возрастающую последовательность, для сортировки чисел в порядке убывания нужно использовать функцию **rsort()**. Или же можно было бы создавать массив по одному элементу, используя цикл **for:**

```
$numbers = array();
for($i=10; $i>0; $i--)
   array_push( $numbers,$i );
```

Цикл **for()** может выполняться в порядке убывания управляющей переменной, как показано в этом примере. Начальное значение устанавливается большим, а затем в конце каждого цикла операция -- уменьшает значение счетчика на единицу.

В этом примере был создан пустой массив, а затем к каждому его элементу применена функция **array_push()** для добавления нового элемента в конец массива. Попутно отметим, что обратной функцией для **array_push()** является функция **array_pop()**. Эта функция удаляет и возвращает один элемент из конца массива.

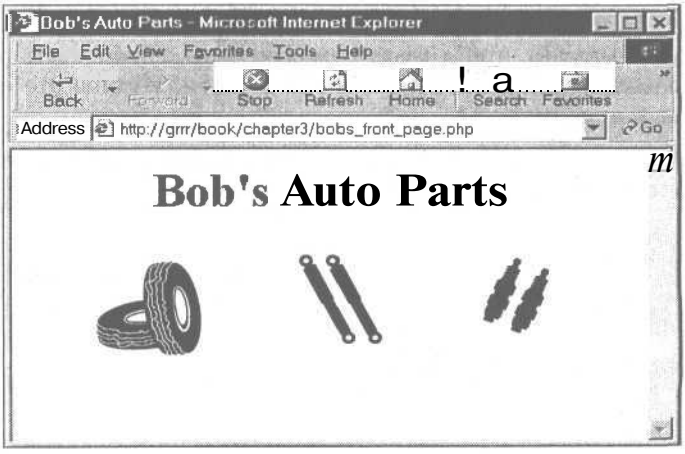


РИСУНОК 3.5
Функция shuffle() позволяет отображать три произвольно выбранных вида товаров.

Можно также воспользоваться функцией **array_reverse()** для изменения порядка следования элементов массива, созданного функцией **range()**.

```
$numbers = range(1,10);
$numbers = array_reverse($numbers);
```

Обратите внимание, что функция **array_reverse()** возвращает модифицированную копию массива. Поскольку исходный массив не нужен, новая копия сохраняется просто поверх исходной.

Загрузка массивов из файлов

В главе 2 заказы клиентов сохранялись в файле. Каждая строка файла выглядит приблизительно так:

```
15:42, 20th April 4 tires 1 oil 6 spark plugs $434.00 22 Short St, Smalltown
```

Для обработки или выполнения этого заказа его можно было бы загрузить обратно в массив. Сценарий, приведенный в листинге 3.2, отображает файл текущего заказа.

Листинг 3.2 vieworders.php — Использование PHP для отображения заказов

```
$orders= file("../../orders/orders.txt");
$number_of_orders = count($orders);
if ($number_of_orders == 0)
{
   echo "<p><strong>No orders pending.
         Please try again later.</strong></p>";
}
for ($i=0; $i<$number_of_orders; $i++)
{
   echo $orders[$i]."<br>";
}
```

Этот сценарий создает почти такой же вывод, как и показанный на рис. 2.4, создаваемый сценарием листинга 2.2 из предыдущей главы. Однако на сей раз используется функция **file()**, которая загружает весь файл в массив. Каждая строка файла становится отдельным элементом массива.

В этом сценарии также используется функция **count()** для выяснения количества элементов в массиве.

Более того, каждый раздел строк заказа можно было бы загрузить в отдельный элемент массива, чтобы разделы можно было обрабатывать по отдельности или форматировать их более привлекательным образом. Именно это и делает сценарий, приведенный в листинге 3.3.

Листинг 3.3 vieworders2.php — Использование PHP для разделения, форматирования и отображения заказов компании Боба

```
<html>
<head>
   <title>Bob's Auto Parts - Customer Orders</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2>Customer Orders</h2>
<?
```

```
    //Считывание всего файла
    //Каждый заказ становится элементом массива
    $orders= file("../../orders/orders.txt");
    // подсчет количества заказов в массиве
    $number_of_orders = count($orders);
    if ($number_of_orders == 0)
    {
      echo "<p><strong>No orders pending.
           Please try again later.</strong></p>";
    }
    echo "<table border=1>\n";
    echo "<tr><th bgcolor = V"#CCCCFF\">Order Date</td>
                <th bgcolor = \"#CCCCFF\">Tires</td>
                <th bgcolor = \"#CCCCFF\">Oil</td>
                <th bgcolor = \"#CCCCFF\">Spark Plugs</td>
                <th bgcolor = \"#CCCCFF\">Total</td>
                <th bgcolor = \"#CCCCFF\">Address</td>
          <tr>";
    for ($i=0; $i<$number_of_orders; $i++)
    {
      //разбиение каждой строки
      $line = explode( "\t", $orders[$i] );
      //сохранение только количества заказанных товаров
      $line[1] = intval( $line[1] );
      $line[2] = intval( $line[2] );
      $line[3] = intval( $line[3] );
      //вывод каждого заказа
      echo "<tr><td>$line[0]</td>
                <td align = right>$line[1]</td>
                <td align = right>$line[2]</td>
                <td align = right>$line[3]</td>
                <td align = right>$line[4]</td>
                <td>$line[5]</td>
            </tr>";
    }
    echo "</table>";
?>
</body>
</html>
```

Сценарий, приведенный в листинге 3.3, загружает весь файл в массив, но в отличие от примера, приведенного в листинге 3.2, в нем используется функция **explode()** для разбиения каждой строки, чтобы перед выводом ее можно было подвернуть определенной обработке и форматированию.

Создаваемый этим сценарием вывод показан на рис. 3.6.

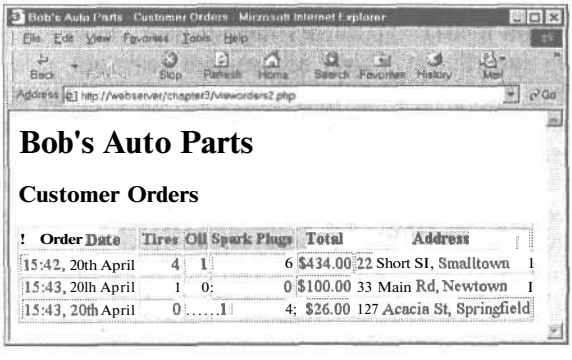


РИСУНОК 3.6
После разбиения записей заказа с помощью функции explode для большей наглядности каждую часть заказа можно поместить в отдельную ячейку таблицы.

Функция **explode** имеет следующий прототип:

array **explode(string *разделитель*,** string строка)

В предыдущей главе при сохранении этих данных в качестве разделителя мы использовали символ табуляции, поэтому здесь мы вызываем

```
explode( "\t", $orders[$i] )
```

В результате переданная строка разбивается на части. Каждый символ табуляции становится разделителем между двумя элементами. Например, строка

```
"15:42, 20th April\t4 tires\tl oil\t6 spark plugs\t$434.00\t22 Short St,
  ➥Small town"
```

разбивается на части **"15:42, 20th April", "4 tires", "1 oil", "6 spark plugs", "$434.00" и "22 Short St, Smalltown".**

В данном случае объем обработки не очень велик. Вместо того чтобы в каждой строке выводить название товара (покрышки, масло и свечи), в ней отображается только заказанное количество каждого из них, а таблица снабжается строкой заголовка, показывающей, что представляет каждое из значений.

Существует ряд способов возможного извлечения чисел из этих строк. В данном случае мы использовали функцию **intval()**. Как было отмечено в главе 1, эта функция преобразует тип **string в тип integer.** Преобразование выполняется достаточно разумно и будет игнорировать фрагменты строки, такие как метка в этом примере, которые не могут быть преобразованы в **integer.** Различные способы обработки строк рассматриваются в следующей главе.

Другие манипуляции с массивами

До сих пор мы осветили только около половины функций обработки массивов. Время от времени полезными могут оказаться многие другие функции.

Перемещение внутри массива: функции each, current(), reset(), end(), next(), pos() и prev()

Ранее уже упоминалось, что каждый массив имеет внутренний указатель, который указывает на текущий элемент массива. Ранее мы косвенно задействовали этот указатель при использовании функции **each(),** но его можно использовать и манипулировать им непосредственно.

При создании нового массива текущий указатель инициализируется так, чтобы указывать на первый элемент массива. Вызов функции **current($array_name)** возвращает первый элемент.

Вызов функции **next() или each()** перемещает указатель вперед на один элемент. Вызов функции **each($array_name)** возвращает текущий элемент, прежде чем переместить указатель. Функция **next()** ведет себя несколько иначе — вызов функции **next($array_name)** приводит к перемещению указателя, а затем к возврату нового текущего элемента.

Мы уже видели, что функция **reset()** возвращает указатель на первый элемент массива. Аналогично, вызов функции **end($array_name)** перемещает указатель в конец массива. Функции **reset() и end()** возвращают, соответственно, первый и последний элементы массива.

Для перемещения в массиве в обратном направлении можно воспользоваться функциями **end() и prev().** Функция **prev()** является обратной по отношению к функции **next().** Она перемещает текущий указатель на один элемент назад, а затем возвращает новый текущий элемент.

Например, следующий фрагмент кода отображает элементы массива в обратном порядке:

```
$value = end ($array);
while ($value)
{
   echo "$value<br>";
   $value = prev($array);
}
```

Если бы массив **$array** был объявлен следующим образом:

```
$array = array(1, 2, 3);
```

вывод в окне браузера выглядел бы как

```
3
2
1
```

С использованием функций **each(), current(), reset(), end(), next(), pos() и prev()** можно создавать код для перемещения в массиве в любом порядке.

Применение любой функции к каждому элементу массива: array_walk()

Иногда требуется выполнять одинаковые действия по отношению ко всем элементам массива или изменять их одинаковым образом. Делать это позволяет функция **array_walk.**

Функция **array_walk()** имеет следующий прототип:

```
int array_walk(array arr, string func, [mixed userdata])
```

Аналогично функции **usort()**, которая применялась ранее, функция **array_walk()** предполагает использование функции, объявленной пользователем.

Как видите, функция **array_walk()** принимает три параметра. Первый, *arr* — массив, который должен быть обработан. Второй, ***func*** — имя определяемой пользователем функции, которая будет применяться к каждому элементу массива. Третий параметр, ***userdata,*** является необязательным. В случае его использования он будет передан определяемой пользователем функции в качестве параметра. Вскоре мы покажем, как он работает.

Удобной определяемой пользователем функцией может оказаться функция, которая отображает каждый элемент в определенном формате.

Следующий фрагмент кода отображает каждый элемент в новой строке, вызывая определяемую пользователем функцию **myPrint()** с каждым элементом массива **$array**:

```
function myPrint($value)
{
   echo "$value<BR>";
}
array_walk($array, myPrint);
```

Создаваемая функция должна иметь конкретную сигнатуру. Для каждого элемента массива функция **array_walk** принимает ключ и значение, хранящиеся в массиве, и

любые данные, переданные в параметре *userdata*, после чего вызывает функцию пользователя, подобную следующей:

```
Yourfunction(значение, ключ, userdata)
```

В большинстве случаев определяемая пользователем функция будет использовать только хранящиеся в массиве значения. В некоторых случаях функции нужно будет передать параметр через ***userdata.***

Иногда наряду со значением интерес может представлять и ключ каждого элемента. Но в общем случае функция может игнорировать ключ и параметр *userdata*, как это имеет место в функции **MyPrint()**.

В качестве несколько более сложного примера создадим функцию, которая изменяет значения в массиве и требует передачи ей параметра. Обратите внимание, что хотя сам по себе ключ не представляет интереса, функция должна принять его, чтобы принять третий параметр.

```
function myMultiply(&$value, $key, $factor)
{
  $value *= $factor;
}
array_walk(&$array, "myMultiply", 3);
```

В этом примере мы определяем функцию **myMultiply()**, которая будет умножать каждый элемент массива на указанный коэффициент. В данном случае необходимо использовать необязательный третий параметр функции **array_walk()**, чтобы она передала его в функцию **rayMultipIyO** в качестве коэффициента умножения. В связи с этим функцию **myMultiply()** необходимо определить так, чтобы она принимала три параметра – значение элемента массива (**$value**), ключ элемента массива (**$key**) и параметр коэффициента умножения (**$factor**). Ключ функция игнорирует.

Следует отметить способ передачи параметра **$value**. Символ амперсанда (&) перед именем переменной в определении функции **myMultiplyO** означает, что $value будет *передаваться по ссылке.* Передача по ссылке позволяет функции изменять содержимое массива.

Подробнее передача по ссылке описывается в главе 5. Пока достаточно отметить, что для передачи по ссылке перед именем переменной помещается символ амперсанда.

Подсчет элементов в массиве: функции count(), sizeof() и array_count_values()

В приведенном ранее примере для подсчета количества элементов в массиве заказов использовалась функция **count().** Функция **sizeof()** имеет совершенно такое же назначение. Обе эти функции возвращают количество элементов в переданном им массиве. Так значение счетчика количества элементов будет равно 1 для обычной скалярной переменной и 0 при передаче либо пустого массива, либо переменной, которая не установлена.

Функция **array_count_values()** сложнее. Если вызвать **array_count_values($array)**, эта функция подсчитывает, сколько раз каждое *уникальное* значение встречается в массиве **$array**. (То есть, эта функция определяет мощность массива.) Она возвращает ассоциативный массив, содержащий таблицу частоты использования элементов. Этот массив содержит в качестве ключей все уникальные значения массива **$array**. Каждый ключ имеет численное значение, указывающее, сколько раз соответствующий ключ встречается в массиве **$array**.

Например, следующий код

```
$array = array(4, 5, 1, 2, 3, 1, 2, 1);
$ac = array_count_values($array);
```

создает массив **$ar,** который содержит

Ключ	*Значение*
4	1
5	1
1	3
2	*2*
3	1

Эта таблица показывает, что ключи 4, 5 и 3 встречаются в массиве **$array** по одному разу, 1 — три раза, а 2 — дважды.

Преобразование массивов в скалярные переменные: функция extract()

При наличии ассоциативного массива, содержащего ряд пар ключ-значение, с помощью функции **extract()** его можно преобразовать в набор скалярных переменных. Эта функция имеет следующий прототип:

```
extract (array var_array [, int extract_type] [, string prefix] );
```

Функция **extract()** предназначена для создания скалярных переменных, имеющих те же имена, что и у ключей массива. Переменным присваиваются значения, равные значениям, которые хранятся в массиве.

Ниже приведен простой пример.

```
$array = array( "key1" => "value1", "key2" => "value2",
   "key3" => "value3");
extract($array);
echo "$key1 $key2 $key3";
```

Этот фрагмент кода создает следующий вывод:

```
value1 value2 value3
```

Массив содержал три элемента с ключами **key1, key2 и key3.** За счет использования функции **extract()** были созданы три скалярные переменные **$keyl,** $key2 и **$key3.** Как видно из вывода, значениями переменных **$key1, $key2 и $key3** являются, соответственно, **"valuel",** "value2" "value3".

Функция **extract()** имеет два необязательных параметра: ***extract_type*** **и** ***prefix.*** Переменная **extract_type** задает функции **extract()** способ обработки конфликтов в случаях существования переменной с таким же именем, как у ключа. По умолчанию значение существующей переменной заменяется новым. Четыре допустимых значения параметра **extract_type** приведены в табл. 3.1.

Таблица 3.1 Допустимые значения параметра extract_type функции extract()

Значение	Описание
EXTR_OVERWRITE	Перезаписывает существующую переменную в случае конфликта.
EXTR_SKIP	Пропускает элемент в случае конфликта.
EXTR_PREFIX_SAME	В случае конфликта создает переменную **$prefix_key.** В этом случае необходимо передать в функцию и параметр **prefix.**
EXTR_PREFIX_ALL	Предваряет имена всех переменных префиксом **prefix.** В этом случае необходимо передать в функцию и этот параметр.

Два наиболее полезных значения определяемое по умолчанию (**EXTR_OVERWRITE**) **и EXTR_PREFIX_ALL**. Остальные два значения могут использоваться в тех случаях, когда заранее известно, что конкретные конфликты будут иметь место и требуется, чтобы ключ был пропущен или имел префикс. Ниже приведен простой пример использования **EXTR_PREFIX_ALL** в качестве параметра. Как видите, созданные переменные получают имена *префикс_имя-ключа*.

```
$array = array( "key1" => "value1", "key2" => "value2",
   "key3" => "value3");
extract($array, EXTR_PREFIX_ALL, "myPrefix");
echo "$myPrefix_key1 $myPrefix_key2 $myPrefix_key3";
```

Этот код также создаст вывод **valuel value2 value3**.

Обратите внимание, что для того, чтобы функция **extract()** извлекла элемент, ключ элемента должен быть допустимым именем переменной, т.е. ключи, которые начинаются с цифр или содержат пробелы, будут пропускаться.

Дополнительная информация

В этой главе освещены наиболее полезные по нашему мнению функции работы с PHP-массивами. Мы решили не освещать все существующие функции этого типа, поскольку краткое описание каждой из них присутствует в интерактивном руководстве по PHP по адресу **http://www.php.net.**

Что дальше

В следующей главе мы рассмотрим функции обработки строк. В ней будут освещены функции, которые выполняют поиск, разделение и объединение строк, а также мощные функции работы с регулярными выражениями, которые позволяют выполнять практически любые действия со строками.

Глава 4

Манипулирование строками и регулярные выражения

В этой главе мы рассмотрим использование строковых функций PHP для форматирования и манипулирования текстом. Мы рассмотрим также использование строковых функций или функций регулярных выражений для поиска (и замены) слов, выражений или иных последовательностей внутри строки.

Эти функции полезны во многих контекстах. Часто требуется очистить или переформатировать вводимые пользователем данные, которые должны быть сохранены в базе данных. В частности, функции поиска просто незаменимы при создании приложений инструментальных средств поиска.

В этой главе освещаются следующие темы:

- Форматирование строк
- Объединение и разделение строк
- Сравнение строк
- Сопоставление и замена подстрок с помощью строковых функций
- Использование регулярных выражений

Пример приложения: Smart Form Mail

В этой главе строковые функции и функции регулярных выражений рассматриваются в контексте приложения Smart Form Mail. Созданные здесь сценарии будут добавлены к сайту Bob's Auto Parts, который исследовался в нескольких предшествующих главах.

На этот раз мы построим простую и часто используемую форму отзывов клиентов, позволяющую клиентам

Боба вводить свои замечания и пожелания, как показано на рис. 4.1. Однако наше приложение будет обладать одним преимуществом по сравнению со многими другими формами, которые можно встретить в Web. Вместо того чтобы отправлять форму по обобщенному адресу электронной почты типа feedback@bobsdomain.com, мы постараемся придать процессу некоторую интеллектуальность за счет поиска в вводимых данных ключевых слов и выражений и последующей отправки сообщения электронной почты соответствующему сотруднику компании Боба. Например, если сообщение электронной почты содержит слово "advertising" ("реклама"), отзыв может быть отправлен в отдел маркетинга. Если сообщение электронной почты поступает от крупнейшего клиента Боба, оно может быть направлено непосредственно Бобу.

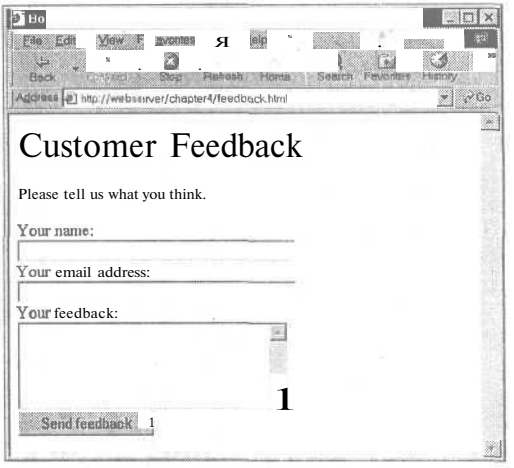


РИСУНОК 4.1 *Форма отзывов запрашивает имя, адрес электронной почты и комментарии клиентов Боба.*

Мы начнем рассмотрение с простого сценария, приведенного в листинге 4.1, дополняя его по мере изучения материала.

Листинг 4.1 processfeedback.php — Основной сценарий для создания содержимого формы сообщения электронной почты

```
<?
    $toaddress = "feedback@bobsdomain.com";
    $subject = "Feedback from web site";
    $mailcontent = "Customer name: ".$name."\n"
                   ."Customer email: ".$email."\n"
                   ."Customer comments: \n".$feedback."\n";
    $fromaddress = "webserver@bobsdomain.com";

    mail($toaddress, $subject, $mailcontent, $fromaddress);
?>
<html>
<head>
   <title>Bob's Auto Parts - Feedback Submitted</title>
</head>
<body>
<h1>Feedback submitted</hl>
<p>Your feedback has been sent.</p>
</body>
</html>
```

Обратите внимание, что в общем случае необходимо проверить заполнение пользователями всех обязательных полей формы, воспользовавшись, например, функцией **isempty()**. Для краткости в приведенном сценарии и других примерах эта процедура опускается.

Как видно из этого сценария, поля формы объединены, а PHP-функция **mail()** применяется для отправки их по электронной почте по адресу **feedback@bobsdomain.com.** До сих пор функция **mail()** еще не использовалась, поэтому давайте рассмотрим, как она работает.

Совершенно очевидно, что эта функция отправляет сообщение электронной почты. Ее прототип выглядит следующим образом:

```
bool mail(string кому, string тема, string сообщение,
   string [дополнительные_заголовки]
```

Первых три параметра обязательны и представляют, соответственно, адрес, по которому должно быть отправлено сообщение, строку темы и содержимое сообщения. Четвертый параметр может использоваться для отправки любых дополнительных допустимых заголовков сообщения электронной почты. Эти заголовки описаны в документе RFC822, который доступен в Internet. (RFC, или Requests For Comment (Запросы на комментарии), — источник многих стандартов Internet. Эти документы рассматриваются в главе 17.) В данном примере четвертый параметр был использован для включения в сообщение адреса "**From:**" ("От:"). Его можно также применять для добавления таких полей, как "**Reply-To:**" ("Ответить:") и "**Cc:**". Если требуется ввести более одного дополнительного заголовка, их нужно просто разделить символами новой строки (\n), как показано в следующем примере:

```
$additional_headers="From:    webserver@bobsdomain.com\n"
                    ."Reply-To: bob@bobsdomain.com";
```

Чтобы можно было воспользоваться функцией email(), в установке PHP следует указать применяемую программу электронной почты. Если в приведенном виде сценарий не работает, внимательно прочтите приложение А.

В ходе данной главы этот основной сценарий будет совершенствоваться за счет добавления функций обработки строк и регулярных выражений PHP.

Форматирование строк

Часто строки, вводимые пользователем (как правило, из интерфейса HTML-формы), придется приводить в порядок, прежде чем их можно будет использовать.

Усечение строк: функции chop(), ltrim() и trim()

Первый шаг по приведению строк в порядок — удаление из них любых лишних пробелов. Хотя это и не обязательно, но может оказаться полезным, если предполагается хранение строки в файле или базе данных либо ее сравнение с другими строками.

Для этой цели в PHP существуют три полезных функции. Мы используем функцию **trim()** для приведения вводимых данных в порядок следующим образом:

```
$name=trim($name);
$email=trim($email);
$feedback=trim($feedback);
```

Функция **trim()** удаляет пробелы в начале и конце строки и возвращает результирующую строку. При этом символами пробелов считаются символы новой строки (\n), возврата каретки (\r), символы горизонтальной (\t) и вертикальной табуляции (\v), конца строки (\0) и обычные пробелы.

В зависимости от конкретной преследуемой цели вместо этой функции можно использовать функции **ltrim()** или chop(). Обе эти функции аналогичны функции **trim()** в том, что принимают интересующую строку в качестве параметра и возвращают отформатированную строку. Различие между ними состоит в том, что функция trim() удаляет пробелы в начале и конце строки, **ltrim()** удаляет пробелы только в начале строки (или левой ее части), а **chop()** — только в конце (или правой части) строки.

Форматирование строк для представления

PHP имеет набор функций, которые можно использовать для переформатирования строк различными способами.

Использование HTML-форматирования: функция nl2br()

Функция **nl2br()** принимает строку в качестве параметра и заменяет в ней все символы новой строки дескриптором **
** HTML. Это полезно при выводе длинной строки в окне браузера. Например, мы использовали эту функцию для форматирования отзыва клиента с целью вывода его на экране:

```
<p>Your feedback (shown below) has been sent.</p>
<pX? echo nl2br($mailcontent); ?> </p>
```

Вспомните, что HTML не обращает внимания на обычные пробелы, поэтому если не подвергнуть этот вывод обработке функцией **nl2br()**, он отобразится в виде единой строки (за исключением символов новой строки, сгенерированных браузером). Результат показан на рис. 4.2.

Форматирование строк для печати

До сих пор мы использовали языковую конструкцию **echo** для вывода строк в окне браузера.

PHP поддерживает также функцию **print()**, выполняет ту же задачу, что и **echo, но** поскольку она является функцией, то возвращает значение (0 или 1, в зависимости от успешности выполнения).

Обе эти конструкции выводят строку "как есть". Используя функции **printf() и sprintf()**, можно применять несколько более сложное форматирование. В основном, эти функции работают практически одинаково, но **printf()** выводит сформатированную строку в окне браузера, а **sprintfO** возвращает сформатированную строку.

Те читатели, которым ранее приходилось программировать на C, найдут, что эти функции идентичны своим C-версиям. В противном случае, чтобы привыкнуть к использованию этих функций, потребуется некоторое время, однако они весьма полезны и предоставляют большие возможности.

Прототипы этих функций выглядят так:

```
string sprintf (string format [, mixed args ...])
int printf (string format [, mixed args ...])
```

В качестве первого параметра обеим этим функциям передается строка формата, описывающая основную форму вывода, в которой вместо переменных используются коды форматирования. Остальными параметрами являются переменные, которые будут подставляться в строку формата.

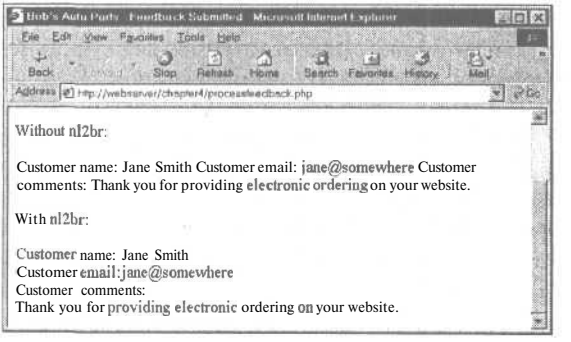


РИСУНОК 4.2 *Использование PHP-функции nl2br() способствует улучшению представления длинных строк HTML.*

Например, в конструкции **echo** мы использовали переменные, которые нужно вывести в строке, подобно следующему:

```
echo "Total amount of order is $Total.";
```

Для получения того же результата с помощью функции **printf()** следовало бы применить

```
printf ("Total amount of order is %s.", $total);
```

Последовательность %s в строке формата называется спецификацией преобразования. Приведенная спецификация означает "заменить строкой". В данном случае она будет заменена значением переменной **$total**, интерпретируемым в качестве строки.

Если бы значением, хранящимся в переменной **$total**, было 12.4, оба приведенные выражения привели к выводу 12.4.

Преимущество использования функции **printf()** в том, что можно использовать более удобную спецификацию преобразования для указания того, что в действительности **$total** — число с плавающей точкой и оно должно содержать два знака после десятичной точки, как показано в следующем примере:

```
printf ("Total amount of order is %.2f", $total);
```

Строка формата может содержать несколько спецификаций преобразования. При использовании п спецификаций преобразования после строки формата необходимо указать п аргументов. Каждая спецификация преобразования будет замещена переформатированным аргументом в том порядке, в каком они перечислены. Например:

```
printf ("Total amount of order is %.2f (with shipping %.2f) ", $total,
  $total_shipping);
```

В данном случае первая спецификация преобразования будет использовать переменную **$total**, а вторая — переменную **$total_shipping**.

Все спецификации преобразования имеют одинаковый формат

```
%['дополняющий_символ][-][ширина][.точность]тип
```

Все спецификации преобразования начинаются с символа %. Если требуется вывести символ %, потребуется применить последовательность %%.

Дополняющий_символ необязателен. Он будет использоваться для дополнения переменной до указанной ширины. Примером может служить добавление ведущих нулей к такому числу, как значение счетчика.

Символ "-" также является необязательным. Он указывает, что данные в поле будут выравниваться по левому краю, а не по правому, как определено по умолчанию.

Параметр *ширина* указывает функции **printf()**, сколько места (в символах) необходимо для подстановки значения переменной.

Параметр *точность* должен начинаться с десятичной точки. Он определяет количество отображаемых десятичных знаков после запятой.

Последним параметром спецификации является код типа. Краткое описание используемых кодов приведено в табл. 4.1.

Вероятно, читатели обратили внимание, что при выводе или повторении строк в окне браузера используется ряд специальных символов типа **\n**. Они служат для записи в вывод специальных символов. Так, \п — символ новой строки. Кроме того, будут встречаться такие базовые символы, как \t, или символ табуляции, и \s, или символ пробела.

Таблица 4.1 Коды типа спецификации преобразования

Тип	Описание
b	Интерпретируется как целое число и выводится как двоичное число.
c	Интерпретируется как целое число и выводится как символ.
d	Интерпретируется как целое число и выводится как десятичное число.
f	Интерпретируется как число двойной точности и выводится как число с плавающей точкой.
o	Интерпретируется как целое число и выводится как восьмеричное число.
s	Интерпретируется как строка и выводится как строка.
x	Интерпретируется как целое число и выводится как шестнадцатеричное число с представлением цифр a-f строчными буквами.
X	Интерпретируется как целое число и выводится как шестнадцатеричное число с представлением цифр A-F прописными буквами.

Изменение регистра строки

Можно также изменять регистр строки. Эта возможность не особенно полезна для нашего приложения, но, тем не менее, мы рассмотрим несколько кратких примеров.

Применительно к строке темы, **$subject**, используемой для сообщения электронной почты, регистр можно изменять с помощью нескольких функций. Их действие кратко описано в табл. 4.2.

Таблица 4.2 Функции изменения регистра строки и их действие

Функция	Описание	Использование	Результат действия
		$subject	Feedback from web site
strtouppper()	**Преобразует символы строки в прописные буквы**	strtouppper($subjecf)	FEEDBACK FROM WEB SITE
strtolower()	**Преобразует символы строки в строчные буквы**	strtolower($subject)	feedback from web site
ucfirst()	**Делает первый символ строки прописным, если он является алфавитным**	ucfirst($subject)	Feedback from web site
ucwords()	**Делает прописным первый символ каждого слова в строке, которая начинается с алфавитного символа.**	ucwords($subject)	Feedback From Web Site

Форматирование строк для хранения: функции AddSlashes() и StripSlashes()

Кроме использования строковых функций для визуального преобразования строки, некоторые из этих функций можно использовать для преобразования строки с целью ее сохранения в базе данных. Хотя сама запись в базу данных будет освещена лишь в части II, в этой главе мы рассмотрим форматирование строк для хранения в базе данных.

Определенные символы являются очень важной частью строки, но могут приводить к проблемам, в особенности при вставке в базу данных, поскольку она может интерпретировать их как управляющие символы. Такими символами являются кавычки (одинарные и двойные), символы обратной косой черты (\) и символ **NUL**.

Нам требуется найти способ пометки, или *отмены* этих символов, чтобы такие базы данных, как MySQL, могли понять, что мы имеем в виду буквальное представление специального символа, а не его интерпретацию в качестве управляющей последовательности. Чтобы *отменить* эти символы, перед ними необходимо поместить символ косой черты. Например, " (двойная кавычка) превращается в \", а \ — в \\. (Это правило применяется ко всем специальным символам, поэтому при наличии в строке символов \\ их следует заменить последовательностью \\\\.)

В PHP имеются две функции, которые специально предназначены для отмены символов. Прежде чем записать любые строки в базу данных, их нужно переформатировать с помощью функции **AddSlashes()**. Например:

```
$feedback = AddSlashes($feedback);
```

Подобно многим другим строковым функциям, **AddSlashes()** принимает строку в качестве параметра и возвращает переформатированную строку.

При использовании функции **AddSlashes()** строка будет сохранена в базе данных с символами косой черты. При получении строки нужно не забыть удалить символы косой черты. Это можно сделать с помощью функции **StripSlashes()**:

```
$feedback = StripSlashes($feedback);
```

Фактическое действие использования этих функций применительно к строке показано на рис. 4.3.

Можно также установить, чтобы PHP добавлял и удалял символы автоматически. Это свойство называется магическими кавычками. Подробнее оно описывается в главе 21.

Объединение и разделение строк с помощью строковых функций

Часто требуется просматривать части строк по отдельности. Например, может потребоваться просмотреть слова в предложении (например, для проверки правописания) или разделить имя домена или адрес электронной почты на их компоненты. PHP предоставляет несколько строковых функций (и одну функцию регулярного выражения), которые позволяют делать это.

В рассматриваемом примере Бобу нужно, чтобы любой отзыв клиента из **bigcustomer.com** поступал непосредственно к нему, поэтому мы разделим адрес электронной почты, введенный клиентом, на составляющие его части для выяснения, того, не соответствуют ли они важному клиенту Боба.

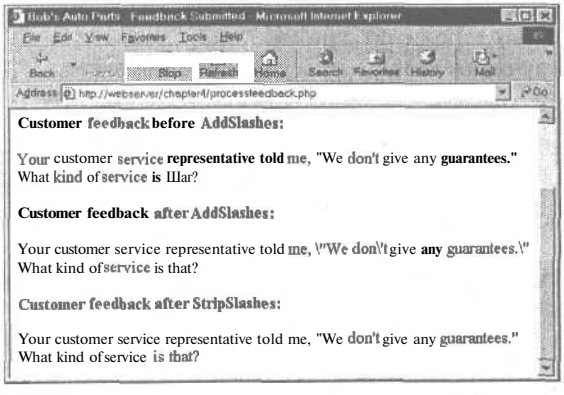


РИСУНОК 4.3
После вызова функции AddSlashes() все кавычки предваряются символами косой черты. Функция StripSlashes() удаляет символы косой черты.

Использование функций explode(), implode() и join()

Первая функция, **explode(),** которую можно было бы задействовать для этой цели, имеет следующий прототип:

```
array explode(string separator, string input);
```

Эта функция принимает строку ***input*** и разделяет ее на части по указанной разделительной строке ***separator.*** Части строки возвращаются в виде массива.

Для получения имени домена из адреса электронной почты в сценарии можно использовать следующий код:

```
$email_array = explode("@", $email);
```

В результате этого вызова функции **explode()** адрес электронной почты разделяется на две части: имя пользователя, который сохраняется в **$email_array[0]**, и имя домена, которое сохраняется в **$email_array[1]**. Теперь можно проверить имя домена для определения источника сообщения клиента и для его переадресации соответствующему лицу:

```
if    ($email_array[1]=="bigcustomer.com")
   $toaddress  =  "bob@bobsdomain.com";
else
   $toaddress  =  "feedback@bobsdomain.com";
```

Обратите внимание, что если имя домена содержит прописные буквы, этот подход не будет работать. Этой проблемы можно было бы избежать, преобразовав все буквы имени домена в прописные или строчные, а затем выполнив проверку:

```
$email_array[1]  =  strtoupper  ($email_array[1]);
```

Эффекта, противоположного действию функции **explode()**, можно добиться, используя функции **implode()** или **join()**, которые идентичны. Например:

```
$new_email  =  implode("@",  $email_array);
```

Эта функция принимает элементы из массива **$email_array** и объединяет их со строкой, переданной в первом параметре. Вызов этой функции во много подобен вызову функции **explode(),** но ее действие противоположно.

Использование функции strtok()

В отличие от функции **explode()**, которая разделяет на части сразу всю строку, функция **strtok()** получат фрагменты (называемые лексемами) из строки по одному. Эта функция — полезная альтернатива использованию функции **explode()** при обработке слов из строки по одному.

Прототип функции **strtok()** имеет следующий вид:

```
srting strtok(string input, string separator);
```

В качестве разделителя ***separator*** можно использовать символ или строку символов, но следует иметь в виду, что строка ввода будет разделяться по каждому из символов разделителя, а не по всей строке разделителя (как имеет место в функции **explode).**

Вызов функции **strtok()** не столь прост, как может показаться, судя по прототипу.

Для получения первой лексемы из строки нужно вызвать функцию strtokQ со строкой, которую требуется разбить на лексемы, и разделителем. Для получения после-

дующих лексем из строки достаточно передать функции единственный параметр – разделитель. Функция сохраняет собственный внутренний указатель на позицию в строке. Если требуется переустановить указатель, можно снова передать строку в функцию.

Как правило, функция **srtok()** используется следующим образом:

```
$token = strtok($feedback, " ");
echo $token."<br>";
while ($token!="")
{
   $token = strtok(" ");
   echo $token."<br>";
};
```

Как обычно, имеет смысл проверить, например, с помощью функции **emty()**, действительно ли пользователь заполнил какие-либо поля в форме. Для краткости эти процедуры в примерах опускаются.

Приведенный фрагмент кода выводит каждую лексему отзыва клиента в отдельной строке и выполняет циклы вплоть до исчерпания лексем. Обратите внимание, что PHP-функция **strtok()** работает не совсем так, как в C. При наличии в исходной строке двух следующих *подряд* разделителей (в данном примере — двух следующих подряд пробелов) функция **strtok()** возвращает пустую строку. Ее нельзя отличить от пустой строки, возвращаемой по достижении конца исходной строки. Кроме того, если одной из лексем является 0, функция также вернет пустую строку. В связи с этим PHP-функция **strtok()** несколько менее полезна, чем в C. Часто лучше просто применять функцию **explode()**.

Использование *функции substr()*

Функция **substr()** позволяет получить доступ к подстроке между заданными начальной и конечной позициями в строке. Для нашего примера она не подходит, но может оказаться полезной, если требуется разделить на части фиксированные отформатированные строки.

Функция **substr()** имеет следующий прототип:

```
string substr(string строка, int начало, int [длина] );
```

Эта функция возвращает подстроку из строки ***строка.***

Давайте рассмотрим примеры использования следующей тестовой строки:

```
$test = "Your customer service is excellent";
```

Если вызвать ее с положительным числом в качестве параметра ***начало*** (единственного), мы получим строку, начиная с позиции ***начало*** и до конца строки. Например,

```
substr($test, 1);
```

возвращает строку **"our customer service is excellent".** Обратите внимание, что нумерация позиций строки начинается с 0, как в массивах.

Если вызвать функцию **substr()** только с отрицательным параметром ***начало,*** мы получим строку, состоящую из символов в конце строки, количество которых определено параметром ***начало.*** Например,

```
substr($test, -9);
```

возвращает подстроку **"excellent".**

Параметр *длина* может использоваться для указания либо количества символов, которые должны быть возвращены (если он положителен), либо последнего символа возвращаемой последовательности (если он отрицателен). Например,

```
substr($test, 0, 4);
```

возвращает первых четыре символа строки, а именно **"Your".** Следующий код:

```
echo substr($test, 4, -13);
```

возвращает символы, расположенные между четвертым символом от начала и тринадцатым символом от конца строки, т.е. **"customer service".**

Сравнение строк

До сих пор мы использовали только операцию == для сравнения двух строк на предмет их равенства. В PHP можно выполнять некоторые более сложные операции сравнения. Мы разделили эти операции на две категории: частичное совпадение и все прочие. В начале мы рассмотрим прочие функции, а затем функции установления частичного совпадения, которые потребуются для дальнейшей разработки примера интеллектуальной формы Smart Form.

Упорядочение строк: функции strcmp(), strcasecmp() и strnatcmp()

Эти функции могут использоваться для упорядочения строк, которое полезно при сортировке данных.

Прототип функции **strcmp()** имеет вид

```
int strcmp(string str1, string str2);
```

Функция принимает две строки, которые и будет сравнивать. Если они равны, функция вернет значение 0. Если в лексикографическом порядке строка **str1** следует за строкой **str2** (или больше ее), функция **strcmp()** вернет положительное число. Если строка **str1** меньше строки **str2**, функция **strcmp()** вернет отрицательное число. Эта функция является чувствительной к регистру.

Функция **strcasecmp()** идентична предыдущей за исключением того, что она не чувствительна к регистру.

Функция **strnatcmp()** и ее нечувствительный к регистру аналог **srtnatcasecmp()** появились в PHP 4. Эти функции сравнивают строки в соответствии с "естественным упорядочением", которое более привычно для человека. Например, **strcmp()** расположила бы строку "2" после строки "12", поскольку лексикографически она больше. Функция **strnatcm()** расположила бы эти строи в обратном порядке. Подробнее естественное упорядочение описано по адресу **http://www.linuxcare.com.au/projects/natsort/**

Проверка длины строки с помощью функции strlen()

Длину строки можно проверить с помощью функции **strlen()**. Если передать этой функции строку, она вернет ее длину. Например, **strlen("hello")** возвращает значение 5.

Это можно задействовать при проверке правильности вводимых данных. Давайте рассмотрим адрес электронной почты в создаваемой форме, хранящийся в переменой **$email**. Один из основных способов проверки правильности адреса электронной почты, хранящегося в переменной **email** — проверка его длины. По мнению авторов минимальная длина адреса электронной почты равна шести символам — например, если

полный адрес содержит код страны без какого-либо домена второго уровня, однобуквенное имя домена и однобуквенный адрес электронной почты, он может иметь вид **a@a.to.** Следовательно, программа могла бы генерировать сообщение об ошибке, если длина адреса оказывается меньшей:

```
if (strlen($email) < 6)
{
   echo "That email address is not valid";
   exit;  // завершение выполнения сценария PHP
}
```

Конечно, это предельно упрощенный способ проверки правильности информации. В следующем разделе рассматривается более рациональный способ.

Сопоставление и замена подстрок с помощью строковых функций

Часто нужно проверить наличие конкретной подстроки в более длинной строке. Обычно это частичное сопоставление более полезно, нежели проверка строк на предмет равенства.

В нашем примере Smart Form требуется выполнить поиск на предмет наличия ключевых фраз в отзыве пользователя и отправить сообщение электронной почты в соответствующее подразделение. Если сообщения электронной почты, в которых речь идет о магазинах Боба, нужно направлять менеджеру розничной продажи, требуется знать, присутствует ли в сообщении слово **"shop"** (или его производные).

Располагая уже рассмотренными функциями, можно было бы использовать функции **explode() или strtok()** для получения отдельных слов сообщения, а затем сравнить их, используя операцию == или функцию **strcmp()**.

Однако, это же можно сделать с помощью единственного вызова к одной из функций сопоставления строк или регулярных выражений. Эти функции используются для поиска определенной последовательности внутри строки. Мы рассмотрим каждый из наборов функций.

Поиск *строк* в *строках: функции strstr(), strchr(), strrchr(), stristr()*

Для поиска строки внутри другой строки можно использовать любую из функций **strstr(), strchr(), strrchr() или stristr()**.

Функция **strstr()** является наиболее общей и может использоваться для поиска соответствующей строки или символа внутри более длинной строки. Следует отметить, что в PHP функция **strchr()** полностью совпадает с функцией **strstr()**, хотя ее имя предполагает, что она применяется для поиска символа в строке, аналогично ее С-версии. В PHP любая из этих функций может использоваться для поиска строки внутри строки, в том числе для поиска строки, состоящей только из одного символа.

Функция **strstr()** имеет следующий прототип:

```
string strstr(string haystack, string needle);
```

В качестве параметра функции передается строка ***haystack,*** в которой нужно выполнить поиск, и строка ***needle,*** которую требуется найти. В случае обнаружения полного соответствия со строкой ***needle*** функция возвращает часть строки ***haystack,*** начинающуюся со строки ***needle;*** в противном случае она возвращает значение **false.** Если строка ***needle*** встречается более одного раза, возвращаемая строка будет начинаться с первого вхождения сроки ***needle.***

Например, в приложении Smart Form решение об адресации сообщения электронной почты можно принять следующим образом:

```
$toaddress = "feedback@bobsdomain.com"; // значение по умолчанию

// Изменение переменной $toaddress при соответствии критериям
if (strstr($feedback, "shop"))
    $toaddress = "retail@bobsdomain.com";
else if (strstr($feedback, "delivery"))
    $toaddress = "fulfilment@bobsdomain.com";
else if (strstr($feedback, "bill"))
    $toaddress = "accounts@bobsdomain.com";
```

Этот код выполняет проверку отзыва на предмет наличия определенных ключевых слов и отправляет сообщение электронной почты соответствующему лицу. Если, например, отзыв клиента звучит как "I still haven't received delivery of my last order", в нем будет найдена строка **"delivery",** и отзыв будет отправлен по адресу **fulfilment@bobsdomain.com**.

Существует два варианта функции **strstr()**. Первый — функция **stristr()**, которая практически идентична предыдущей, но не чувствительна к регистру. Это удобно в данном приложении, поскольку клиент мог бы ввести **"delivery", "Delivery"** или **"DELIVERY"**.

Второй вариант — функция **strrchr()**, которая также почти идентична функции **strstr(),** но будет возвращать часть строки ***haystack,*** начиная с последнего вхождения строки ***needle.***

Определение позиции подстроки: функции strpos(), strrpos()

Функции **strpos()** и **strrpos()** действуют аналогично функции **strstr()** за исключением того, что вместо подстроки они возвращают числовую позицию строки ***needle*** внутри строки ***haystack.***

Функция **strpos()** имеет следующий прототип:

```
string strpos(string haystack, string needle, int [offset] );
```

Целочисленное возвращаемое значение представляет *первое* вхождение строки *needle* внутри строки ***haystack.*** Как обычно, первый символ размещается в нулевой позиции.

Например, следующий код повторит значение 4 в окне браузера:

```
$test = "Hello world";
echo strpos($test, "o");
```

В данном случае в качестве строки ***needle*** был передан единственный символ, но ею может быть строка любой длины.

Необязательный параметр **offset** используется для указания позиции внутри строки ***haystack,*** с которой должен начинаться поиск. Например,

```
echo strpos($test, "o", 5);
```

Этот код повторит значение 7 в окне браузера, поскольку PHP начинает поиск символа о с 5 позиции и, следовательно, не видит этот символ в позиции 4.

Функция **strrpos()** действует почти так же, но будет возвращать позицию последнего вхождения строки ***needle*** в строке ***haystack.*** В отличие от функции **strposQ,** эта функция работает только с одно-символьной строкой ***needle.*** Поэтому, если передать строку как ***needle,*** *для* сопоставления функция будет использоваться только первый символ строки.

В любом из этих случаев, если *needle* не является строкой, функция **strpos()** или **strrpos()** будет возвращать значение **false.** Это может порождать проблемы, поскольку

в слабо типизированном языке наподобие PHP значение **false** эквивалентно 0, т.е., первому символу в строке.

Во избежание этой проблемы можно использовать операцию === для проверки возвращаемых значений:

```
$result = strpos($test, "H");
if ($result === false)
   echo "Not found"
else
   echo "Found at position 0";
```

Обратите внимание, что этот метод подходит только для PHP 4 — в предшествующих версиях эту проверку можно выполнить, проверяя, является ли возвращаемое значение строкой (т.е., значением **false).**

Замена *подстрок: str_replace(), substr_replace()*

Функциональные возможности поиска и замены могут оказаться исключительно полезными и при работе со строками. Ранее мы уже использовали поиск и замену для персонификации документов, сгенерированных PHP, например, заменяя ***«name»*** именем конкретного лица, а ***«address»*** — его адресом. Эти функции можно использовать также для цензуры определенных терминов, например, в приложении дискуссионного форума или даже в приложении Smart Form.

Для выполнения этой задачи можно опять воспользоваться строковыми функциями или функциями обработки регулярных выражений.

Строковая функция, которую чаще всего используют для замены — **str_replace()**. Она имеет следующий прототип:

```
string str_replace(string needle, string new_needle, string haystack);
```

Эта функция будет замещать все экземпляры строки ***needle*** в строке ***haystack*** на строку ***new_needle***.

Например, поскольку пользователи могут использовать приложение Smart Form для выражения своего недовольства, они могут прибегать к некоторым выразительным словам. Будучи программистами, мы можем избавить персонал различных подразделений компании Боба от всякого рода оскорблений:

```
$feedback = str_replace($offcolor, "%!@*", $feedback);
```

Функция **substr_replace()** используется для поиска и замены конкретной подстроки в строке. Она имеет следующий прототип:

```
string substr_replace(string needle, string replacement,
   int start, int [length] ) ;
```

Эта функция будет замещать часть строки ***string*** строкой ***replacement.*** Какую именно часть — зависит от значений параметра ***start*** и необязательного параметра ***length.***

Значение ***start*** представляет смещение позиции в строке, с которого должна начинаться замена. Если оно равно 0 или положительно, то определяет смещение от начала строки, если отрицательно — смещение от конца строки. Например, следующая строка кода будет заменять последний символ в переменной **$test** символом "X":

```
$test = substr_replace($test, "X", -1) ;
```

Параметр ***length*** необязателен и представляет позицию, в которой замена должна быть прекращена. Если это значение не указано, замена будет выполняться от позиции, определенной параметром ***start,*** и до конца строки.

Если значение ***length*** равно нулю, фактически строка замены будет *вставляться* в строку без замещения существующей строки.

Положительное значение параметра ***length*** представляет количество символов, которые требуется заменить новой строкой.

Отрицательное значение параметра *length* представляет позицию, в которой нужно прекратить замену символов, определяемую от конца строки.

Введение в регулярные выражения

PHP поддерживает два стиля синтаксиса регулярных выражений: POSIX и Perl. Стиль POSIX регулярных выражений компилируется в PHP по умолчанию, а стиль Perl можно активизировать, выполнив компиляцию с библиотекой PCRE (Perl-compatible regular expression — Perl-совместимые регулярные выражения). Мы рассмотрим более простой стиль POSIX, но программисты Perl или читатели, которые желают больше узнать о PCRE, могут прочесть интерактивное руководство на сайте **http://php.net.**

До сих пор все действия по сопоставлению шаблонов выполнялись с использованием строковых функций. Эти действия ограничивались случаями точного соответствия строк или подстрок. Например, в предшествующих примерах отыскивались такие члены регулярных выражений, как **"shop"** или **"delivery".**

В PHP сопоставление регулярных выражений больше подобно сопоставлению с помощью функции **strstr()**, чем сравнению на предмет равенства, поскольку выполняется сопоставление строки, находящейся где-то внутри другой строки. (Строка может находиться в любом месте строки, если только не указано иначе.) Например, строка **"shop"** соответствует регулярному выражению "shop". Она соответствует также регулярным выражениям "h", "ho" и т.д.

В дополнение к точному сопоставлению символов можно использовать специальные символы для указания метазначений.

Например, с помощью специальных символов можно указать, что шаблон должен встречаться в начале или конце строки, что часть шаблона может повторяться или символы в шаблоне должны иметь конкретный тип. Можно также сопоставлять литеральные значения специальных символов. Мы рассмотрим все упомянутые случаи.

Наборы символов и классы

Использование наборов символов немедленно расширяет спектр возможностей регулярных выражений по сравнению с выражениями точного сопоставления. Наборы символов могут использоваться для сопоставления любого символа конкретного *типа* — фактически, они являются одним из видов группового символа.

Прежде всего, символ "." можно использовать в качестве группового символа для любого другого одиночного символа, за исключением символа новой строки (**\n**). Например, регулярное выражение

```
.at
```

соответствует, в частности, строкам **"cat",** "sat" и "mat".

Этот вид сопоставления групповых символов часто используется для сопоставления имен файлов в операционных системах.

Однако, используя регулярные выражения, можно точнее указывать тип символов, которые нужно сопоставить, и можно действительно указывать набор, к которому должен принадлежать символ. В предыдущем примере регулярное выражение со-

ответствует строкам "mat" и "cat", но оно соответствует также и строке "#at". Если соответствие необходимо ограничить символами от а до *г,* его можно указать следующим образом:

```
[a-z]
```

Все, заключенное в специальные символы квадратных скобок [и], представляет класс символов — набор символов, к которому должен принадлежать сопоставляемый символ. Обратите внимание, что заключенное в квадратные скобки выражение сопоставляется только с *одиночным символом.*

Набор можно указать в виде списка. Например,

```
[aeiou]
```

означает любую гласную английского алфавита.

Можно описать также диапазон, как это было только что сделано с помощью символа дефиса, или набор диапазонов:

```
[a-zA-Z]
```

Этот набор диапазонов означает любой строчный или прописной символ английского алфавита.

Наборы можно использовать также для указания символа, который не может быть членом набора. Например,

```
[^a-z]
```

соответствует любому символу, *не* относящемуся к диапазону a-z. Когда он помещен внутрь квадратных скобок, символ вставки ^ означает *"не".* При использовании вне квадратных скобок он имеет другое значение, что будет вскоре рассмотрено.

В дополнение к наборам перечисления и диапазонам в регулярных выражениях может использоваться ряд предопределенных *классов символов,* которые перечислены в табл. 4.3.

Таблица 4.3 Классы символов, используемые в регулярных выражениях стиля POSIX

Класс	*Соответствие*
[[:alnum:]]	Алфавитно-цифровые символы
[[:alpha:]]	Алфавитные символы
[[:lower:]]	Строчные буквы
[[:upper:]]	Прописные буквы
[[:digit:]]	Десятичные цифры
[[:xdigit:]]	Шестнадцатеричные цифры
[[:punct:]]	Знаки пунктуации
[[:blanc:]]	Символы табуляции и пробелов
[[:space:]]	Любые символы свободного места
[[:cntrl:]]	Управляющие символы
[[:print:]]	Все печатные символы
[[:graph:]]	Все печатные символы, за исключением символов пробелов

Повторение

Часто требуется указать возможность наличия нескольких вхождений конкретной строки или класса символов. В регулярном выражении это можно сделать с помощью двух специальных символов. Символ * означает, что шаблон может повторяться нуль или более раз, а символ + — 1 или более раз. Эти символы должны указываться непосредственно после той части выражения, к которой они применяются. Например,

```
[[:alnum:]]+
```

означает "по крайней мере один алфавитно-цифровой символ".

Подвыражения

Часто удобно разделять выражение на подвыражения, чтобы, например, можно было представить выражение "по меньшей мере одна из этих строк, за которой следует именно одна из следующих". Это можно сделать с использованием круглых скобок совершено так же, как это делается в арифметических выражениях. Например,

```
(very )*large
```

соответствует строкам **"large", "very large", "very very large" и т.д.**

Подсчитываемые подвыражения

Количество повторений какой-либо строки можно указать с помощью числового выражения, заключенного в фигурные скобки ({}). При этом можно указывать точное число повторений (**{3}** означает в точности 3 повторения), диапазон повторений (**{2,** 4} означает от 2 до 4 повторений) или открытый диапазон повторений (**{2,}** означает не менее двух повторений).

Например,

```
(very ){1, 3}
```

соответствует строкам "very", "very very" и "very very very".

Привязка к началу или концу строки

Можно указать, должно ли конкретное подвыражение появляться в начале, конце или в начале и конце строки. Это достаточно удобно, когда необходимо убедиться, что в строке присутствует только искомый термин и ничего больше.

Символ вставки (^) используется в начале регулярного выражения для указания того, что оно должно присутствовать в начале просматриваемой строки, а символ $ используется в конце регулярного выражения для указания того, что оно должно присутствовать в конце строки.

Например, следующая строка соответствует наличию подстроки bob в начале строки:

```
^bob
```

Эта строка соответствует наличию подстроки com в конце строки:

```
com$
```

И, наконец, это выражение соответствует любому одиночному символ от a до z, расположенному в отдельной строке:

```
^[a-z]$
```

Ветвление

Вариант в регулярном выражении можно представить с помощью символа вертикальной черты (|). Например, если требуется сопоставить строку **com, edu** или net, можно воспользоваться выражением:

```
(com) I (edu) | (net)
```

Сопоставление с литеральными значениями специальных символов

Если нужно сопоставить Один из специальных символов, упомянутых в этом разделе, таких как ., { или $, перед ним необходимо поместить символ косой черты (\). Если нужно представить символ косой черты, его следует заменить двумя символами косой черты, т.е. \\.

Краткое описание специальных символов

Краткое описание всех специальных символов приведено в табл. 4.4 и 4.5. В табл. 4.4 перечислены значения специальных символов вне квадратных скобок, а в табл. 4.5 – их значения внутри квадратных скобок.

Таблица 4.4 Краткое описание специальных символов, используемых в регулярных выражениях POSIX вне квадратных скобок

Символ	*Значение*
\	Символ **отмены**
^	**Соответствие в начале строки**
$	**Соответствие в конце строки**
.	**Соответствие любому символу, за исключением символа новой строки** (\n)
|	**Начало альтернативной ветви**
(	**Начало подшаблона**
)	**Конец подшаблона**
*	**Повторение 0 или более раз**
+	**Повторение 1 или более раз**
{	**Начало указателя минимального/максимального количества повторений**
}	**Конец указателя минимального/максимального количества повторений**

Таблица 4.5 Краткое описание специальных символов, используемых в регулярных выражениях POSIX внутри квадратных скобок

Символ	*Значение*
\	**Символ отмены**
^	**НЕ, только если используется в начальной позиции**
-	**Используется для указания диапазонов символов**

Использование регулярных выражений в приложении Smart Form

В приложении Smart Form регулярные выражения могут найти, по меньшей мере, два потенциальных применения. Во-первых — для выявления конкретных тер-

минов в отзыве клиента. Использование регулярных выражений позволяет выполнить эту задачу несколько проще. При использовании строковой функции для сопоставления строк **"shop", "customer service"** или **"retail"** приходилось выполнять три различных поиска. В случае применения регулярного выражения можно сопоставить сразу все три строки:

```
shop I customer  service|retail
```

Второе применение — проверка правильности адреса электронной почты клиента путем кодирования стандартизованного формата адреса электронной почты в регулярном выражении. Формат включает некоторые алфавитно-цифровые символы и символы пунктуации, за которыми следуют символ @, затем строка алфавитно-цифровых символов или дефисов, затем точка, затем опять алфавитно-цифровые символы и дефисы и, возможно, дополнительные точки, вплоть до конца строки; этот формат можно закодировать следующим образом:

```
^[a-zA-Z0-9_]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$
```

Подвыражение **^[a-zA-Z0-9_]+** означает "начало строки по меньшей мере с одной буквы, цифры или символа подчеркивания, или какого-либо их сочетания".

Символ @ соответствует литеральному значению @.

Подвыражение **[a-zA-ZO-9\-]+** соответствует первой части имени хоста, включающего алфавитно-цифровые символы и дефисы. Обратите внимание, что перед символом дефиса поставлена косая черта, поскольку внутри квадратных скобок он является специальным символом.

Комбинация \. соответствует литеральному значению ".".

Подвыражение **[a-zA-ZO-9\-\.]+$** соответствует остальной части имени домена, включающего буквы, цифры, дефисы и дополнительные точки, если они требуются, вплоть до конца строки.

Небольшой анализ показывает, что можно ввести неправильные адреса электронной почты, которые все же будут соответствовать этому регулярному выражению. Практически невозможно предусмотреть все возможные варианты адресов, но это регулярное выражение позволит все-таки немного улучшить ситуацию.

Теперь, когда читатели познакомились с регулярными выражениями, давайте рассмотрим PHP-функции, в которых они используются.

Поиск подстрок с помощью регулярных выражений

Поиск подстрок — основное применение только что рассмотренных регулярных выражений. В PHP существуют две функции, предназначенные для сопоставления с регулярными выражениями: **ereg() и eregi().**

Функция **ereg()** имеет следующий прототип:

```
int ereg(string pattern, string search, array [matches]);
```

Эта функция выполняет поиск в строке ***search,*** отыскивая в ней соответствия с регулярным выражением, определенным в шаблоне ***pattern.*** Если соответствия подвыражений с шаблоном ***pattern*** будут найдены, они сохраняются в массиве соответствий ***matches,*** по одному подвыражению в каждом элементе массива.

Функция **eregi()** идентична предыдущей, за исключением того, что она не чувствительна к регистру.

Приложение Smart Form можно приспособить к использованию регулярных выражений следующим образом:

```
if    (!eregi("^[a-zA-Z0-9_]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$",
   $email))
{
   echo "That is not a valid email address.  Please return to the"
         ." previous page and try again.";
  exit;
}
$toaddress = "feedback@bobsdomain.com";   // значение по умолчанию
if  (eregi("shop|customer service|retail", $feedback))
   $toaddress = "retail@bobsdomain.com";
else if  (eregi("deliver.*|fulfil.*", $feedback))
   $toaddress = "fulfilment@bobsdomain.com";
else if  (eregi("bill|account", $feedback))
   $toaddress = "accounts@bobsdomain.com";

if  (eregi("bigcustomer\.com", $email))
   $toaddress = "bob@bobsdomain.com";
```

Замена подарок с помощью регулярных выражений

Регулярные выражения можно использовать также для поиска и замены подстрок так же, как это делалось с помощью функции **str_replace()**. Для выполнения этой задачи существуют две функции: **ereg_replace() и eregi_replace()**. Функция **ereg_replace()** имеет следующий прототип:

```
string ereg_replace(string pattern, string replacement, string search);
```

Эта функция ищет регулярное выражение ***pattern*** в строке ***search*** и заменяет его строкой ***replacement.***

Функция **eregi_replace()** идентична предыдущей, но не чувствительна к регистру.

Разделение строк с помощью регулярных выражений

Еще одна полезная функция обработки регулярных выражений — **split(),** которая имеет следующий прототип:

```
array split(string pattern, string search, int [max]);
```

Эта функция разделяет строку ***search*** на подстроки по регулярному выражению *pattern* и возвращает подстроки в массив. Целочисленный параметр ***max*** ограничивает количество элементов, которые могут быть помещены в массив.

Эта функция может оказаться полезной для разделения имен доменов или дат. Например,

```
$domain = "yallara.cs.rmit.edu.au";
$arr = split ("\.", $domain);
while (list($key, $value) = each ($arr))
   echo "<br>".$value;
```

Этот фрагмент кода разделяет имя хоста на пять компонентов и выводит каждый из них в отдельной строке.

Сравнение строковых функций и функций регулярных выражений

В общем случае функции регулярных выражений выполняются менее эффективно, чем строковые функции с аналогичными возможностями. Если приложение достаточно простое, чтобы можно было использовать строковые функции, их следует использовать.

Дополнительная информация

Регулярным выражениям посвящено огромное количество источников. Если вы используете UNIX, знакомство с ними можно начать со страницы интерактивного руководства, посвященной **regexp**. Кроме того, несколько огромных статей находятся на сайтах **devshed.com и phpbuilder.com**.

На Web-сайте компании Zend можно ознакомиться с более сложной и предоставляющей большие возможности функцией проверки правильности сообщений электронной почты, чем те, которые рассматривались в этой главе. Она называется **MailVal()** и доступна по адресу **http://www.zend.com/codex.php?id=88&single=1**.

На освоение регулярных выражений потребуется некоторое время — чем больше примеров вы рассмотрите и выполните, тем увереннее будете чувствовать себя по отношению к регулярным выражениям.

Что дальше

В следующей главе мы рассмотрим несколько способов использования PHP для уменьшения трудозатрат и времени, затрачиваемых на программирование, и для предотвращения избыточности за счет повторного использования существующего кода.

Глава 5

Повторное использование кода и создание функций

В этой главе поясняется, каким образом повторное использование кода способствует созданию более единообразных, надежных, пригодных к внесению изменений программ при меньших затрачиваемых на это усилиях. В ней будут описаны технологии создания модулей и повторного использования кода, начиная с простого использования операторов **require()** и **include()** для использования одного и того же кода на нескольких страницах. Мы покажем, в чем эти операторы превосходят включения на стороне сервера. Приведенные примеры продемонстрируют использование включеных файлов для обеспечения единообразного вида во всем сайте.

Мы поясним, как создавать и вызывать пользовательские функции, используя в качестве примеров функции генерации страниц и форм.

В этой главе освещаются следующие вопросы:

- Для чего требуется повторное использование кода?
- Использование операторов **require()** и **include()**
- Введение в функции
- Для чего нужно определять собственные функции?
- Базовая структура функции
- Параметры
- Возврат значений
- Передача по ссылке и передача по значению
- Область действия
- Рекурсия

Для чего требуется повторное использование кода?

Одна из целей, к которой стремятся разработчики программного обеспечения, – повторное использование кода вместо создания нового кода. Это связано вовсе не с тем, что разработчики программного обеспечения отличаются особой ленью. Повторное использование существующего кода снижает стоимость, повышает надежность и единообразие программ. В идеале новый проект создается путем объединения существующих пригодных для повторного использования компонентов при минимальном объеме разработки с нуля.

Стоимость

В течение эффективного срока службы какой-либо части программного обеспечения поддержка, модифицирование, тестирование и документирование будут занимать значительно больше времени, чем ее первоначальное создание. При создании коммерческой программы следует попытаться ограничить количество ее вариантов, используемых в организации. Один из наиболее практичных способов достижения этой цели — повторное использование существующего кода вместо создания несколько иной версии этого же кода для выполнения новой задачи. Меньший объем кода означает меньшую стоимость. Если уже существует программное обеспечение, которое отвечает требованиям нового проекта, следует использовать его. Стоимость приобретения существующего программного обеспечения почти всегда меньше стоимости разработки эквивалентной программы. Тем не менее, к использованию существующего программного обеспечения, которое *почти* соответствует предъявляемым требованиям, следует подходить обдуманно. Иногда модификация существующего кода может оказаться труднее создания нового.

Надежность

Если программный модуль используется в той или иной организации, вероятно, он был тщательно протестирован. Даже если он состоит всего из нескольких строк, при его изменении существует вероятность пропустить что-либо, предусмотренное автором кода или добавленное в первоначальный код после выявления недостатков во время тестирования. Как правило, существующий, проверенный код более надежен, чем новый, "зеленый".

Единообразие

Внешние интерфейсы системы, включая пользовательские интерфейсы и интерфейсы внешних систем, должны быть единообразными. Создание нового кода, который согласован с остальными частями системы, требует целеустремленности и продуманных усилий. При повторном использовании кода, который применяется в другой части системы, единообразие функционирования должно обеспечиваться автоматически.

Кроме этих преимуществ, повторное использование кода означает меньший объем работы для разработчика, по крайней мере, если исходный код имеет модульную структуру и хорошо продуман. В ходе работы над программами старайтесь определить те разделы кода, к которым можно будет снова обращаться в будущем.

Использование операторов require() и include()

PHP предоставляет два простых, но очень полезных оператора, которые позволяют повторно использовать любой тип кода. Используя операторы require() и **include(),** можно загружать файл в PHP-сценарий. Файл может содержать все, что обычно вво-

дится в сценарий, включая PHP-операторы, текст, HTML-дескрипторы, PHP-функции или PHP-классы.

Эти операторы работают аналогично серверным включениям (Server Side Includes), которые обеспечиваются многими Web-серверами, и операторам **#include** в языке C или C++.

Использование оператора require()

Следующий код хранится в файле **reusable.php:**

```
<?
   echo "Here is a very simple PHP statement.<BR>";
?>
```

А этот код хранится в файле **main.php:**

```
<?
   echo "This is the main file.<BR>";
   require( "reusable.php" );
   echo"The script will end now.<BR>";
?>
```

Вероятно, не удивительно, что при загрузке файла **reusable.php** в окне браузера отображается текст **"Here is a very simple PHP statement."** Однако при загрузке файла **main.php** происходит кое-что более интересное. Вывод этого сценария показан на рис. 5.1.

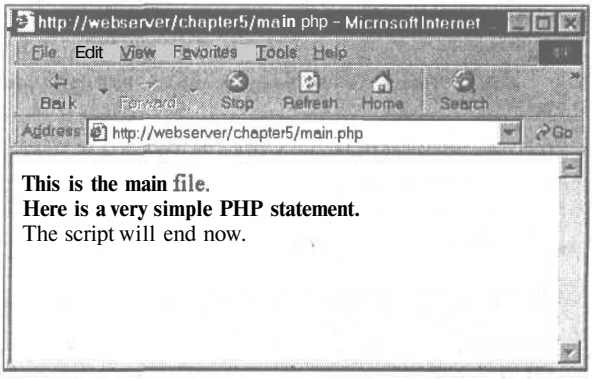


РИСУНОК 5.1
Вывод, создаваемый файлом main.php, отображает результат выполнения оператора require().

Файл вынужден использовать оператор **require()**. В предыдущем примере был использован файл **reusable.php.** При выполнении этого сценария оператор **require()**

```
require( "reusable.php" );
```

заменяется содержимым запрошенного файла, после чего сценарий выполняется. Это означает, что при загрузке файла **main.php** он выполняется так, как если бы сценарий имел следующий вид:

```
<?
   echo "This is the main file.<BR>";
   echo "Here is a very simple PHP statement.<BR>";
   echo"The script will end now.<BR>";
?>
```

При использовании оператора **require()** следует обратить внимание на различия в обработке расширений имен файлов и PHP-дескрипторов.

Расширения имен файлов и оператор require()

PHP не обращает внимания на расширение имени запрашиваемого файла. Следовательно, файл можно называть как угодно, если только он не будет вызываться непосредственно. При использовании оператора **require()** для загрузки файла, файл фактически становится частью PHP-файла и будет выполняться как таковой.

Обычно PHP-операторы не должны выполняться, если они находятся в файле, названном, например, **page.html.** Как правило, PHP вызывается только для анализа файлов с определенными расширениями, такими как **.php.** Однако если загрузить файл **page.html** через оператор **require()**, любые хранящиеся внутри него PHP-операторы будут обработаны. Следовательно, для включенных файлов можно использовать любые расширения, однако имеет смысл придерживаться практичного соглашения и использовать расширения наподобие **.inc**.

При этом следует иметь в виду, что если файлы, имеющие расширение **.inc** или какое-либо другое нестандартное расширение, сохраняются в дереве документов Web, и пользователи непосредственно загружают их в браузеры, они смогут просмотреть код в виде простого текста, что распространяется и на любые пароли. Поэтому важно либо хранить включаемые файлы вне дерева документов, либо использовать стандартные расширения.

PHP-дескрипторы и оператор requireQ

В рассматриваемом примере повторно используемый файл (**reusable.php**) имел следующий вид:

```
<?
   echo "Here is a very simple PHP statement.<BR>";
?>
```

Внутри файла PHP-код был помещен в PHP-дескриптор. Это нужно делать, если требуется, чтобы PHP-код внутри запрошенного файла обрабатывался именно как PHP-код. Если опустить PHP-дескриптор, код будет обрабатываться просто как текст или HTML и выполняться не будет.

Использование оператора require() для шаблонов Web-сайта

Если страницы на Web-сайте данной компании выглядят и действуют единообразно, можно воспользоваться PHP для добавления к странице шаблона и стандартных элементов с помощью оператора **require()**.

Например, Web-сайт вымышленной компании TLA Consulting имеет ряд страниц, которые все выглядят и ведут себя подобно показанной на рис. 5.2.

Когда возникает потребность в новой странице, разработчик может открыть существующую страницу, вырезать существующий текст из файла, ввести новый текст и сохранить файл под новым именем.

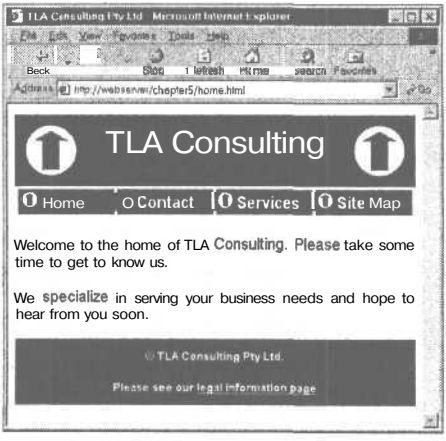


РИСУНОК 5.2 *Все Web-страницы сайта компании TLA Consulting выглядят и ведут себя единообразно.*

Давайте рассмотрим следующий сценарий протекания событий: Web-сайт существует уже некоторое время и теперь содержит десятки, сотни или, возможно, тысячи страниц, которые все выдержаны в одном стиле. Принято решение частично изменить стандартный вид — изменение может быть каким-либо незначительным, например, включение адреса электронной почты в нижний колонтитул или помещение единственной новой записи в меню навигации по страницам. Как вам понравится перспектива вносить небольшое изменение в десятки, сотни или даже тысячи страниц?

Непосредственное использование разделов HTML, общих для всех страниц — значительно более рациональный подход, нежели вырезание и вставка, выполняемая в десятках, сотнях или тысячах страниц. Исходный код начальной страницы (**home.html**), показанной на рис. 5.2, приведен в листинге 5.1.

Листинг 5.1 home.html - HTML-код, создающий начальную страницу компании TLA Consulting

```
<html>
<head>
  <title>TLA Consulting Pty Ltd</title>
  <style>
    h1 {color:white; font-size:24pt; text-align:center;
        font-family:arial,sans-serif}
    .menu {color:white; font-size:12pt; text-align:center;
           font-family:arial,sans-serif; font-weight:bold}
    td {background:black}
    p {color:black; font-size:12pt; text-align:justify;
       font-family:arial,sans-serif}
    p.foot {color:white; font-size:9pt; text-align:center;
            font-family:arial,sans-serif; font-weight:bold}
    a:link,a:visited,a:active{color:white}
  </style>
</head>
<body>

  <!-- page header -->
   <table width="100%" cellpadding = 12 cellspacing =0 border = 0>
  <tr bgcolor = black>
      <td align = left><img src = "logo.gif"></td>
    <td>
        <h1>TLA Consulting</hl>
    </td>
     <td align = right><img src = "logo.gif"></td>
  </tr>
  </table>

  <!-- menu -->
   <table width = "100%" bgcolor = white cellpadding = 4 cellspacing = 4>
   <tr >
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Home</span></td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Contact</span></td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Services</span></td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Site Map</spanX/td>
   </tr>
   </table>
```

```
  <!-- page content -->
  <p>Welcome to the home of TLA Consulting.
 Please take some time to get to know us.</p>
 <p>We specialize in serving your business needs
 and hope to hear from you soon.</p>

 <!-- page footer -->
  <table width = "100%" bgcolor = black cellpadding = 12 border =
  0>
  <tr>
    <td>
      <p class=foot>&copy; TLA Consulting Pty Ltd.</p>
        <p class=foot>Please see our <a href ="">
                    legal information page</aX/p>
     </td>
 </tr>
 </table>
</body>
</html>
```

Как видно из листинга 5.1, в этом файле имеется ряд отдельных разделов кода. HTML-заголовок содержит CSS-определения (Cascading Style Sheet, CSS), используемые страницей. Раздел, озаглавленный "page header" ("верхний колонтитул страницы"), отображает название компании и ее логотип, "menu" ("меню") создает навигационную панель, а "page content" ("содержимое страницы") представляет текст, уникальный для данной страницы. Под ними расположен нижний колонтитул страницы. Вполне можно разделить этот файл и назвать части **header.inc, home.php и footer.inc.** Файлы **header.inc и footer.inc** содержат код, который будет повторно использоваться на других страницах.

Файл **home.php** служит заменой для файла **home.html** и содержит уникальное содержимое страницы и два оператора **require()**, как показано в листинге 5.2.

Листинг 5.2 home.php — **PHP-код**, создающий начальную страницу сайта компании TLA Consulting

```
<?
    require("header.inc");
?>
  <!-- page content -->
  <p>Welcome to the home of TLA Consulting.
  Please take some time to get to know us.</p>
  <p>We specialize in serving your business needs
  and hope to hear from you soon.</p>
<?
    require("footer.inc");
?>
```

Операторы **require()** в файле **home.php** загружают файлы **header.inc и footer.inc.**

Как уже отмечалось, имена, присвоенные этим файлам, не влияют на способ их обработки при вызове через оператор **require()**. Общепринято, но не обязательно, называть частичные файлы, которые предназначены для включения в другие файлы, наподобие ***something*.inc** (в данном случае inc означает include (включить)). Кроме того, общепринято и имеет смысл помещать включаемые файлы в каталог, который доступен сценарию, но не позволяет включаемым файлам быть загружен-

ными самостоятельно через Web-сервер. Это предотвратит самостоятельную загрузку этих файлов, что может а) вызывать какие-либо ошибки, если расширением файла является **.php,** но он содержит только часть страницы или сценария, или b) при использовании другого расширения разрешать пользователям читать исходный код.

Файл **header.inc** содержит CSS-определения, используемые этой страницей, таблицы, которые отображают название компании, и навигационное меню (см. листинг 5.3).

Листинг 5.3 header.inc — повторно используемый верхний колонтитул для всех Web-страниц сайта компании TLA

```
<html>
<head>
  <title>TLA Consulting Pty Ltd</title>
  <style>
    h1 {color:white; font-size:24pt; text-align:center;
        font-family:arial,sans-serif}
    .menu {color:white; font-size:12pt; text-align:center;
           font-family:arial,sans-serif; font-weight:bold}
    td {background:black}
    p {color:black; font-size:12pt; text-align:justify;
       font-family:arial,sans-serif}
    p.foot {color:white; font-size:9pt; text-align:center;
            font-family:arial,sans-serif; font-weight:bold}
    a:link,a:visited,a:active {color:white}
  </style>
</head>
<body>

  <!-- page header -->
   <table width="100%" cellpadding = 12 cellspacing =0 border = 0>
  <tr bgcolor = black>
      <td align = leftximg src = "logo.gif"></td>
    <td>
        <h1>TLA Consulting</hl>
    </td>
     <td align = right><img src = "logo.gif "></td>
  </tr>
  </table>

  <!-- menu -->
   <table width = "100%" bgcolor = white cellpadding = 4 cellspacing
   = 4>
   <tr >
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Home</span></td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Contact</span></td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Services</spanX/td>
      <td width = "25%">
         <img src = "s-logo.gif"> <span class=menu>Site Map</spanX/td>
   </tr>
   </table>
```

Файл **footer.inc** содержит таблицу, которая отображает нижний колонтитул в нижней части каждой страницы. Этот файл показан в листинге 5.4.

Листинг 5.4 footer.inc - повторно используемый нижний колонтитул для всех Web-страниц сайта компании TLA

```
<!-- page footer -->
   <table width = "100%" bgcolor = black cellpadding = 12 border = 0>
   <tr>
     <td>
       <p class=foot>&copy; TLA Consulting Pty Ltd.</p>
       <p class=foot>Please see our
                  <a href ="legal .php3">legal information page</aX/p>
     </td>
  </tr>
  </table>
</body>
</html>
```

Этот подход позволяет очень легко получить единообразно выглядящий Web-сайт, и новую страницу в этом же стиле можно создать, введя что-либо наподобие:

```
<? require("header.inc"); ?>
Here is the content for this page
<? require("footer.inc"); ?>
```

Самое главное, даже после того, как создано множество страниц, использующих этот верхний и нижний колонтитулы, можно легко изменить файлы верхнего и нижнего колонтитула. Независимо от того, вносятся ли незначительные изменения в тексте, или полностью изменяется внешний вид сайта, изменение потребуется внести только один раз. Не нужно изменять каждую страницу сайта в отдельности, поскольку страница загружается в файлы верхнего и нижнего колонтитулов.

В приведенном примере в теле, верхнем и нижнем колонтитулах страниц используется простой HTML-код. Это вовсе не обязательно. Внутри этих файлов можно было бы использовать PHP-операторы для динамической генерации частей страницы.

Использование директив auto_prepend_file и auto_append_file

Если нет желания использовать оператор **require()** для добавления верхнего и нижнего колонтитулов к каждой странице, это можно сделать и другим способом. Файл php.ini содержит два параметра конфигурации: **auto_prepend_file** и **auto_append_file**. Установив в качестве их значений файлы верхнего и нижнего колонтитулов, можно обеспечить загрузку этих файлов, соответственно, перед и после каждой страницы.

Для Windows настройки будут выглядеть как-то так:

```
auto_prepend_file = "C:/inetpub/include/header.inc"
auto_append_file = "C:/inetpub/include/footer.inc"
```

Для UNIX они должны иметь вид:

```
auto_prepend_file = "/home/username/include/header.inc"
auto_append_file = "/home/username/include/footer.inc"
```

При использовании этих директив не нужно вводить операторы **recuire()**, но в этом случае верхние и нижние колонтитулы больше не будут необязательными на страницах.

При использовании Web-сервера Apache различные параметры конфигурации, подобные этим, можно изменять для отдельных каталогов. Для этого сервер должен быть настроен на разрешение замещения его главного файла (файлов) конфигурации. Что-

бы установить автоматическое добавление перед и после файлов для каталога, создайте в нем файл, названный **.htaccess.** Этот файл должен содержать следующие две строки:

```
php_value   auto_prepend_file   "/home/username/include/header.inc"
php_value  auto_append_file  "/home/username/include/footer.inc"
```

Обратите внимание, что синтаксис несколько отличается от синтаксиса этого же параметра в файле php.ini: строка начинается с **php_value** и не содержит знака равенства. Ряд других настроек конфигурации в файле php.ini также можно изменять аналогичным образом.

Синтаксис отличается от такового в PHP 3. При использовании предыдущей версии строки в файле **.htaccess** должны выглядеть следующим образом:

```
php3_auto_prepend_file  "/home/username/include/header.inc"
php3_auto_append_file  "/home/username/include/footer.inc"
```

Установка параметров в файле **.htaccess**, а не в файле php.ini или в файле конфигурации Web-сервера обеспечивает очень большую степень свободы. Можно изменять настройки на компьютере совместного использования, влияющие только на конкретные каталоги. При этом не требуется перезапускать Web-сервер и не нужно иметь права доступа администратора. Недостаток метода с использованием файла **.htaccess** состоит в том, что файлы считываются и анализируются при каждом запросе файла из данного каталога, а не один раз при начальном запуске компьютера, что ведет к снижению производительности.

Использование оператора include()

Операторы **require()** и **include()** весьма подобны, но их работа различается несколькими очень важными особенностями.

Оператор **include()** оценивается при каждом выполнении оператора и вообще не оценивается, если оператор не выполняется. Оператор **require()** выполняется при первом синтаксическом анализе оператора, независимо от того, будет ли выполняться содержащий его бок кода.

Если только сервер не очень загружен, это не имеет особого значения, но это означает, что код, в котором операторы **require()** располагаются внутри условных операторов, неэффективен.

```
if($variable == true)
{
    require("file1.inc");
}
else
{
    require("file2.inc");
}
```

Этот код будет напрасно загружать оба файла при каждом выполнении сценария, но будет использовать только один из них, в зависимости от значения переменной **$variable**. Однако, если бы в коде использовались два оператора **include()**, только один из файлов загружался бы и выполнялся, как в следующей версии:

```
if($variable == true)
{
    include("file1.inc");
}
```

```
else
{
    include("file2.inc");
}
```

В отличие от файлов, загруженных при помощи оператора **require()**, файлы, которые загружены через оператор **include()**, могут возвращать значение. Следовательно, можно уведомлять другие части программы об успешном выполнении или ошибке во включенном файле, либо возвращать ответ или результат.

В случае, если файлы открываются очень часто, вместо того чтобы каждый раз повторно вводить одни и те же строки кода, можно использовать для этого включенный файл. Такой включенный файл мог бы называться **"openfile.inc" и** иметь следующий вид:

```
<?
@ $fp = fopen($name, $mode);
   if (!$fp)
   {
    echo "<p><strong> Oh No! I could not open the file.</strong></p>";
    return 0;
   }
   else
   {
     return 1;
   }
?>
```

Этот файл будет пытаться открыть файл **$name**, используя режим, определенный переменной **$mode**. В случае неудачи он будет выводить сообщение об ошибке и возвращать значение 0. В случае удачи он будет возвращать значение 1 без генерации какого-либо вывода.

Этот файл можно вызвать сценарии следующим образом:

```
$name = "file.txt";
$mode = "r";
$result = include("openfile.php");
if( $result == 1 )
{
   // выполнение необходимых действий с файлом
   // обращение к переменной $fp, созданной во включенном файле
}
```

Обратите внимание, что переменные можно создавать в основном файле, или во включенном или запрошенном файле, и переменная будет существовать в обоих файлах. В этом отношении и **require(), и include()** ведут себя одинаково.

Операторы **require()** нельзя использовать совершенно так же, как здесь описано, поскольку из них нельзя возвращать значения. Возвращаемое значение может быть полезным, поскольку позволяет уведомлять об ошибке выполняемые далее части программы или выполнять определенную независимую обработку и возвращать ответ. Функции — еще лучшее средство разбиения кода на независимые модули, нежели включенные файлы. Мы рассмотрим их в следующем разделе.

Может возникать вопрос: зачем использовать оператор **require()**, учитывая преимущества по сравнению с ним оператора **include()?** Ответ заключается в том, что оператор **require()** работает несколько быстрее.

Использование функций в PHP

Функции существуют во многих языках программирования. Они используются для выделения кода, который выполняет отдельную, четко определенную задачу. Это упрощает чтение кода и позволяет повторно его использовать при каждом выполнении задачи.

Под функцией понимают независимый модуль кода, который устанавливает интерфейс вызова, выполняет определенную задачу и, необязательно, возвращает результат.

Мы уже видели ряд функций. В предшествующих главах мы постоянно обращались к ряду функций, встроенных в PHP. Мы также создавали несколько простых функций, но не вникали при этом в детали. В этом разделе вызов и создание функций описываются далее подробно.

Вызов функций

Следующая строка — простейшее обращение к функции:

```
function_name();
```

Она вызывает функцию с именем **function_name**, которая не требует параметров. Эта строка кода игнорирует любое значение, которое может возвращаться указанной функцией.

Множество функций вызываются именно таким образом. Функция **phpinfo()** часто полезна во время тестирования, поскольку она отображает установленную версию PHP, информацию о PHP, параметры установки Web-сервера и значения различных переменных PHP и сервера. Эта функция не принимает никаких параметров, и в общем случае ее возвращаемое значение игнорируется; поэтому вызов **phpinfo()** будет иметь следующий вид:

```
phpinfo();
```

Большинство функций требуют передачи одного или более параметров — информации, передаваемой в функцию во время ее вызова и влияющей на результат ее выполнения. Параметры передаются путем помещения данных или имени переменной, содержащей данные, в круглые скобки вслед за именем функции. Обращение к функции с параметром имеет следующий вид:

```
function_name("parameter");
```

В этом случае использованный параметр — строка, содержащая только слово parameter, но следующие обращения также являются обращениями к функции:

```
function_name(2);
function_name(7.993);
function_name($variable);
```

В последней строке переменная **$variable** может быть PHP-переменной любого типа, в том числе и массивом.

Параметр может быть данным любого типа, но конкретные функции обычно требуют передачи совершенно конкретных типов данных.

Количество принимаемых функцией параметров, что представляет каждый из них и какой тип данных он должен иметь, можно выяснить из *прототипа* функции. При описании функции мы часто приводим ее прототип; полный набор прототипов функций из библиотеки PHP-функций можно найти по адресу http://www.php.net.

Вот, например, как выглядит прототип функции **fopen()**:

```
int fopen( string filename, string mode, [int use_include_path] );
```

Прототип сообщает ряд особенностей и важно уметь правильно интерпретировать эти спецификации. В данном случае слово int перед именем функции указывает, что эта функция будет возвращать целочисленное значение. Параметры функции заключаются в круглые скобки. В данном случае в прототипе указаны три параметра. Параметры *filename* (имя_файла) и *mode* (режим) являются строками, а *use_include_path* — целочисленным значением.

Квадратные скобки вокруг *use_include_path* показывают, что этот параметр необязателен. Для необязательных параметров можно передавать значения или же их можно игнорировать — в этом случае будет использоваться значение, определенное по умолчанию.

После ознакомления с прототипом этой функции понятно, что следующий фрагмент кода будет допустимым вызовом функции **fopen()**:

```
$name = "myfile.txt";
$openmode = "r";
$fp = fopen($name, $openmode)
```

Этот код вызывает функцию с именем fopen(). Возвращаемое функцией значение будет сохраняться в переменной **$fp**. В функцию передается переменная **$name**, содержащая строку, которая представляет открываемый файл, и переменная **$openmode**, содержащая строку, которая представляет требуемый режим открытия файла. Третий параметр не задан.

Обращение к неопределенной функции

При попытке вызвать функцию, которая не существует, отобразится сообщение об ошибке, как показано на рис. 5.3.

Как правило, выводимые PHP сообщения об ошибках очень полезны. Данное сообщение точно указывает файл и строку сценария, в которых произошла ошибка, и имя функции, попытка вызова которой была предпринята. Это должно существенно упростить поиск и исправление ошибки.

При отображении сообщения об ошибке необходимо проверить два момента:

1. Правильно ли указано имя функции?
2. Существует ли данная функция в используемой версии PHP?

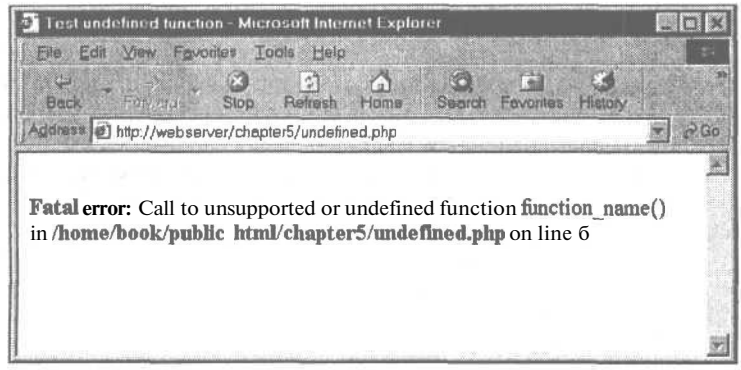


РИСУНОК 5.3
Это сообщение об ошибке — результат вызова функции, которая не существует.

Правильное написание имени функции не всегда легко запомнить. Например, некоторые функций, состоящие из двух слов, содержат символ подчеркивания между словами, некоторые — нет. Так, в имени функции **stripslashes()** два слова слиты в одно, в то время как в имени функции **strip_tags()** они разделены символом подчеркивания. Ввод имени функции с ошибкой в вызове функции приводит к ошибке (см. рис. 5.3).

Поскольку в книге предполагается, что применяется, по крайней мере, PHP 4.0, многие из использованных здесь функций отсутствуют в PHP 3.0. В каждой версии определяются новые функции, и если вы используете одну из предшествующих версий, дополнительные функциональные возможности и более высокая производительность оправдывают обновление программного обеспечения. Для выяснения того, когда была добавлена конкретная функция, можно обратиться к интерактивному руководству на сайте **www.php.net**. Попытка вызова функции, которая не объявлена в используемой версии, приведет к генерации сообщения об ошибке, подобного показанному на рис. 5.3.

Регистр и имена функций

Обратите внимание, что имена функций **не** зависят от регистра, поэтому все обращения **function_name()**, **Function_Name()** или **FUNCTION_NAME()** являются допустимыми и приведут к одному и тому же результату. Прописные буквы можно использовать в имени функции любым образом, по вашему мнению облегчающим чтение, но при этом следует стремиться к единообразию. В этой книге и в большинстве документации по PHP принято использовать строчные буквы.

Важно отметить, что имена функций ведут себя иначе, чем имена переменных. Имена переменных **зависят** от регистра, и поэтому **$Name** и **$name** — различные переменные, но **Name()** и **name()** — одна и та же функция.

В предыдущих главах приводилось много примеров использования некоторых встроенных функций PHP. Однако реальная сила языка программирования проявляется в возможности создания собственных функций.

Для чего нужно определять собственные функции?

Функции, встроенные в PHP, позволяют взаимодействовать с файлами, использовать базы данных, создавать графические изображения и подключаться к другим компьютерам. Однако во многих случаях придется выполнять что-либо, не предусмотренное разработчиками языка.

К счастью, действия программиста не ограничиваются встроенными функциями, поскольку для выполнения стоящей задачи можно создавать свои собственные функции. Вероятно, создаваемый код будет представлять собой комбинацию существующих функций и специализированной логики выполнения той или иной задачи. Группирование блока кода для выполнения задачи, который, скорее всего, придется вызывать в нескольких местах сценария или в нескольких сценариях, имеет смысл оформить этот блок в виде функции.

Объявление функции позволяет использовать созданный код так же, как встроенные функции. Достаточно просто вызвать функцию и передать в нее параметры. Это означает, что одну и ту же функцию можно многократно использовать в сценарии.

Базовая структура функции

Объявление функции создает, или *объявляет*, новую функцию. Объявление начинается с ключевого слова **function**, задает имя функции, требуемые параметры и содержит код, который будет выполняться при каждом вызове функции.

Вот объявление простой функции:

```
function my_function()
{
    echo "My function was called"
}
```

Это объявление функции начинается с ключевого слова **function**, которое сообщает человеку и программе синтаксического анализа PHP, что далее следует определяемая пользователем функция. Именем функции является **my_function**. Новую функцию можно вызвать с помощью следующего оператора:

```
my_function();
```

Как вы, вероятно, догадались, вызов этой функции приведет к отображению текста "**My function was called**" в окне браузера.

Встроенные функции доступны для всех PHP-сценариев, но при объявлении собственных функций они доступны только для того сценария (сценариев), в которых они были объявлены. Имеет смысл поддерживать отдельный файл, который содержит часто используемые функции. Тогда в каждый из своих сценариев можно поместить оператор **require()**, чтобы получить доступ к этим функциям.

Внутри функции код, который выполняет требуемую задачу, заключается в фигурные скобки. Между скобками можно помещать любые конструкции, допустимые где-либо в PHP-сценарии, в том числе вызовы функций, объявления новых переменных и функций, операторы **require()** или **include()** и обычный HTML-текст. Если внутри функции нужно выйти из среды PHP и ввести обычный HTML-текст, это делается так же, как в любом другом месте сценария — при помощи закрывающего PHP-дескриптора, за которым помещается HTML-текст. Ниже показан допустимый вариант приведенного примера, который генерирует такой же вывод:

```
<?
    function my_function()
    {
?>
My function was called
<?
    }
?>
```

Обратите внимание, что PHP-код заключен между соответствующими открывающим и закрывающим PHP-дескрипторами. В этой книге в большинстве примеров небольших фрагментов кода эти дескрипторы не показаны. Здесь же они показаны потому, что они обязательны внутри примера, а также выше и ниже него.

Соглашения об именовании функций

При выборе имен присвоения имен функциям наиболее важно учитывать, что имя должно быть коротким и описательным. Если функция создает верхний колонтитул страницы, подходящим именем может быть **pageheader()** или **page_header()**.

На имена функций накладываются следующие ограничения:

- Функция не может иметь то же имя, что у существующей функции.
- Имя функции может содержать только буквы, цифры и символы подчеркивания.
- Имя функции не может начинаться с цифры.

Многие языки программирования допускают повторное использование имен функций. Это свойство называется *перегрузкой функций*. Однако PHP не поддерживает перегрузку функций, поэтому функция не может иметь имя, совпадающее с именем любой встроенной или существующей определяемой пользователем функции. Имейте в виду, что хотя любой PHP-сценарий распознает все встроенные функции, определяемые пользователем функции существуют только в тех сценариях, в которых они объявлены. Это означает, что имя функции можно повторно использовать в другом файле, но это может приводить к путанице, поэтому подобных ситуаций следует избегать.

Следующие имена функций допустимы:

```
name()
name2()
name_three()()
_namefour()
```

А эти недопустимы:

```
5name()
name-six()
fopen()
```

(Последнее имя было бы допустимым, если бы оно уже не существовало.)

Параметры

Чтобы иметь возможность выполнять свои задачи, большинство функций требуют передачи в них одного и более параметров. Параметр позволяет передавать данные в функцию. Ниже приведен пример функции, которая требует передачи в нее параметра. Эта функция принимает одномерный массив и отображает его в виде таблицы.

```
function create_table($data)
{
   echo "<table border = 1>";
   reset($data); // Это используется для указания на начало
  $value = current($data);
  while ($value)
  {
       echo "<tr><td>$value</td></tr>\n";
     $value = next($data);
  }
   echo "</table>";
}
```

Если вызвать функцию **create_table()** следующим образом:

```
$my_array = array("Line one.","Line two.","Line three.");
create_table($my_array);
```

на экране отобразится вывод, показанный на рис. 5.4.

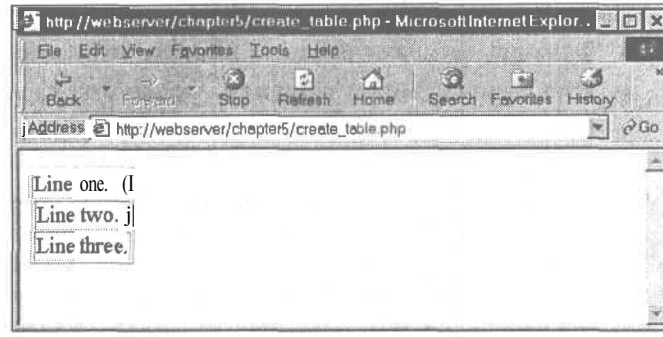


РИСУНОК 5.4
Эта HTML-таблица — результат вызова функции create_table().

Передача параметра позволяет передать в функцию данные, которые были созданы вне функции — в данном случае, массив **$data**.

Как и в случае встроенных функций, определяемые пользователем функции могут иметь несколько параметров и необязательные параметры. Функцию **create_table()** можно усовершенствовать многими способами, но одним из них могло бы быть предоставление пользователю возможности указывать рамку или другие атрибуты таблицы. Ниже приведена улучшенная версия функции. Она весьма подобна предыдущей, но позволяет при желании определять ширину рамки, расстояние между ячейками и способ заполнения ячеек.

```
function create_table2( $data, $border=1, $cellpadding=4,
   $cellspacing=4  )
{
    echo "<table border = $border cellpadding = $cellpadding"
          ." cellspacing = $cellspacing>";
   reset($data);
   $value = current($data);
   while ($value)
   {
        echo "<tr><td>$value</td></tr>\n";
      $value = next($data);
   }
    echo "</table>";
}
```

Первый параметр функции **create_table2()** по-прежнему обязателен. Следующие три — не обязательны, поскольку для них определены значения по умолчанию. Вывод, очень похожий на показанный на рис. 5.4, можно создать с помощью следующего вызова функции **create_table2().**

```
create_table2($my_array);
```

Если нужно, чтобы эти же данные отображались в более просторной форме, новую функцию можно было бы вызвать следующим образом:

```
create_table2($my_array, 3, 8, 8);
```

Необязательные параметры можно передавать не все — некоторые передать, а другие игнорировать. Параметры будут присваиваться слева направо.

Имейте в виду, что нельзя пропускать какой-либо необязательный параметр — требуется передать параметр, указанный в списке за ним. В данном примере, если требуется передать значение для **cellspacing,** придется передать также и значение для **cellpadding.** Это ограничение — причина распространенных ошибок программирования.

В связи с этим же необязательные параметры указываются в конце любого списка параметров.

Следующий вызов функции:

```
create_table2($my_array, 3);
```

вполне допустим и приведет к тому, что значение переменной **$border** будет установлено равным 3, а значения переменных **$cellpadding** и **$cellspacing** будут установлены равными своим значениям по умолчанию.

Область действия

Возможно, читатели обратили внимание, что при необходимости использования переменных внутри требуемого или включенного файла мы просто объявляем их в сценарии перед оператором **require()** или **include()**, но при использовании функции мы явно передаем эти переменные в функцию. Частично это связано с тем, что не существует никакого механизма для явной передачи переменных в требуемый или включенный файл, а частично с тем, что область действия переменных для функций определяется иначе.

Диапазон действия переменных управляет тем, где переменная видима и применима. В различных языках программирования действуют различные правила, устанавливающие диапазон действия переменных. В PHP действуют очень простые правила:

- Переменные, которые объявлены внутри функции, действуют в области от оператора, в котором они объявлены до закрывающей скобки в конце функции. Эта область называется *областью функции,* а такие переменные — *локальными переменными.*
- Переменные, которые объявлены вне функции, действуют в области от оператора, в котором они объявлены до конца файла, но **не внутри функций.** Эта область называется *глобальной областью,* а такие переменные — *глобальными переменными.*
- Использование операторов **require()** и **include()** не влияет на область действия переменных. Если оператор используется внутри функции, применяется область функции. Если он используется не внутри функции, применяется глобальная область.
- Ключевое слово **global** может использоваться для указания вручную того, что переменная, которая определена или используется внутри функции, будет иметь глобальную область действия.
- Переменные могут быть вручную удалены при помощи функции **unset(*$variable_name*)**. Если переменная удалена, она больше не находится в области действия.

Следующие примеры помогут разобраться с описанными концепциями.

Следующий код не создает никакого вывода. В нем объявляется переменная **$var** внутри функции **fn()**. Поскольку эта функция объявляется внутри функции, она имеет область действия функции и существует от места ее объявления до конца функции. При новом обращении к **$var** вне функции, создается новая переменная **$var.** Эта новая переменная имеет глобальную область действия и будет видима до конца файла. К сожалению, если единственный оператор, используемый с этой новой переменной **$var** — **echo,** она никогда не будет иметь значения.

```
function fn()
{
   $var = "contents";
}
echo $var;
```

Следующий пример противоположен предыдущему. Мы объявляем переменную снаружи функции, а затем пытаемся ее использовать внутри функции.

```
function fn()
{
   echo "inside the function, \$var = ".$var."<br>";
   $var = "contents2";
   echo "inside the function, \$var = ".$var."<br>";
}
$var = "contents 1";
fn();
echo "outside the function, \$var = ".$var."<br>";
```

Этот код создаст следующий вывод:

```
inside the function, $var =
inside the function, $var = contents 2
outside the function, $var = contents 1
```

Функции не выполняются до тех пор, пока они не будут вызваны, поэтому первым выполняемым оператором является **$var = "contents 1";** Он создает переменную **$var,** имеющую глобальную область действия и содержимое **"contents 1".** Следующий выполняемый оператор — обращение к функции **fn()**. Строки внутри оператора выполняются по очереди. Первая строка в функции обращается к переменной **$var**. Когда эта строка выполняется, она не может видеть предшествующую созданную нами переменную **$var**, поэтому она создает новую переменную, имеющую область функции, и повторяет ее в выводе. В результате создается первая строка вывода.

Следующая строка внутри функции устанавливает содержимое переменной **$var** равным **"contents** 2". Поскольку действия выполняются внутри функции, эта строка изменяет значение локальной переменной **$var**, а не глобальной. Вторая строка вывода подтверждает выполнение этого изменения.

На этом выполнение функции завершается, поэтому выполняется заключительная строка сценария. Этот оператор **echo** демонстрирует, что значение глобальной переменной не изменилось.

Если нужно, чтобы переменная, созданная внутри функции, была глобальной, можно использовать ключевое слово **global,** как показано в следующем примере:

```
function fn()
{
   global $var;
    $var = "contents";
    echo "inside the function, \$var = ".$var."<br>";
}
fn();
echo "outside the function, \$var = ".$var."<br>";
```

В этом примере переменная **$var** была явно объявлена как глобальная, т.е. после вызова функции переменная будет существовать и вне функции. Вывод этого сценария будет выглядеть следующим образом:

```
inside the function, $var = contents
outside the function, $var = contents
```

Обратите внимание, что переменная определена в области, начиная с того места, где выполняется строка **global $var;**. Функцию можно было бы объявить выше или ниже того места, где она вызывается. (Обратите внимание, что область действия функции значительно отличается от области действия переменной!). Место объявления функции не существенно — важно лишь то, где мы вызываем функцию и, следовательно, выполняем содержащийся внутри нее код.

Ключевое слово **global** можно использовать также в начале сценария при первом использовании переменной для объявления того, что весь сценарий должен быть областью ее действия. Вероятно, это — наиболее распространенное использование ключевого слова **global.**

Как видно из приведенных примеров, вполне допустимо повторно использовать имя переменной внутри и снаружи функции без взаимного влияния между ними. Однако, в общем случае делать это не рекомендуется, поскольку, не вникнув в код и не приняв во внимание область действия переменных, пользователи могут решить, что эти переменные являются одной и той же переменной.

Передача по ссылке и передача по значению

Если требуется создать функцию **increment()**, позволяющую увеличивать значение, программист может предпринять попытку создать ее следующим образом:

```
function increment($value, $amount = 1)
{
   $value = $value +$amount;
}
```

Этот код будет бесполезен. В результате выполнения следующего теста выводится значение "10".

```
$value = 10;
increment ($value);
echo $value;
```

Как видим, содержимое переменной **$value** не изменилось.

Это связано с правилами области действия. Этот код создает переменную **$value**, которая содержит значение 10. Затем программа вызывает функцию **increment()**. Переменная **$value** создается в функции при ее вызове. К значению добавляется 1, поэтому внутри функции значение **$value** равно 11 до тех пор, пока выполнение функции не завершается и не осуществляется возврат к вызвавшему ее коду. В этом коде переменной **$value** является другая переменная, определенная в глобальной области, и поэтому она остается неизменной.

Один из способов решения этой проблемы — объявление переменной **$value** в функции в качестве глобальной, но это означает, что для использования этой функции переменную, значение которой требуется увеличить, потребовалось бы назвать **$value**. Более рациональным подходом было бы использование *передачи по ссылке.*

Обычный способ вызова параметров функции называется *передачей по значению.* При передаче параметра создается новая переменная, которая содержит переданное значение. Она является копией исходной переменной. Это значение можно изменять любым образом, но при этом значение исходной переменной вне функции остается неизменным.

Более рациональный подход — использование *передачи по ссылке.* В этом случае при передаче параметра, вместо того чтобы создавать новое значение, функция принима-

ет ссылку на исходную переменную. Эта ссылка имеет имя переменной, начинающееся со знака доллара, и может использоваться совершенно так же, как любая другая переменная. Различие состоит в том, что вместо того, чтобы иметь собственное значение, она просто ссылается на исходную переменную. Любые изменения, применяемые к ссылке, влияют также и на оригинал.

Передача используемого параметра по ссылке указывается путем помещения символа амперсанда (&) перед его именем в определении функции. Никакие изменения в вызове функции не требуются.

Ранее приведенный пример функции **increment()** можно изменить, передав ей один параметр по ссылке, после чего функция будет работать правильно.

```
function increment(&$value, $amount = 1)
{
   $value = $value + $amount;
}
```

Теперь мы располагаем работающей функцией и можем назвать переменную, значение которой нужно увеличивать как угодно. Как уже упоминалось, использование одного и того же имени внутри и снаружи функции может привести к путанице, поэтому переменной в основном сценарии мы присвоим новое имя. Теперь следующий код будет выводить на экран значение 10 перед обращением к функции **increment()** и 11 — после него.

```
$a = 10;
echo $a
increment ($a);
echo $a;
```

Возврат из функций

Ключевое слово **return** прекращает выполнение функции. Когда выполнение функции завершается либо потому, что все операторы выполнены, либо по причине использования ключевого слова **return,** управление возвращается к оператору, следующему за вызовом функции.

При вызове следующей функции выполняется только первый оператор **echo.**

```
function test_return()
{
   echo "This statement will be executed";
  return;
   echo "This statement will never be executed";
}
```

Очевидно, это не очень полезный способ использования оператора **return.** Обычно возврат из функции будет требоваться только при выполнении определенного условия.

Условие возникновения ошибки — распространенная причина применения оператора **return** с целью преждевременного прекращения выполнения функции. Если, например, была создана функция для определения большего из двух чисел, выход из нее может требоваться в случае отсутствия любого из чисел.

```
function larger( $x, $y )
{
  if (!isset($x) || !isset($y))
  {
     echo "this function requires two numbers";
    return;
  }
```

```
    if    ($x>=$y)
        echo $x;
    else
        echo $y;
}
```

Встроенная функция isset() сообщает, была ли создана переменная и было ли ей присвоено значение. Этот код должен генерировать сообщение об ошибке и выполнять возврат, если любой из параметров не имеет установленного значения. Эта проверка выполняется с помощью выражения **!isset(),** означающего "НЕ isset()", и следовательно, оператор if можно прочесть как "если x не установлено или y не установлено". Функция будет выполнять возврат, если любое из этих условий истинно.

Если оператор **return** выполняется, последующие строки кода в функции будут игнорироваться. Выполнение программы вернется к точке, в которой функция была вызвана. Если оба параметра установлены, функция выведет на экран больший из них.

Выводом следующего кода:

```
$a = 1;
$b = 2.5;
$c = 1.9;
larger($a, $b);
larger($c, $a);
larger($d, $a);
```

будет следующий:

```
2.5
1.9
this function requires two numbers
```

Возврат значений из функций

Возврат из функции — не единственная причина использования оператора **return.** Во многих функциях операторы **return** используются для обмена данными с вызывающим их кодом. Функция была более полезной, если бы вместо вывода на экран результата сравнения в функции **larger()** она возвращала бы ответ. В этом случае вызвавший ее код мог бы принимать решение, нужно ли и когда именно отображать или использовать этот ответ. Эквивалентная встроенная функция max() действует именно таким образом.

Функцию **larger()** можно было бы определить следующим образом:

```
function larger ($x, $y)
{
  if (!isset($x)||!isset($y))
     return -1.7E+308;
   else if ($x>=$y)
      return $x;
  else
     return $y;
}
```

Эта функция возвращает большее из двух переданных ей значений. В случае ошибки функция будет явно возвращать другое число. Если одно из чисел отсутствует, можно ничего не возвращать или вернуть значение -1.7×10^{308}. Это число очень мало и его вряд ли можно спутать с реальным ответом. Встроенная функция **max()** ничего не возвращает, если обе переменные не установлены, а если только одна переменная была установлена, функция возвращает упомянутое значение.

ПРИМЕЧАНИЕ

Почему было выбрано число -1.7×10^{308}? Во многих языках определены минимальное и максимальное значения для чисел. К сожалению, в PHP такие предельные значения не определены. Число -1.7×10^{308} — минимальное число, поддерживаемое PHP 4.0, но если подобный тип поведения важен, следует иметь в виду, что нельзя гарантировать сохранение этого же предела в будущем. Поскольку в настоящее время величина предела определяется лежащим в основе PHP типом данных двойной точности C, теоретически она может быть различной в различных операционных системах или компиляторах.

Этот КОД:

```
$a = 1; $b = 2.5; $c = 1.9;
echo larger($a, $b)."<br>";
echo larger($c, $a)."<br>";
echo larger($d, $a)."<br>";
```

создаст следующий вывод:

```
2.5
1.9
-1.7E+308
```

Часто функции, которые выполняют определенную задачу, но не должны возвращать значение, возвращают значение **true** или **false** для указания на успешное или неуспешное выполнение. Значения **true и false** могут быть соответственно представлены значением 1 или 0.

Блоки кода

Группа операторов объявляется блоком путем помещения их в фигурные скобки. Это не влияет на действие большей части кода, но оказывает специфичное воздействие в том числе и на способ выполнения таких управляющих структур, как циклы и условные операторы.

Следующие два примера работают различным образом.

Пример без блока кода

```
for($i = 0; $i < 3; $i++ )
   echo "Line 1<br>";
echo "Line 2<br>";
```

Пример с блоком кода

```
for($i = 0; $i < 3; $i++ )
{
   echo "Line 1<br>";
   echo "Line 2<br>";
}
```

В обоих примерах цикл **for** повторяется три раза. В первом примере только единственная строка, расположенная непосредственно под этим кодом, выполняется в цикле **for.** Вывод, генерируемый этим примером, имеет следующий вид:

```
Line 1
Line 1
Line 1
Line 2
```

Во втором примере блок кода используется для группирования двух строк. Это означает, что обе строки выполняются в цикле **for** три раза. Вывод, генерируемый этим примером, имеет вид:

```
Line 1
Line 2
Line 1
Line 2
Line 1
Line 2
```

Поскольку код в этих примерах отображен с соответствующими отступами, вероятно, несложно с первого взгляда увидеть различие между ними. Отступы в коде служат для упрощения понимания того, на какие строки воздействует цикл **for.** Однако следует помнить, что пробелы не влияют на обработку кода в PHP.

В некоторых языках блоки кода влияют на область действия переменных, но в PHP это не так.

Рекурсия

PHP поддерживает рекурсивные функции. Под *рекурсивной функцией* понимают функцию, которая вызывает саму себя. Эти функции особенно полезны для перемещения по динамическим структурам данных, таким как связные списки и деревья.

Однако лишь немногие Web-приложения требуют столь сложных структур данных, поэтому рекурсия используется очень редко. Во многих случаях рекурсия может использоваться вместо итерации, поскольку оба эти подхода позволяют повторно выполнять те или иные действия. Рекурсивные функции работают медленнее и используют больший объем памяти, чем итерация, поэтому всегда по возможности следует отдавать предпочтение итерации.

Ради полноты изложения рассмотрим краткий пример, приведенный в листинге 5.5.

Листинг 5.5 recursion.php — эта программа предназначена просто для обращения строки с помощью использования рекурсии (итеративная версия также показана)

```
function reverse_r($str)
{
    if (strlen($str)>0)
      reverse_r(substr($str, 1));
    echo substr($str, 0, 1);
    return;
}

function reverse_i($str)
{
    for ($i=1; $i<=strlen($str); $i++)
    {
      echo substr($str, -$i, 1);
    }
    return;
}
```

В этом листинге реализованы две функции. Обе будут выводить строку в обратном порядке. Функция **reverse_r()** — рекурсивная, а **reverse_i()** — итеративная.

Функция **reverse_r()** принимает строку в качестве параметра. При ее вызове она будет вызывать саму себя, каждый раз передавая символы строки со второго до последнего.

Например, если вызвать функцию

```
reverse_r("Hello");
```

она вызовет себя несколько раз со следующими параметрами:

```
reverse_r("ello");
reverse_r("llo");
reverse_r("lo");
reverse_r("o");
reverse_r("");
```

При каждом обращении к самой себе функция создает новую копию кода функции в памяти сервера, но с другим параметром. Внешне это выглядит так, словно в действительности каждый раз вызывается другая функция. Это предотвращает возникновение путаницы с экземплярами функции.

При каждом вызове проверяется длина переданной строки. По достижении конца строки условие оказывается ложным (strlen()==0). После этого последний экземпляр функции (**reverse_r("")**) будет завершен и будет выполнена следующая строка кода, повторяющая на экране первый символ переданной строки — в данном случае никакой, поскольку строка пуста.

Затем этот экземпляр функции возвращает управление экземпляру, который вызвал его, а именно — **reverse_r("o")**. Этот экземпляр выводит первый символ в своей строке, "o", и возвращает управление вызвавшему его экземпляру.

Этот процесс — вывод символа и возврат к экземпляру функции, расположенному над ним в порядке вызова — продолжается до тех пор, пока управление не будет возвращено основной программе.

Рекурсивные решения весьма изящны и соответствуют математическим методам. Однако, в большинстве случаев лучше использовать итеративные решения. Код для реализации этого метода также приведен в листинге 5.5. Обратите внимание, что по длине он не превышает рекурсивную программу (хотя при использовании итеративных функций это и не всегда так) и обеспечивает совершенно такой же результат.

Основное различие между ними заключается в том, что рекурсивная функция будет создавать в памяти копии самой себя и, соответственно, приводит к накладным расходам, связанным с несколькими вызовами функции.

Рекурсивное решение имеет смысл использовать, когда соответствующий код оказывается гораздо короче и изящнее итеративной версии, но в данной области применения подобное случается нечасто.

Хотя рекурсия выглядит более элегантной, программисты часто забывают определить условие прекращения рекурсии. Это означает, что функция будет продолжать вызывать себя до тех пор, пока сервер не столкнется с нехваткой памяти или пока не истечет максимальное время выполнения, в зависимости от того, что произойдет раньше.

Дополнительная информация

Использование операторов **include()**, **require()**, function и return объясняется также в интерактивном руководстве. Более подробно о таких понятиях, как рекурсия, передача по значению/ссылке и область действия, применяющихся во многих языках программирования, можно прочесть во многих учебниках по программированию.

Что дальше

Теперь, когда читатели научились использовать включенные и требуемые файлы и функции для того, чтобы сделать свои программы более пригодными для внесения изменений и повторного использования, в следующей главе рассматривается объектно-ориентированное программирование и его поддержка в PHP. Использование объектов позволяет достичь целей, подобных концепциям, представленным в этой главе, но при этом достигаются еще большие преимущества во время разработки сложных проектов.

Глава 6

Объектно-ориентированное программирование на PHP

В этой главе поясняются концепции объектно-ориентированной разработки и способы их применения PHP.

В главе освещаются следующие основные темы

- Концепции объектно-ориентированного программирования
- Создание классов, атрибутов и операций
- Использование атрибутов класса
- Вызов операций класса
- Наследование
- Вызов методов класса
- Разработка классов
- Написание кода класса

Концепции объектно-ориентированного программирования

Современные языки программирования обычно поддерживают или даже требуют применения объектно-ориентированного подхода к разработке программного обеспечения. В процессе объектно-ориентированной (ОО) разработки предпринимается попытка использования классификаций, отношений и свойств объектов системы для упрощения разработки программ.

Классы и объекты

В контексте объектно-ориентированного программного обеспечения объектом может быть практически любой элемент или сущность — такой физический объект, как рабочий стол или клиент, или такой концептуальный объект, существующий только в программе, как область ввода текста или файл. В общем случае наибольший интерес представляют концептуальные объекты, в том числе реальные объекты, которые должны быть представлены в программе.

Объектно-ориентированное программное обеспечение разрабатывается и создается в виде набора самостоятельных объектов, имеющих атрибуты и операции, которые взаимодействуют с целью удовлетворения определенных нужд. Под *атрибутами* понимают свойства или переменные, имеющие отношение к объекту. Под *операциями* понимают методы, действия или функции, которые объект может выполнять для изменения самого себя или какого-либо внешнего объекта.

Основное преимущество объектно-ориентированного программного обеспечения заключается в его способности поддерживать и стимулировать *инкапсуляцию,* называемую также сокрытием данных. По существу, доступ к данным внутри объекта возможен только через операции объекта, называемые *интерфейсом* объекта.

Действия, выполняемые объектом, распространяются только на используемые им данные. Можно легко изменять особенности реализации объекта для повышения производительности, добавления новых свойств или исправления программных ошибок *без необходимости изменять интерфейс,* что могло бы оказать влияние на целый проект.

В других областях разработки программного обеспечения объектно-ориентированный подход является нормой, а функционально-ориентированное программное обеспечение считается устаревшим. К сожалению, по ряду причин большинство сценариев для Web все еще разрабатываются и создаются с применением *специализированного* подхода, следующего функционально-ориентированной методологии.

Это связано с рядом причин. Основное множество Web-проектов сравнительно невелики и просты. Можно просто вооружиться пилой и соорудить деревянную полочку для специй, не планируя заранее своих действий; точно так же можно выполнить основное множество проектов по разработке программного обеспечения для Web, поскольку они малы. Однако вряд ли удастся добиться приемлемых результатов, если пытаться строить дом с помощью одной только пилы, не имея формального плана – это же справедливо и по отношению к большим программным проектам.

Многие проекты для Web начинаются с набора страниц, связанных гиперссылками, и завершаются сложными приложениями. Эти сложные приложения, представлены ли они посредством диалоговых окон или динамически генерируемых HTML-страниц, нуждаются в тщательно продуманной методологии разработки. Ориентация на объекты может способствовать уменьшению сложности проектов, повышению пригодности кода для повторного использования и, тем самым, снижению затрат на поддержку.

В объектно-ориентированном программном обеспечении объект — это уникальная и идентифицируемая коллекция сохраненных данных и операций, манипулирующих данными. Например, могли бы существовать два объекта, которые представляют кнопки. Даже если обе кнопки обозначены "OK", имеют ширину 60 пикселей и высоту 20 пикселей и у них совпадают любые другие атрибуты, все же нужно иметь возможность работать с той или иной кнопкой. В программе для этого используются отдельные переменные, которые действуют как *дескрипторы* (уникальные идентификаторы) объектов.

Объекты могут группироваться в классы. Классы представляют наборы объектов, которые могут отличаться один от другого, но должны иметь нечто общее. Класс содержит объекты, в которых одинаковые операции действуют одинаково, а одинаковые атрибуты представляют те же самые свойства, хотя количество этих атрибутов будет изменяться от объекта к объекту.

Существительное "велосипед" можно было бы считать классом объектов, описывающим множество различных велосипедов, имеющих много общих характеристик, или *атрибутов* — таких как наличие двух колес, цвет и размер, а также выполняемые ими операции, например, передвижение.

Личный велосипед можно считать объектом, относящимся к классу велосипедов. Он имеет все характеристики, присущие всем велосипедам, включая операцию передвижения, которая выполняется так же, как у большинства остальных велосипедов — даже если данный велосипед используется реже. Атрибуты данного велосипеда имеют уникальные атрибуты, поскольку он имеет, скажем, зеленый цвет, а не все велосипеды — зеленые.

Полиморфизм

Объектно-ориентированный язык программирования должен поддерживать *полиморфизм* — т.е. одна и та же операция может выполняться в различных классах поразному. Например, при наличии класса автомобилей и класса велосипедов операции передвижения в них различны. Применительно к реальным объектам это редко будет составлять проблему. Вряд ли можно спутать велосипеды с автомобилями и начать использовать операцию передвижения автомобиля вместо операции передвижения велосипеда. Однако понятие обычного здравого смысла неприменимо к языку программирования, поэтому язык должен поддерживать полиморфизм, чтобы знать, какую операцию передвижения нужно использовать применительно к конкретному объекту.

Полиморфизм — это скорее характеристика поведения, нежели объектов. В PHP только функции-члены класса могут быть полиморфными. Функции-члены можно сравнить с глаголами обычного языка. В реальном мире велосипед можно чистить, передвигаться на нем, разбирать, ремонтировать или красить.

Эти глаголы описывают общие действия, поскольку неизвестно, к какого рода объекту они применяются. (Такого рода абстракция объектов и действий — одна из отличительных характеристик человеческого разума.)

Например, для езды на велосипеде требуется выполнение совершено иных действий, чем для езды в автомобиле, несмотря на то что эти понятия аналогичны. Глагол *ездить* может быть связан с конкретным набором действий только после того, как известен объект, к которому он применяется.

Наследование

Наследование позволяет с помощью *подклассов* создавать иерархические взаимосвязи между классами. Подкласс наследует атрибуты и операции своего *суперкласса.* Например, автомобиль и велосипед имеют некоторые общие характеристики. Можно было бы создать класс транспортных средств, содержащий такие характеристики, как атрибут цвета и операцию перевозки, присущие всем транспортным средствам, которые бы затем наследовались у него классами автомобилей и велосипедов.

Используя наследование, можно создавать надстройки и дополнения существующих классов. По мере необходимости из простого базового класса можно получать более сложные и специализированные производные классы. Это делает код более при-

годным для повторного использования, что является одним из важных преимуществ объектно-ориентированного подхода.

Использование наследования позволяет уменьшить объем выполняемой работы, если операции могут создаваться лишь один раз в суперклассе вместо многократного их создания в отдельных подклассах. Оно позволяет также точнее моделировать взаимосвязи реального мира. Если применительно к двум классам выражение "является" имеет смысл, вероятно, наследование вполне подходит в этом случае. Выражение "автомобиль является транспортным средством" имеет смысл, но выражение "транспортное средство является автомобилем" лишено его, поскольку не все транспортные средства являются автомобилями. Следовательно, автомобиль наследует характеристики транспортного средства.

Создание классов, атрибутов и операций в PHP

До сих пор мы рассматривали классы достаточно абстрактно. При создании класса в PHP используется ключевое слово **class.**

Структура класса

Минимальное объявление класса выглядит следующим образом:

```
class classname
{
}
```

Чтобы быть полезными, классы должны иметь атрибуты и операции. Атрибуты создаются путем объявления переменных внутри определения класса с помощью ключевого слова **var**. Следующий код создает класс **classname** с двумя атрибутами: **$attribute1** и **$attribute2**.

```
class classname
{
   var $attribute1;
   var $attribute2;
}
```

Операции создаются путем объявления функций внутри определения класса. Следующий код создает класс **classname** с двумя операциями, которые не выполняют никаких действий. Операция **operation1()** не принимает никаких параметров, а операция **operation2()** принимает два параметра.

```
class classname
{
   function operation1 ()
   {
   }
   function operation2($param1, $param2)
   {
   }
}
```

Конструкторы

Большинство классов будет иметь специальный тип операции, называемый конструктором. *Конструктор* вызывается при создании объекта и обычно выполняет такие полезные задачи по инициализации, как установка приемлемых начальных значений атрибутов или создание других объектов, требуемых для данного объекта.

Конструктор объявляется так же, как другие операции, но он имеет такое же имя, как у класса. Хотя конструктор можно вызывать вручную, в основном, он предназначается для автоматического вызова во время создания объекта. Следующий код объявляет класс с конструктором:

```
class classname
{
   function classname ($param)
   {
         echo "Constructor called with parameter $param <br>;
   }
}
```

Следует помнить, что PHP **не** поддерживает перегрузку функций — т.е. можно определять только одну функцию с любым конкретным именем; в этом отношении конструктор не является исключением. (Это свойство поддерживается во многих объектно-ориентированных языках.)

Создание экземпляров класса

После объявления класса необходимо создать объект — конкретный отдельный элемент, являющийся членом класса, с которым будет выполняться работа. Этот процесс называют также созданием экземпляров класса. Объект создается с помощью ключевого слова **new.** При этом нужно указать, экземпляром какого класса будет объект, и предоставить все параметры, которые требуются для конструктора.

Следующий код объявляет класс **classname** с конструктором, а затем создает три объекта типа **classname:**

```
class classname
{
   function classname ($param)
   {
      echo "Constructor called with parameter $param <br>";
   }
}
$a = new classname ("First");
$b = new classname ("Second");
$c = new classname ();
```

Поскольку конструктор вызывается при каждом создании объекта, этот код создает следующий вывод:

```
Constructor called with parameter First
Constructor called with parameter Second
Constructor called with parameter
```

Использование атрибутов класса

Внутри класса имеется доступ к специальному указателю с именем **$this**. Если атрибут текущего класса имеет имя **$attribute**, ссылка на него при установке или при обращении к переменной из операции внутри класса выполняется в виде **$this->attribute**.

Следующий код — пример установки и обращения к атрибуту внутри класса:

```
class classname
{
   var $attribute;
   function operation ($param)
   {
      $this->attribute = $param
      echo $this->attribute;
   }
}
```

Некоторые языки программирования позволяют ограничивать доступ к атрибутам за счет объявления таких данных приватными или защищенными. Это свойство в PHP не поддерживается, поэтому все атрибуты и операции видны вне класса (т.е. они являются общедоступными).

Эту же задачу можно выполнить за пределами класса, используя несколько иной синтаксис:

```
class classname
{
   var $attribute;
}
$a = new classname();
$a->attribute = "value";
echo $a->attribute;
```

Непосредственное обращение к атрибутам вне класса — не очень хорошая идея. Одно из преимуществ объектно-ориентированного подхода в том, что он поощряет инкапсуляцию. Хотя в PHP нельзя **принудительно** требовать сокрытие данных, при некотором желании можно достичь такого же результата.

Если вместо того чтобы осуществлять доступ к атрибутам класса непосредственно, создать *функции доступа*, доступ можно будет осуществлять через один раздел кода. При первоначальном создании функций доступа они могут выглядеть следующим образом:

```
class classname
{
   var $attribute;
   function get_attribute()
   {
      return $this->attribute;
   }
   function set_attribute($new_value)
   {
      $this->attribute = $new_value;
   }
}
```

Этот код просто предоставляет функции для доступа к атрибуту **$attribute**. В нем определена функция **get_attribute()**, которая просто возвращает значение атрибута **$attribute**, и функция **set_attribute()**, которая присваивает новое значение атрибуту **$attribute**.

На первый взгляд этот код может показаться не особо ценным. Возможно, что в том виде, в каком он приведен, это и так, но причина использования функций доступа проста: в этом случае обращение к конкретному атрибуту будет выполняться в рамках только одного раздела кода.

При наличии только одной точки обращения можно реализовать проверку того, что сохраняться будут только имеющие смысл данные. Если впоследствии выяснится, что значение атрибута **$attribute** может лежать в диапазоне между 0 и 100, можно только **один раз** добавить несколько строк кода и выполнять проверку перед тем, как разрешать изменения. Функцию **set_attribute()** можно изменить следующим образом:

```
function set_attribute($new_value)
{
   if( $new_value >= 0 && $newvalue <= 100 )
      $this->attribute = $new_value;
}
```

Это изменение тривиально, но если бы мы не использовали функцию доступа, пришлось бы просматривать каждую строку кода и изменять каждое обращение к атрибуту **$attribute** — трудоемкая и чреватая ошибками задача.

При наличии единственной точки доступа можно легко изменять реализацию, лежащую в основе кода. Если по какой-либо причине требуется изменить способ хранения атрибута **$attribute**, функции доступа позволяют делать это, изменяя код только в одном месте.

Может выясниться, что вместо хранения **$attribute** в виде переменной необходимо получать ее из базы данных, вычислять текущее значение при каждом ее запросе, получать значение, исходя из значений других атрибутов, или кодировать данные в виде более короткого типа данных. Какое бы изменение ни требовалось выполнить, можно просто изменить функции доступа. Это не окажет влияния на другие разделы кода до тех пор, пока функции доступа будут продолжать принимать или возвращать данные, ожидаемые другими частями программы.

Вызов операций класса

Операции класса можно вызывать во многом подобно вызову атрибутов класса. Предположим, что имеется следующий класс:

```
class classname
{
   function operation1()
   {
   }
   function operation2($param1, $param2)
   {
   }
}
```

и мы создаем объект типа **classname** с именем $a следующим образом:

```
$a = new classname()
```

Тогда операции можно вызывать так же, как вызываются другие функции: используя их имя и помещая нужные параметры в круглые скобки. Поскольку эти операции принадлежат объекту, а не обычным функциям, необходимо указать объект, которому они принадлежат. Имя объекта используется так же, как атрибуты объекта:

```
$a->operation1();
$a->operation2(12, "test");
```

Если операции возвращают что-либо, данные возврата можно получить следующим образом:

```
$x = $a->operation1();
$y = $a->operation2(12, "test");
```

Реализация наследования в PHP

Если класс должен быть подклассом другого класса, для указания этого можно воспользоваться ключевым словом **extends.** Следующий код создает класс B, унаследованный от некоторого ранее определенного класса A.

```
class B extends A
{
   var $attribute2;
   function operation2()
   {
   }
}
```

Если бы класс A был объявлен следующим образом:

```
class A
{
   var $attribute1;
   function operation1()
   {
   }
}
```

все следующие обращения к операциям и атрибутам объекта типа B были бы допустимыми:

```
$b = new B();
$b->operation1();
$b->attribute1 = 10;
$b->operation2();
$b->attribute2 = 10;
```

Обратите внимание, что поскольку класс B является расширением класса A, к операции **operation1()** и атрибуту **$attribute** можно обращаться, хотя они и были объявлены в классе A. Будучи подклассом класса A, класс B обладает всеми его функциональными возможностями и данными. Кроме того, в классе B объявляются его собственные атрибут и операция.

Важно отметить, что наследование работает только в одном направлении. Подкласс, или дочерний класс, наследует характеристики своего родительского класса, или суперкласса, но родительский класс не получает характеристики своего дочернего класса. Это означает, что две последних строки в следующем коде неверны:

```
$a = new A();
$a->operation1();
$a->attribute1 = 10;
$a->operation2();
$a->attribute2 = 10;
```

Класс A не имеет операции **operation2** или атрибута **attribute2**.

Перекрытие

Мы рассмотрели подкласс, в котором объявляются новые атрибуты и операции. Допустимо и иногда полезно повторно объявлять некоторые атрибуты и операции. Это может потребоваться для присвоения атрибуту в подклассе значения, которое отлича-

ется от значения по умолчанию этого же атрибута в суперклассе, или для придания операции в подклассе иных функциональных возможностей, нежели у той же операции в суперклассе. Упомянутый процесс носит название *перекрытия*.

Например, при наличии класса A:

```
class A
{
   var $attribute = "default value";
   function operation()
   {
      echo "Something<br>";
      echo "The value of \$attribute is $this->attribute<br>";
   }
}
```

и необходимости изменить значение по умолчанию атрибута **$attribute** и придать новые функциональные возможности операции **operation()**, можно создать следующий класс B, в котором выполняется перекрытие **$attribute и operation():**

```
class B extends A
{
   var $attribute = "different value";
   function operation()
   {
      echo "Something else<br>";
      echo "The value of \$attribute is $this->attribute<br>";
   }
}
```

Объявление класса B не влияет на исходное определение класса **A**. Давайте рассмотрим следующие две строки кода:

```
$a = new A();
$a -> operation();
```

Мы создали объект типа A и обратились к его функции **operation()**. В результате это даст вывод

```
Some thing
The value of $attribute is default value
```

при условии, что класс B не изменил класс A. Если создать объект типа B, получим другой вывод.

Код

```
$b = new B();
$b -> operation();
```

создает вывод

```
Something else
The value of $attribute is different value
```

Точно так же, как объявление новых атрибутов или операций в подклассе не оказывает влияния на суперкласс, перекрытие атрибутов или операций в подклассе также не влияет на суперкласс.

Подкласс будет наследовать все атрибуты и операции своего суперкласса, если только не будут определены замены для них. При вводе замещающего определения, оно получает приоритет и перекрывает исходное определение.

В отличие от некоторых других объектно-ориентированных языков, PHP не позволяет выполнять перекрытие функций, сохраняя возможность вызова версии, определенной в родительском классе.

Наследование может быть многоуровневым. Можно объявить класс C, который расширяет класс B и, следовательно, наследует характеристики из класса B и из родительского класса A. В классе C можно также выборочно перекрывать и замещать атрибуты и операции, унаследованные от родительских классов.

Множественное наследование

Некоторые объектно-ориентированные языки поддерживают множественное наследование, но PHP не делает этого. Это означает, что каждый класс может наследовать характеристики только от одного родительского класса. Количество дочерних классов, имеющих общий родительский класс, не ограничено.

Значение сказанного может быть понятно не сразу. На рис. 6.1 показаны три возможных способа наследования для трех классов A, B и C.

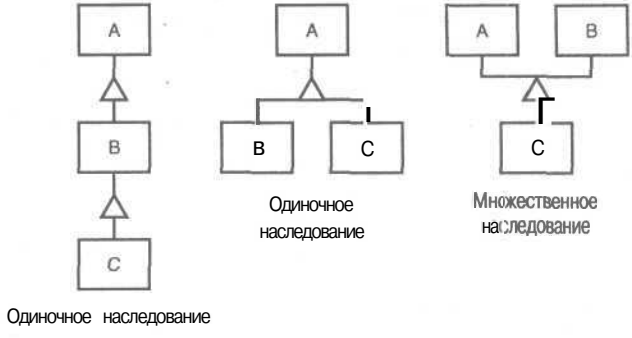


РИСУНОК 6.1
PHP не поддерживает множественное наследование.

На рисунке *слева* показано, что класс C унаследован от класса B, который, в свою очередь, унаследован от класса A. Каждый класс имеет не более одного родительского класса, следовательно, это — совершенно допустимое в PHP одиночное наследование.

На рисунке в *центре* показано, что классы B и C унаследованы от класса A. Каждый класс имеет не более одного родительского класса, следовательно, и это — допустимое одиночное наследование.

Справа показано, что класс C унаследован от обоих классов A и B. В этом случае класс C имеет два родительских класса, следовательно, это — множественное наследование и оно в PHP недопустимо.

Разработка классов

Теперь, когда читатели познакомились с концепциями, лежащими в основе объектов и классов, и с синтаксисом их реализации в PHP, пора рассмотреть способы разработки полезных классов.

Многие классы в программе будут представлять классы или сущности реального мира. Классы, которые, возможно, будут использоваться при разработке Web-приложений, могут включать страницы, компоненты пользовательского интерфейса, покупательскую тележку, обработку ошибок, категории товаров или клиентов.

Объекты в программе могут также представлять специфичные экземпляры ранее упомянутых классов — например, начальную страницу, конкретную кнопку или покупательскую тележку, находящуюся в пользовании у Фреда Смита в определенный промежуток времени. Сам Фред Смит может быть представлен объектом типа клиент. Каждый товар, приобретаемый Фредом, может представляться объектом, принадлежащим к определенной категории или классу.

В предыдущей главе простые включенные файлы использовались для придания единообразного вида различным страницам Web-сайта вымышленной компании TLA Consulting. Используя классы и воспользовавшись возможностью сэкономить время, которую обеспечивает наследование, можно создать более совершенную версию этого же сайта.

Нам требуется иметь возможность быстро создавать страницы для сайта TLA, которые выглядят и ведут себя одинаково. В то же время страницы должны быть пригодными для модификации, чтобы соответствовать различным частям сайта.

Нам предстоит создать класс **Page.** Основное назначение этого класса — ограничить объем HTML-кода, требуемого для создания новой страницы. Он должен допускать изменение тех частей, которые изменяются от страницы к странице, в то же время автоматически генерируя элементы, остающиеся неизменными.

Класс должен предоставлять гибкую структуру для создания новый страниц, но при этом не должен ограничивать свободу действий.

Поскольку страница генерируется из сценария, а не с помощью статического HTML, можно добавить любое количество полезных элементов, в том числе новые функциональные возможности, которые делают возможным следующее:

- Изменение элементов страницы только в одном месте. Например, при изменении примечания относительно авторских прав и добавлении дополнительной кнопки модификацию следовало бы вносить только в одном месте.
- Существование содержимого по умолчанию для большинства частей страницы при наличии возможности изменения любого элемента, когда это требуется, устанавливая пользовательские значения для таких элементов, как заголовок и метадескрипторы.
- Распознавание, какая страница будет просматриваться, и соответствующее изменение навигационных элементов — например, наличие на начальной странице кнопки, которая выполняет переход на начальную страницу, не имеет смысла.
- Позволяет замещать стандартные элементы для конкретных страниц. Например, если в определенных разделах сайта требуются другие навигационные кнопки, необходимо иметь возможность замещать стандартные кнопки.

Написание кода класса

После определения, как должен выглядеть вывод программы и какие новые функции она должна выполнять, как же это все реализовать?

Далее в этой книге рассматривается разработка и управление большими проектами. Пока давайте сосредоточим внимание на действиях, характерных для создания объектно-ориентированного PHP-кода.

Классу потребуется логическое имя. Поскольку он представляет страницу, назовем его **Page.** Для объявления класса **Page** следует ввести

```
class Page
{
}
```

Классу требуются некоторые атрибуты. Элементы, которые, возможно, придется изменять от страницы к странице, мы определим в качестве атрибутов класса. Основное содержимое страницы, которое будет комбинацией HTML-дескрипторов и текста, назовем **$content**. Содержимое можно объявить при помощи следующей строки кода внутри определения класса:

```
var $content;
```

Можно также определить атрибуты для хранения заголовка страницы. Вероятно, их придется изменять, чтобы посетитель получал четкое представление о просматриваемой странице. Вместо использования пустых заголовков, с помощью следующего объявления определим заголовок по умолчанию:

```
var $title = "TLA Consulting Pty Ltd";
```

Большинство коммерческих Web-страниц включают в себя метадескрипторы, помогающие поисковым механизмам выполнять их индексацию. Чтобы они были полезны, метадескрипторы, вероятно, должны изменяться от страницы к странице. В этом случае мы также определяем значение по умолчанию:

```
var $keywords = "TLA Consulting, Three Letter Abbreviation,
                 some of my best friends are search engines";
```

Навигационные кнопки, показанные на исходной странице на рис. 5.2 (см. предыдущую главу), скорее всего, должны оставаться неизменными на всех страницах, чтобы посетители не путались; однако, для их простого изменения, они также делаются атрибутами. Поскольку количество кнопок может изменяться, мы объявляем массив и сохраняем в нем и текст кнопки, и Web-адрес, на который она должна указывать.

```
var $buttons = array( "Home"     => "home.php",
                      "Contact"  => "contact.php",
                      "Services" => "services.php",
                      "site Map" => "map.php"
                    );
```

Чтобы выполнять определенные функциональные действия, классу потребуются также операции. Их определение можно начать с определения функций доступа для установки и получения значений определенных нами атрибутов. Они все принимают форму наподобие следующей:

```
function SetContent($newcontent)
{
   $this->content = $newcontent;
}
```

Поскольку маловероятно, что любые из этих значений будут запрашиваться извне класса, мы сознательно решили не определять соответствующий набор функций **GET.**

Основное назначение этого класса — отображение HTML-страницы, а для этого требуется функция. Она получает имя **Display()** и выглядит следующим образом:

```
function Display()
{
  echo "<html>\n<head>\n";
  $this -> DisplayTitle();
  $this -> DisplayKeywords();
  $this -> DisplayStyles();
  echo "</head>\n<body>\n";
  $this -> DisplayHeader();
```

```
        $this -> DisplayMenu($this->buttons);
        echo $this->content;
        $this -> DisplayFooter();
        echo "</body>\n</html>\n";
    }
```

Функция включает в себя несколько простых операторов **echo** для отображения HTML-текста, но в основном состоит из вызовов других функций класса. Как, вероятно, легко догадаться из их имен, эти функции отображают части страницы.

Вовсе не обязательно разбивать функции таким образом. Все эти отдельные функции можно было бы объединить в одну большую функцию. Мы же разделили их по ряду причин.

Каждая функция должна выполнять определенную задачу. Чем проще эта задача, тем проще создавать и тестировать функцию. Но не следует слишком увлекаться этим – если разбить программу на слишком большое количество небольших фрагментов, ее чтение может быть затруднено.

Используя наследование, можно выполнять перекрытие операций. Можно заместить одну большую функцию **Display()**, но вряд ли потребуется изменять способ отображения всей станицы. Гораздо рациональнее разбить действия по отображению на несколько самостоятельных задач и иметь возможность выполнять перекрытие только тех частей, которые требуется изменять.

Функция **Display** вызывает функции **DisplayTitle(), DisplayKeywords(), DisplayStyles(), DisplayHeader(), DisplayNenu() и DisplayFooter()**. Это означает, что нужно определить эти операции. Одно из усовершенствований PHP 4 по сравнению с PHP 3 — возможность записи операций или функций в этом логическом порядке, вызывая операцию или функцию прежде фактического кода функции. В PHP 3 и многих других языках функция или операция должна быть записана прежде, чем ее можно будет вызвать.

Большинство используемых в данном случае операций весьма просты и необходимы для отображения некоторого HTML-текста и, возможно, содержимого атрибутов.

Полный класс, который мы сохранили в виде файла **page.inc** для включения или затребования в других файлах, приводится в листинге 6.1.

Листинг 6.1 page.inc — класс Page обеспечивает простой гибкий способ создания страниц сайта TLA

```
<?
class Page
{
  // атрибуты класса Page
  var $content;
  var $title = "TLA Consulting Pty Ltd";
  var $keywords = "TLA Consulting, Three Letter Abbreviation,
                   some of my best friends are search engines";

  var $buttons = array( "Home"     => "home.php",
                        "Contact"  => "contact.php",
                        "Services" => "services.php",
                        "Site Map" => "map.php"
                      );
  // операции класса Page
  function SetContent($newcontent)
  {
    $this->content = $newcontent;
  }
  function SetTitle($newtitle)
  {
    $this->title = $newtitle;
  }
```

```
   function SetKeywords ( $newkeywords )
   {
     $this->keywords = $newkeywords;
   (
   function SetButtons ($newbuttons)
    {
     $this->buttons = $newbuttons;
   >
   function Display ()
    {
     echo "<html>\n<head>\n";
     $this -> DisplayTitle () ;
     $this -> DisplayKeywords () ;
     $this -> DisplayStyles () ;
     echo "</head>\n<body>\n";
     $this -> DisplayHeader();
     $this -> DisplayMenu($this->buttons) ;
     echo $this->content;
     $this -> DisplayFooter() ;
     echo "</body>\n</html>\n";
   }
   function DisplayTitle ()
   <
     echo "<title> $this->title </title>";
   >
   function DisplayKeywords()
   <
     echo "<META name=\"keywords\" content=\"$this->keywords\">";
   }

   function DisplayStyles ()
   {
?>
   <style>
     h1 {color:white; font-size:24pt; text-align:center;
         font-family:arial,sans-serif}
      .menu{color:white; font-size:12pt; text-align: center;
           font-family:arial,sans-serif;font-weight:bold}
     td {background:black}
     p {color:black; font-size:12pt; text-align:justify;
        font-family:arial,sans-serif}
     p.foot{color:white; font-size:9pt;text-align: center,'
           font-family:arial,sans-serif;font-weight:bold}
     a:link,a:visited,a:active{color:white}
   </style>
<?
   }

   function DisplayHeader()
   {
?>
   <table width="100%" cellpadding = 12 cellspacing =0 border = 0>
   <tr bgcolor = black>
     <td align = left><img src = "logo.gif"></td>
     <td>
         <h1>TLA Consulting Pty Ltd</h1>
     </td>
     <td align = right><img src = "logo.gif"></td>
   </tr>
   </table>
<?
   }
   function DisplayMenu ($buttons)
    {
     echo "<table width = \"100%\" bgcolor = white"
          ." cellpadding = 4 cellspacing = 4>\n";
     echo "  <tr>\n";

     //вычисление размеров кнопки
     $width = 100/count($buttons) ;
```

```
    while (list($name, $url) = each($buttons))
    {
      $this -> DisplayButton($width, $name, $url,
                             !$this->IsURLCurrentPage($url));
    }
    echo "  </tr>\n";
    echo "</table>\n";
  }
  function IsURLCurrentPage($url)
  {
    if(strpos( $GLOBALS["SCRIPT_NAME"], $url) ==false)
    {
      return false;
    }
    else
    {
      return true;
    }
  }
  function DisplayButton($width, $name, $url, $active = true)
  {
    if ($active)
    {
      echo "<td width = \"$width%\">
              <a href = \"$url\">
              <img src = \"s-logo.gif\" alt = \"$name\" border = OX/a>
              <a href = \"$url\"><span class=menu>$name</span></a></td>";
    }
    else
    {
      echo "<td width = \"$width%\">
              <img src = \"side-logo.gif\">
              <span class=menu>$name</span></td>";
    }
  }
  function DisplayFooter()
  {
?>
    <table width = "100%" bgcolor = black cellpadding = 12 border = 0>
    <tr>
      <td>
        <p class=foot>&copy; TLA Consulting Pty Ltd.</p>
        <p class=foot>Please see our
                      <a href ="">legal information page</a></p>
      </td>
    </tr>
    </table>
<?
  }
}
?>
```

При просмотре этого листинга обратите внимание, что функции **DisplayStyles()**, **DisplayHeader()** и **DisplayFooter()** должны отображать большой блок статического HTML-кода без обработки какого-либо PHP-кода. Поэтому внутри функций мы просто воспользовались завершающим PHP-дескриптором (?>), ввели HTML-код, а затем с помощью открывающего PHP-дескриптора (<?) вновь перешли к PHP.

В этом классе определены еще две операции. Операция **DisplayButton()** выводит единственную кнопку меню. Если кнопка указывает на текущую страницу, вместо нее отображается неактивная кнопка, которая выглядит несколько иначе и не связана ни с какими другими страницами. Это обеспечивает единообразный внешний вид страниц и позволяет посетителям визуально определять местоположение.

Операция **IsURLCurrentPage()** определяет, указывает ли связанный с кнопкой Internet-адрес на текущую страницу. Для этого применяется множество технологий. Мы

воспользовались строковой функцией **strpos()** для определения того, содержится ли данный Internet-адрес в одной из переменных, установленных сервером. Оператор **strpos($GLOBALS["SCRIPT_NAME"], $url)** будет возвращать либо номер, если строка, хранящаяся в переменной **$url,** присутствует внутри глобальной переменной **SCRIPT_NAME**, либо значение **false,** если это не так.

Чтобы использовать этот класс страницы, файл **page.inc** нужно включить в сценарий и обратиться к функции **Display()**.

Код, приведенный в листинге 6.2, будет создавать начальную страницу сайта компании TLA Consulting и обеспечивать вывод, который очень похож на сгенерированный ранее и показанный на рис. 5.2.

Листинг 6.2 home.php — эта начальная страница использует класс Page для выполнения большей части действий, требуемых для генерации страницы

```
<?
   require ("page.inc");

   $homepage = new Page();

   $homepage -> SetContent("<p>Welcome to the home of TLA Consulting.
                            Please take some time to get to know us.</p>
                            <p>We specialize in serving your business needs
                            and hope to hear from you soon.</p>"
                            );
   $homepage -> Display();
?>
```

Код из листинга 6.2 выполняет следующие действия:

1. Использует оператор **require** для включения содержимого файла **page.inc,** содержащего определение класса **Page.**
2. Создает экземпляр класса **Page** с именем **$homepage**.
3. Внутри объекта **$homepage** вызывает операцию **SetContent()** и передает некоторый текст и HTML-дескрипторы, которые должны отображаться на странице.
4. Внутри объекта **$homepage** вызывает операцию **Display()**, обеспечивающую отображение страницы в окне браузера посетителя.

Как видно из листинга 6.2, для генерации новых страниц с использованием класса Page требуется выполнение очень незначительного объема работы. Такое использование класса обусловливает очень высокую степень подобия всех страниц.

Если требуется, чтобы в некоторых разделах сайта использовался вариант стандартной страницы, можно просто скопировать **page.inc** в новый файл **page2.inc** и внести в него некоторые изменения. В этом случае при каждом обновлении или исправлении в файле **page.inc** нужно не забыть внести эти же изменения в файл **page2.inc**.

Более рациональный подход — использование наследования для создания нового класса, который наследует большую часть своих функциональных возможностей от класса **Page,** но перекрывает те части, которые должны отличаться.

Применительно к сайту TLA требуется, чтобы страница служб содержала вторую навигационную панель.

Сценарий, приведенный в листинге 6.3, решает эту задачу, создавая новый класс **ServicesPage**, унаследованный от класса **Page.** В этом классе определяется новый массив **$row2buttons**, который содержит кнопки и связи, необходимые во второй строке. Поскольку необходимо, чтобы в основном этот класс вел себя подобно родительско-

му классу, мы выполняем перекрытие только той части, которая должна изменяться — операции **Display().**

Листинг 6.3 services.php — страница служб наследует данные и операции из класса Page, но в ней выполняется перекрытие операции Display() с целью изменения вывода

```
<?
   require ("page.inc");

   class ServicesPage extends Page
   {
      var $row2buttons = array( "Re-engineering" => "reengineering.php",
                                "Standards Compliance" => "standards.php",
                                "Buzzword Compliance" => "buzzword.php",
                                "Mission Statements" => "mission.php"
                              );
      function Display()
      {
         echo "<html>\n<head>\n";
         $this -> DisplayTitle();
         $this -> DisplayKeywords();
         $this -> DisplayStyles();
         echo "</head>\n<body>\n";
         $this -> DisplayHeader();
         $this -> DisplayMenu($this->buttons);
         $this -> DisplayMenu($this->row2buttons);
         echo $this->content;
         $this -> DisplayFooter();
         echo "</body>\n</html>\n";
      }
   }
   $services = new ServicesPage();
   $content ="<p>At TLA Consulting, we offer a number of services.
              Perhaps the productivity of your employees would
              improve if we re-engineered your business.
              Maybe all your business needs is a fresh mission
              statement, or a new batch of buzzwords.";
   $services -> SetContent($content);
   $services -> Display();
?>
```

Перекрытая операция **DisplayO** очень похожа на операцию родительского класса, но содержит дополнительную строку

```
$this -> DisplayMenu($this->row2buttons);
```

для второго вызова операции **DisplayMenu()** и создания второй панели меню.

Вне определения класса мы создаем экземпляр класса **ServicesPage,** устанавливаем значения, которые должны отличаться от значений по умолчанию, и вызываем операцию **Display().**

Как видно из рис. 6.2, мы получаем новый вариант стандартной страницы. При этом пришлось создать код только для отличающихся частей страницы.

Создание страниц при помощи PHP-классов обладает очевидными преимуществами. Учитывая, что класс выполняет большую часть действий, для создания новой страницы приходится выполнять меньший объем работы. Можно обновлять сразу все страницы, просто обновляя класс. Используя наследование, из оригинала можно получать различные версии класса, не теряя при этом упомянутые преимущества.

Однако, как чаще всего случается в жизни, эти преимущества имеют свою цену.

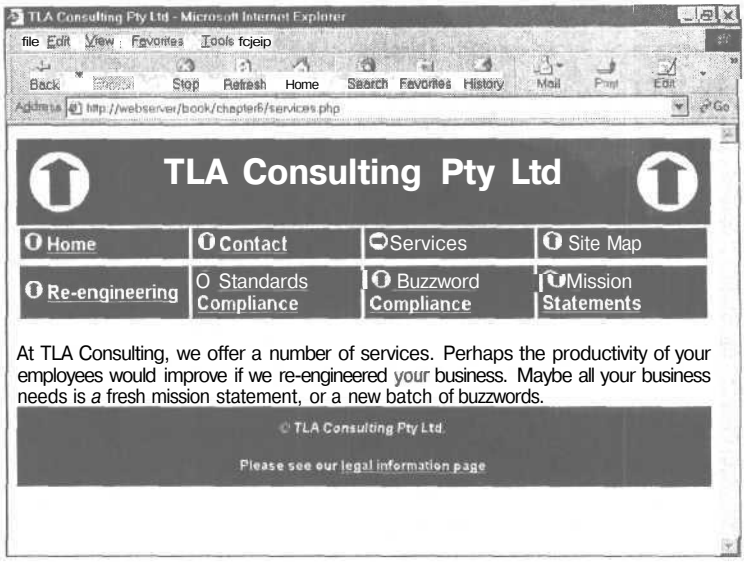


РИСУНОК 6.2
Страница служб создается с использованием наследования с целью повторного использования большей части стандартной страницы.

Для создания страниц из сценария требуется больше операций процессора, чем для простой загрузки статической HTML-страницы с диска и пересылки ее в браузер. Для перегруженного сайта это будет важно, и придется либо использовать статические HTML-страницы, либо по возможности кешировать вывод сценариев, дабы уменьшить нагрузку на сервер.

Что дальше

Следующий раздел посвящен MySQL. Мы рассмотрим создание и заполнение базы данных MySQL, а затем воспользуемся имеющимися знаниями PHP для получения доступа к базе данных из Web.

Часть 2

Использование MySQL

В этой части:

7 Проектирование Web-баз данных

8 Создание Web-базы данных

9 Работа с базой данных MySQL

10 Доступ к базе данных MySQL из Web с помощью PHP

11 Дополнительные возможности MySQL

Глава 7

Проектирование Web-баз данных

Теперь, когда вы уже знакомы с основами PHP, можно начинать рассматривать процесс интегрирования баз данных в сценарии. Возможно, вы помните, в главе 2 обсуждались преимущества использования реляционных баз данных по сравнению с двумерными файлами. Среди них:

- СУРБД обеспечивают более быстрый доступ к данным, чем двумерные файлы.
- СУРБД можно просто отправить запрос на поиск наборов данных, отобранных по определенному критерию.
- СУРБД обладают встроенным механизмом для работы с параллельным доступом, так что вам, как программисту, беспокоиться об этом не нужно.
- СУРБД обеспечивают произвольный доступ к данным.
- СУРБД имеют встроенные системы поддержки привилегий.

Если говорить более предметно, то использование реляционной базы данных дает возможность быстро и без особых усилий ответить на такие вопросы, как: "откуда ваши покупатели?", "какой тип продукции продается наиболее эффективно?" или "какие покупатели тратят больше денег?" Эта информация может помочь улучшить сайт, дабы не только не потерять клиентов, но и привлечь новых. В данном разделе пойдет речь о базах данных MySQL. Однако, прежде чем углубиться в ее специфику (в следующей главе), необходимо прояснить некоторые вопросы:

- Концепции и терминология реляционных баз данных.
- Проектирование Web-баз данных.
- Архитектура Web-баз данных.

Содержание последующих глав:

- Глава 8 описывает основные конфигурации, необходимые для подключения базы данных MySQL к Web.
- Глава 9 расскажет, как производить поиск в базе данных, добавлять и уничтожать записи, работая в командной строке.
- Глава 10 объяснит технологию объединения PHP и MySQL, чтобы была возможность использовать и администрировать базу данных через Web-интерфейс.
- Глава 11 раскрывает некоторые прогрессивные новшества MySQL, которые могут пригодиться во время разработки более требовательных Web-приложений.

Общее представление о реляционных базах данных

Реляционные базы данных на сегодняшний день являются, пожалуй, наиболее часто используемыми. Они основаны на мощном теоретическом базисе реляционной алгебры. Для того чтобы пользоваться реляционными базами данных (а это весьма неплохая вещь!), вовсе необязательно штудировать алгебраическую теорию, однако все же основными понятиями о базах данных следует овладеть.

Таблицы

Реляционные базы данных построены на основе отношений, обычно называемых таблицами. Таблица представляет собой именно то, что и должна — таблицу с данными. Если вы когда-либо имели дело с электронной крупноформатной таблицей, считайте, что опыт работы с реляционными таблицами у вас есть.

Рассмотрим пример.

На рис. 7.1 показана простая таблица. В ней размещены имена и адреса клиентов книжного магазина "Book-O-Rama".

РИСУНОК 7.1
Данные о покупателях магазина "Book-O-Rama" размещены в таблице.

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

У таблицы есть название — Customers (Клиенты), некоторое количество столбцов, каждый из которых содержит определенного рода данные, и строки, в которых записаны клиенты.

Столбцы

Каждый столбец в таблице имеет уникальное имя и содержит различную информацию. Каждому столбцу отвечает определенный тип данных. Например, в таблице Customers на рис. 7.1 можно увидеть, что идентификатор клиента (CustomerID) представлен целым числом, а остальные три столбца являются строками. Столбцы иногда называют также полями или атрибутами.

Строки

Каждая строка в таблице представляет отдельного клиента. Вследствие табличного формата у всех строк одни и те же атрибуты. Строки также называют записями или кортежами.

Значения

Каждая строка состоит из набора персональных значений, соответствующих столбцам. Каждому значению отвечает тип данных, задаваемый столбцом.

Ключи

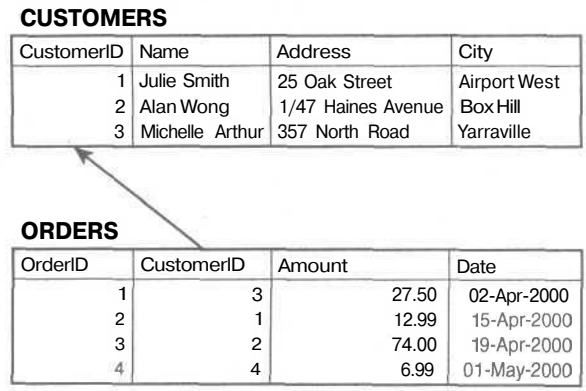
Клиентов нужно различать. Имена для этой цели подходят не самым лучшим образом — если ваше имя достаточно распространенное, вы, надеюсь, понимаете почему. Взять, к примеру Julie Smith из таблицы Customers. Если открыть телефонную книгу, там найдется слишком много людей с такими же именем и фамилией.

Есть несколько способов отличить Julie от других. Вероятнее всего, например, то, что она — единственная Julie Smith, живущая по адресу "25, Oak Street, Airport West". Да только строка эта получается длинноватой и звучит чересчур официально. К тому же в одном столбце таблицы ее не поместить.

Мы поступили следующим образом (наверняка, и вы сделаете так же) — каждому клиенту присвоили уникальный CustomerID (идентификатор клиента). Принцип точно такой же, как в банке, где выдают номер счета, или в клубе, где каждый имеет собственную членскую карточку. При этом условии информацию о клиенте легче разместить в базе данных, а искусственный номер претендует на звание уникального. Причем для того, чтобы присвоить этот номер, вовсе не обязательно располагать обильной информацией о клиенте.

Столбец идентификации в таблице называется *ключом* или *первичным ключом*. Ключ может состоять из нескольких столбцов. Если, например, мы захотим идентифицировать Julie как "Julie Smith, of 25 Oak Street, Airport West", то ключ составят столбцы: Name (Имя), Address (Адрес), City (Город), и его уникальность гарантировать будет сложно.

В базах данных обычно содержится несколько таблиц, а ключ служит связующим звеном между ними. На рис. 7.2 в базу данных добавлена вторая таблица. В ней размещаются заказы, сделанные клиентами. Каждая строка в таблице Orders (Заказы) представляет один заказ, сделанный одним клиентом. Мы знаем, каким именно клиентом, поскольку храним его CustomerID (идентификатор клиента). Можно, к примеру, посмотреть на заказ с OrderID 2 и узнать, что его осуществил клиент с CustomerID 1. Затем, обратившись к таблице Customers, узнаем, что CustomerID 1 принадлежит клиенту по имени Julie Smith.



CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

ORDERS

OrderID	CustomerID	Amount	Date
1	3	27.50	02-Apr-2000
2	1	12.99	15-Apr-2000
3	2	74.00	19-Apr-2000
4	4	6.99	01-May-2000

РИСУНОК 7.2
Каждый заказ в таблице Orders соответствует клиенту из таблицы Customers.

По терминологии реляционных таблиц такое отношение называется *внешним ключом*. CustomerID — первичный ключ в таблице Customers, однако когда он появляется в другой таблице, например, Orders, он уже становится внешним ключом.

Возможно, вызывает удивление наше решение использовать две разные таблицы - почему бы просто не сохранить адрес Julie в таблице Orders? Более детально это объясняется в следующем параграфе.

Схемы

Полный набор проектов таблиц для базы данных называется *схемой* базы данных. В чем-то схема подобна чертежу. Схема должна показывать таблицы вместе с их столбцами, типы данных столбцов, указывать первичный ключ каждой таблицы и все внешние ключи. Схема не включает в себя данные, однако вам, возможно, захочется вписать некоторую информацию просто для примера, как образец, чтобы ориентироваться, что и для чего нужно. Схемы могут представляться в виде диаграмм, которыми пользуемся мы, в виде диаграмм "сущность-отношение" (в данной книге они не рассматриваются) или в форме текста, как то:

Customers(CustomerID. Name, Address, City)

Orders(OrderID. CustomerlD, Amount, Date)

Подчеркнутые элементы в схеме являются первичными ключами в том отношении, в котором они подчеркнуты. Элементы, подчеркнутые пунктиром, — внешние ключи в том отношении, в котором они появляются -подчеркнутыми пунктиром.

Отношения

Внешние ключи представляют отношения между данными в двух таблицах. Например, связь Orders с Customers представляет отношение между строкой в таблице Orders и строкой в таблице Customers.

В реляционной базе данных существуют три основных типа отношений. Их классифицируют в зависимости от количества элементов по каждую сторону отношения. Различают отношения типа "один к одному", "один ко многим", "многие ко многим".

Отношение "один к одному" означает, что в отношении участвует по одному из каждых элементов. Например, если поместить адреса не в таблицу Customers, а в какую-либо другую, между этими таблицами будет отношение "один к одному". От Addresses к Customers может лежать внешний ключ или другой похожий вариант (не обязательно оба вместе).

При отношении "один ко многим" одна строка в первой таблице ссылается на несколько строк другой таблицы. В нашем примере один Customer (Клиент) может сделать несколько Orders (Заказов). В таких отношениях таблица, в которой хранится несколько строк, будет иметь внешний ключ к таблице с одной строкой. Для того чтобы проиллюстрировать этот пример, мы вставили CustomerlD в таблицу Orders.

Если существует отношение типа "многие ко многим", значит, несколько строк одной таблицы ссылаются на несколько строк другой. Предположим, существуют две таблицы Books (Книги) и Authors (Авторы). Одну книгу могли написать несколько авторов, каждый из которых написал и другие книги, причем некоторые из них могли быть написаны опять же в соавторстве с другими. Такой тип отношений, как правило, полностью замыкает все таблицы друг на друга, так что у вас могут быть и Books, и Authors, и Books_Authors. Третья таблица будет содержать только ключи из других

таблиц в качестве попарных внешних ключей, чтобы показать, какие авторы к каким книгам имеют отношение.

Как спроектировать собственную Web-базу данных

Способность чувствовать, когда возникает необходимость в новой таблице и знание ключей, нужных в этой таблице, можно смело отнести к искусству. Вы, конечно, можете перечитывать горы научной литературы, освещающей диаграммы "сущность-отношение" и нормализацию баз данных. Данная книга подобных вопросов не раскрывает. В любом случае, не забывайте о базовых принципах проектирования. Рассмотрим их в контексте нашего проекта "Book-O-Rama".

Подумайте о реальных объектах, которые вы моделируете

При создании базы данных вы, как правило, моделируете предметы и отношения реального мира и сохраняете информацию об этих объектах и отношениях.

В целом каждый класс реальных объектов, которые моделируются, потребует собственной таблицы. Задумайтесь: мы хотим содержать одинаковую информацию обо всех наших клиентах, а если есть набор данных одинаковой "формы", мы запросто можем создать таблицу, отвечающую этим данным.

В примере с магазином "Book-O-Rama" потребуются данные о клиентах, о продаваемых книгах и об особенностях заказов. Книги обладают соответствующими атрибутами: ISBN, автор, название и цена.

Исходя из этого, делаем вывод, что в базе данных необходимы, как минимум, три таблицы: Customers (Клиенты), Orders (Заказы) и Books (Книги). Исходная схема представлена на рис. 7.3.

В данном случае, глядя на модель, нельзя узнать, какие книги были востребованы в каждом заказе. Этим вопросом займемся несколько позже.

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

ORDERS

OrderID	CustomerID	Amount	Date
1	3	27.50	02-Apr-2000
2	1	12.99	15-Apr-2000
3	2	74.00	19-Apr-2000
4	4	6.99	01-May-2000

BOOKS

ISBN	Author	Title	Price
0-672-31687-8	Michael Morgan	Java 2 for Professional Developers	34.99
0-672-31745-1	Thomas Down	Installing Debian GNU/Linux	24.99
0-672-31509-2	Pruitt, et al.	Teach Yourself GIMP in 24 Hours	24.99

РИСУНОК 7.3
Исходная схема состоит из таблиц Customers, Orders и Books.

Избегайте хранения избыточной информации

Немногим ранее мы задавались вопросом: "Почему бы просто не сохранить адрес Julie Smith в таблице Orders?"

Если Julie закажет в магазине "Book-O-Rama" несколько книг (на что мы искренне надеемся), данные *о* ней придется перезаписывать несколько раз. В итоге таблица Orders может принять вид, показанный на рис. 7.4.

ORDERS

OrderID	Amount	Date	CustomerID	Name	Address	City
12	199.50	25-Apr-2000	1	Julie Smith	28 Oak Street	Airport West
13	43.00	29-Apr-2000	1	Julie Smith	28 Oak Street	Airport West
14	15.99	30-Apr-2000	1	Julie Smith	28 Oak Street	Airport West
15	23.75	01-May-2000	1	Julie Smith	28 Oak Street	Airport West

РИСУНОК 7,4 *Проект базы данных, в котором хранится избыточная информация, занимает больше места и может вызывать аномальные явления в используемых данных.*

С этим связаны две основные проблемы.

Во-первых, имеет место расточительная трата пространства на жестком диске. Зачем сохранять информацию о Julie трижды, если необходимо и достаточно всего одного раза?

Во-вторых, это может привести к *аномалиям обновлений,* т.е. к ситуациям, когда мы обновляем базу данных, а в результате получаем несоответствия. Нарушается целостность данных, после чего неизвестно, где верная информация, а где нет. Как правило, подобного рода ситуации оборачиваются потерей данных.

Следует избегать трех типов аномалий обновлений: аномалию модификации, ввода и удаления.

Если Julie, ожидая заказа, переедет в другой дом, ее адрес придется менять в трех местах, а не в одном, проделывая в три раза больше работы. Можно подойти к этой проблеме и с альтернативной стороны, изменив адрес всего в одном месте, однако это приведет к еще худшим последствиям — несоответствию данных в базе. Такие проблемы называют *аномалиями модификации,* поскольку они появляются вследствие попыток модификации базы данных.

Имея такой проект, потребуется вводить данные о Julie каждый раз, принимая заказ; таким образом, постоянно нужно проверять, соответствуют ли существующие данные записанным в таблице. Если этого не делать, то вскоре в таблице могут появиться две строки с противоречащей друг другу информацией о Julie. Скажем, одна строка сообщает нам, что Julie живет в Airport West, а другая — что в Airport. Такая проблема называется *аномалией ввода,* поскольку появляется при вводе данных.

Третий тип аномалий называется *аномалией удаления,* поскольку появляется (сюрприз, сюрприз !!!) при удалении строк из базы данных. Для примера представим, что после выполнения заказ удаляется из базы данных. То есть, как только все текущие заказы Julie выполнены, их всех удаляют из таблицы Orders. А это означает, что у нас больше нет записей об адресе Julie. Мы не сможем прислать ей какие-либо специальные предложения, и в следующий раз, когда она пожелает заказать что-нибудь у нас, придется опять собирать все ее данные.

Логично было бы предположить, что необходима база данных, в которой нет ни одной из перечисленных аномалий.

Используйте атомарные значения столбцов

Это значит, что в каждом атрибуте каждой строки сохраняется только один элемент. Например, требуется узнать, какие книги отобраны для определенного заказа. Этого можно достичь несколькими путями.

Можно добавить в таблицу Orders столбец, в котором будет размещен список всех заказанных книг, как показано на рис. 7.5.

ORDERS

OrderID	CustomerID	Amount	Date	Books Ordered
1	3	27.50	02-Apr-2000	0-672-31697-8
2	1	12.99	15-Apr-2000	0-672-31745-1, 0-672-31509-2
3	2	74.00	19-Apr-2000	0-672-31697-8
4	3	6.99	01-May-2000	0-672-31745-1, 0-672-31509-2, 0-672-31697-8

РИСУНОК 7.5 *При таком подходе атрибут Books Ordered (Заказанные Книги) в каждой строке имеет несколько значений.*

Этот вариант не очень хорош в силу нескольких причин. В этом случае мы помещаем целую таблицу в один столбец — таблицу, которая показывает отношения заказов к книгам. Поступая таким образом, будет очень сложно ответить на вопрос типа: "Сколько экземпляров *PHP4. Учебник* было заказано?" Система больше не сможет просто подсчитывать записи, которые совпадают друг с другом. Вместо этого она проанализирует значение каждого атрибута, чтобы найти внутри него любые совпадения.

Поскольку таким образом мы создаем таблицу в таблице, то стоило бы, на самом деле, просто создать новую таблицу. Она называется Order_Items (Заказанные Экземпляры) и показана на рис. 7.6.

ORDER_ITEMS

OrderID	ISBN	Quantity
1	0-672-31697-8	1
2	0-672-31745-1	2
2	0-672-31509-2	1
3	0-672-31697-8	1
4	0-672-31745-1	1
4	0-672-31509-2	2
4	0-672-31697-8	1

РИСУНОК 7.6
Этот проект упрощает процесс поиска заказанных книг

Эта таблица обеспечивает связь между таблицами Orders и Books. Такая таблица весьма типична для случаев, когда между двумя объектами существует отношение типа "многие ко многим" — т.е. один заказ может включать в себя несколько книг, а какую-либо книгу могут заказать несколько человек.

Выбирайте подходящие ключи

Будьте уверены в том, что выбранные ключи уникальны. Как раз для этого мы создали специальные ключи для клиентов (CustomerID) и для заказов (OrderID), поскольку у этих реальных объектов может не оказаться идентификатора, претендующего на звание уникального. Книгам создавать подобный идентификатор не требуется, он у них уже есть — это ISBN. Для Order_Item можно, при желании, добавить один ключ, однако комбинация таких атрибутов как Order_Item и ISBN будет уникальной до тех

пор, пока заказ двух и больше экземпляров одной и той же книги рассматривается как одна строка. По этой причине в таблице Order_Items присутствует столбец Quantity (Количество).

Подумайте над вопросами, которые потребуется задать базе данных

Продолжая тему, подумайте, на какие вопросы вы хотите получить ответ от базы данных. (Вспомните, о чем говорилось в начале этой главы. Например, какие книги магазина "Book-O-Rama" продаются лучше других?) Прежде чем ожидать точного ответа, убедитесь, содержит ли база данных всю необходимую информацию и существуют ли между таблицами надлежащие связи.

Избегайте проектов с большим количеством пустых атрибутов

Если возникнет необходимость добавить в базу данных рецензии на книги, есть, по меньшей мере, два варианта, как это сделать. Они продемонстрированы на рис. 7.7.

РИСУНОК 7.7 *Чтобы добавить рецензии, можно либо добавить в таблицу Books столбец Review (Рецензии), либо создать специальную таблицу для рецензий.*

BOOKS

ISBN	Author	Title	Price	Review
0-672-31687-8	Michael Morgan	Java 2 for Professional Developers	34.99	
0-672-31745-1	Thomas Down	Installing Debian GNU/Linux	24.99	
0-672-31509-2	Pruitt, et al.	Teach Yourself GIMP in 24 Hours	24.99	

BOOK_REVIEWS

ISBN	Review

Первый вариант подразумевает добавление столбца Review в таблицу Books. В таком случае каждую книгу будет сопровождать поле с рецензией. Если в базе данных много книг и рецензент не собирается делать обзор их всех, во многих строках этот атрибут не будет иметь значения (или, как говорят, будет иметь нулевое значение).

Наличие большого количества нулевых значений в базе данных — плохая практика. Это влечет за собой нецелесообразное использование места на жестком диске, проблемы с подсчитыванием итоговых сумм и прочими функциями числовых столбцов. Когда пользователь видит в таблице нуль, он не знает, что это: либо несоответствующий атрибут, либо в базе данных ошибка, либо данные просто не введены.

Используя альтернативный вариант проекта, можно избежать большинства проблем с нулями. Для этого мы предлагаем второй проект, представленный на рис. 7.7. Здесь в таблице Book_Reviews (Рецензии книг) размещаются только книги с рецензиями (рецензии прилагаются).

Заметьте, что этот проект основан на идее использования единого внутреннего рецензента. С той же легкостью можно обеспечить клиентов рецензиями авторов. Для этого в таблицу Book_Reviews потребуется добавить CustomerID.

Типы таблиц

В принципе, вы придете к выводу, что проект базы данных состоит, как правило, из двух типов таблиц:

- Простые таблицы, описывающие реальные объекты. Они могут иметь ключи к другим простым объектам, которые поддерживают отношения типа "один к од-

ному" и "один ко многим". Например, один клиент может сделать несколько заказов, однако в то же время один заказ приходит от одного клиента. Так, в заказе создается ссылка на клиента.

- Связывающие таблицы, которые описывают отношения типа "многие ко многим" между двумя реальными объектами, например, между Orders и Books. Такие таблицы зачастую ассоциируются с некоторой реальной транзакцией.

Архитектура Web-баз данных

Теперь, когда рассмотрена внутренняя архитектура базы данных, пришло время взглянуть на внешнее построение системы Web-баз данных и обсудить методологию ее разработки.

Архитектура

Основная операция Web-сервера проиллюстрирована на рис. 7.8. Эта система состоит из двух объектов: Web-браузера и Web-сервера. Между ними должен существовать канал связи. Web-браузер посылает запрос на сервер, сервер отсылает обратно ответ. Для сервера, отсылающего обычные статические страницы, такая архитектура подходит. Архитектура же сайта, который включает в себя базу данных, несколько сложнее.

РИСУНОК 7.8
Отношение типа клиент/сервер между Web-браузером и Web-сервером требует наличия связи.



Приложения Web-баз данных, которые будут разрабатываться на страницах этой книги, наследуют глобальную структуру Web-баз данных, показанную на рис. 7.9. Большая часть этой структуры должна выглядеть знакомо.

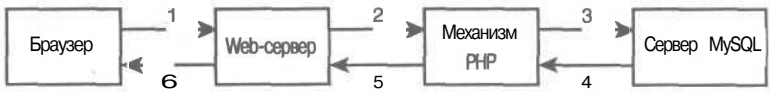


РИСУНОК 7.9 *Базовая архитектура Web-баз данных включает в себя Web-браузер, Web-сереер, сценарный механизм и сервер баз данных.*

Типичная транзакция Web-базы данных состоит из этапов, обозначенных цифрами на рис. 7.9. Мы рассмотрим их на примере магазина "Book-O-Rama".

1. Web-браузер пользователя отправляет HTTP-запрос определенной Web-страницы. Например, поиск в магазине "Book-O-Rama" всех книг, написанных Лорой Томсон (Laura Thomson), используя HTML-форму. Страница с результатами поиска называется results.php.
2. Web-сервер принимает запрос на results.php, получает файл и передает его механизму PHP на обработку.

3. Механизм PHP начинает синтаксический анализ сценария. В сценарии присутствует команда подключения к базе данных и выполнения запроса в ней (поиск книг). PHP открывает соединение с сервером MySQL и отправляет необходимый запрос.
4. Сервер MySQL принимает запрос в базу данных, обрабатывает его, а затем отправляет результаты — в данном случае, список книг — обратно в механизм PHP.
5. Механизм PHP завершает выполнение сценария, форматируя результаты запроса в виде HTML, после чего отправляет результаты в HTML-формате Web-серверу.
6. Web-сервер пересылает HTML в браузер, с помощью которого пользователь просматривает список необходимых книг.

Процесс этот, как правило, протекает вне зависимости от того, какой сценарный механизм и какой сервер баз данных используется. Зачастую программное обеспечение Web-сервера, механизм PHP и сервер баз данных находятся на одной машине. Правда, не менее часто сервер базы данных работает на другой машине. Это делается из соображений безопасности, увеличение объема или разделения потока. С точки зрения перспектив развития, в работе оба варианта одинаковы, однако в плане производительности второй вариант может оказаться более предпочтительным.

Дополнительная информация

В данной главе были раскрыты некоторые принципы проектирования реляционных баз данных. Если вы желаете глубже изучить теорию реляционных баз данных, можете прочесть книги реляционных гуру (например, Си Джей Дейта (C. J. Date)). Однако имейте в виду, что материал, который вы почерпнете, может оказаться по большей части теоретическим и не сразу найдет себе применение в коммерческих Web-разработках. Среднестатистическая база данных не должна быть особо сложной.

Что дальше

В следующей главе мы начнем настраивать базу данных MySQL. Сначала вы научитесь настраивать базу данных MySQL под Web и отправлять ей запросы, а после — отправлять запросы из PHP-кода.

Глава 8

Создание Web-базы данных

В этой главе мы поговорим о методике настройки базы данных MySQL для использования на Web-сайте. Мы обсудим следующие вопросы:

- Создание базы данных.
- Пользователи и привилегии.
- Знакомство с системами привилегий MySQL.
- Создание таблиц баз данных.
- Типы столбцов в MySQL.

В этой главе мы продолжим использовать в качестве примера онлайновый магазин "Book-O-Rama". Напомним схему приложения "Book-O-Rama":

Customers(CustomerID, Name, Address, City)

Orders(OrderID, CustomerID, Amount, Date)

Books(ISBN, Author, Title, Price)

Order_Items(OrderID, ISBN, Quantity)

Book_Reviews(ISBN, Reviews)

Не забывайте, что первичные ключи подчеркивают обычной линией, а внешние ключи — пунктирной.

Чтобы использовать материал данного раздела, необходимо иметь доступ к MySQL. В большинстве случаев подразумевается, что:

1. Проведена базовая установка MySQL на Web-сервере, которая включает в себя:
   - Установку файлов
   - Установку пользователя MySQL
   - Настройку
   - Запуск **mysql_install_db**, если необходимо

- Установку пароля для пользователя root
- Удаление анонимного пользователя (anonymous)
- Запуск сервера MySQL и его настройку на автоматический запуск.

Если все это проделано, можно смело приступать к прочтению данной главы, а если нет — инструкции находятся в приложении А.

Если на каком-либо этапе работы с этой главой возникают проблемы, то они могут быть следствием неправильной настройки системы MySQL. Если это действительно так, обратитесь к вышеприведенному списку и приложению А, дабы устранить все возможные неточности.

2. Имеется доступ к MySQL на машине, где у вас нет прав администратора (например, на машине, поддерживающей службу Web-хостинга, на рабочей станции и т.п.).

   Если это так, то для того чтобы работать с примерами или создать собственную базу данных нужно, чтобы ваш администратор завел вас как пользователя и вверил в ваше распоряжение базу данных, после чего сообщил регистрационное имя, пароль и назначенное имя базы.

   Разделы главы, в которых объясняется, как заводить пользователей и устанавливать базы данных, можно либо пропустить, либо прочесть, чтобы поточнее объяснить свои требования администратору. Обычному пользователю не разрешается выполнять команды подключения пользователей и создания баз данных.

Примеры, приведенные в этой главе, были построены и апробированы в среде MySQL версии 3.22.27. У некоторых более ранних версий MySQL выполняемых функций поменьше. Лучше всего устанавливать наиболее свежую стабильную версию или обновить до таковой существующую. Текущую версию можно выгрузить из сайта MySQL по адресу **http://mysql.com.**

Замечания к использованию монитора MySQL

Легко заметить, что примеры команд MySQL в этой и следующей главах завершаются точкой с запятой (;), которая сообщает MySQL о том, что команду необходимо выполнить. Если точку с запятой не поставить — ничего не произойдет. Начинающие пользователи частенько сталкиваются с подобной проблемой.

Кроме того, это значит, что команду можно вводить в нескольких строках. Мы воспользовались такой возможностью для того, чтобы примеры легче читались. Завершения строк находятся без особого труда, поскольку MySQL выдает символ продолжения. Он выглядит как стрелка следующей формы:

```
mysql> grant select
    ->
```

Это значит, что MySQL ожидает ввода новых символов. До тех пор, пока не будет введена точка с запятой, символы продолжения будут появляться на экране после каждого нажатия клавиши Enter.

Отметьте также и то, что SQL-операторы нечувствительны к регистру, а вот базы данных и названия таблиц — чувствительны. Более подробно об этом — далее в главе.

Вход в систему MySQL

Для входа в систему MySQL в интерфейс командной строки на своей машине и наберите следующее:

```
> mysql -h hostname -u username -p
```

Командная подсказка может выглядеть по-разному, в зависимости от используемых операционной системы и командной оболочки.

Команда **mysql** запускает монитор MySQL. Это клиент командной строки, который выполняет соединение с сервером MySQL.

Ключ -h используется для обозначения хоста, к которому осуществляется подключение — это компьютер с запущенным сервером MySQL. При вводе этой команды на той же машине, на которой находится сервер MySQL, применять этот ключ, равно как и параметр **hostname,** вовсе не обязательно. Если на другой, то параметр **hostname** следует заменить именем машины, на которой выполняется сервер MySQL.

Ключ -и используется для указания имени пользователя, под которым требуется осуществить соединение. Если не указать имя пользователя, по умолчанию будет использоваться имя, под которым вы входили в операционную систему.

Если сервер MySQL установлен на вашем собственном компьютере или сервере, необходимо войти в систему под именем root и создать базу данных, о которой мы поговорим чуть позже в этом разделе. Если установка производилась впервые, то root будет единственным пользователем, который имеет доступ к системе.

Если MySQL используется на машине, которую администрирует другой человек, применяйте имя пользователя, которое вам назначил этот администратор.

Ключ -p сообщает серверу о том, что вы хотите соединиться с использованием пароля. Можете не использовать этот ключ, если для пользователя, под именем которого вы регистрируетесь, пароль не требуется.

Если вы входите в систему под именем root и до сих пор еще не установили пароль, настоятельно рекомендуется перечитать приложение А и побыстрее установить пароль. Без пароля для пользователя root система не защищена.

Включать пароль в эту строку не обязательно. Сервер MySQL запросит его самостоятельно. На самом деле лучше даже его не включать. Если набрать пароль в командной строке, он появится на экране в виде текста и таким образом станет доступным остальным пользователям.

После ввода предыдущей команды должен быть получен приблизительно такой ответ:

```
Enter password: ****
```

(Если этого не произошло, убедитесь в том, что MySQL-сервер запущен и где-то в. вашем пути присутствует команда mysql.)

Необходимо ввести пароль. Если все пройдет хорошо, должен наблюдаться приблизительно такой ответ:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9 to server version: 3.22.34-shareware-debug
Type 'help' for help.
mysql>
```

Если на вашей машине такого ответа не последовало, убедитесь, что была выполнена команда **mysql_install_db** (если это необходимо), установлен и правильно введен пароль для пользователя root.

Если компьютер не ваш, просто убедитесь в корректности ввода пароля.

Сейчас вы должны наблюдать приглашение MySQL на ввод команды, т.е. система готова к созданию базы данных.

Если машина ваша, выполняйте инструкции из следующего раздела.

Если вы заходите в систему с чужого компьютера, все это уже должно быть сделано. В этом случае можно сразу перейти к разделу "Использование требуемой базы данных". Хотя может и возникнуть интерес ознакомиться с промежуточными разделами, однако описанные в них команды выполнить не удастся. (Во всяком случае, не стоит пытаться!)

Создание баз данных и подключение пользователей

Система баз данных MySQL может поддерживать множество различных баз данных. Обычно на одно приложение будет существовать одна база данных. В нашем примере с "Book-O-Rama" база данных будет называться **books.**

Создание базы данных

Это самая простая часть. Введите в командной строке MySQL:

```
mysql> create database dbname;
```

Вместо *dbname* следует ввести имя базы, которую требуется создать. Для целей примера "Book-O-Rama" необходимо создать базу данных с именем **books.**

Вот так. Ответ должен выглядеть приблизительно так:

```
Query OK, 1 row affected (0.06 sec)
```

Это значит, что все сработало как следует. Если подобного ответа не последовало, посмотрите, присутствует ли в конце строки точка с запятой. Точка с запятой сообщает MySQL, что ввод команды завершен и ее пора выполнять.

Пользователи и привилегии

Система MySQL может содержать много пользователей. Пользователь root из соображений безопасности должен использоваться только для административных целей. Каждый пользователь, которому необходимо работать в системе, должен получить учетную запись и пароль. Они не должны быть точно такими же, как вне MySQL (например, имя и пароль, которые используются для входа в UNIX или NT). То же самое относится и к пользователю root. Вообще, разумно иметь разные пароли для входа в систему и для MySQL, особенно, если речь идет о пароле пользователя root.

Пароли для обычных пользователей устанавливать не обязательно, но все же настоятельно рекомендуется сделать это.

При создании Web-базы данных стоит завести хотя бы одного пользователя в каждом Web-приложении.

Вы спросите: "Почему так следует поступать?" -- и ответ подскажут привилегии.

Знакомство с системами привилегий MySQL

Одна из лучших характеристик MySQL заключается в поддержке сложных систем привилегий.

Привилегия — это право определенного пользователя выполнять определенное действие над определенным объектом. Концепция очень похожа на концепцию прав доступа к файлам.

При подключении пользователя к MySQL, ему даются определенные права, которые обозначают, что пользователь может делать в системе, а что не может.

Принцип наименьших привилегий

Принцип наименьших привилегий может использоваться для улучшения защиты любой компьютерной системы.

Это стандартный, но очень важный принцип, который часто упускается из виду. Заключается он в следующем:

> Пользователь (или процесс) должен обладать наименьшим уровнем привилегий, необходимым для выполнения назначенного задания.

Это справедливо для MySQL так же, как и для любой другой системы. Так, чтобы выполнить запрос из Web, обычному пользователю не нужны все те привилегии, которые имеются у пользователя root. Следовательно, потребуется завести еще одного пользователя, который будет обладать всеми необходимыми привилегиями для доступа к только что созданной базе данных.

Установка *пользователей:* команда *GRANT*

Команды **GRANT и REVOKE** используются для того, чтобы предоставлять и отнимать права у пользователей MySQL на четырех уровнях привилегий. Вот эти уровни:

- Глобальный
- Базы данных
- Таблицы
- Столбца

Чуть позже будет показано, как применять каждую из этих команд.

С помощью команды **GRANT** можно заводить пользователей и предоставлять им привилегии. Общий вид команды **GRANT** выглядит следующим образом:

```
GRANT privileges  [columns]
ON item
TO user_name  [IDENTIFIED BY  'password' ]
 [WITH GRANT OPTION]
```

Конструкции, заключенные в квадратные скобки, являются необязательными. В данной синтаксической структуре присутствует ряд заполнителей.

Первый, *privileges (привилегии),* должен заполняться разделенным запятыми списком привилегий, четко определенных в MySQL. Список будет рассматриваться ниже.

Заполнитель *columns (столбцы)* необязателен. Им можно воспользоваться для того, чтобы назначать привилегии по конкретным столбцам. Можно определять либо имя одного столбца, либо разделенный запятыми список имен столбцов.

Заполнитель *item (элемент)* может быть базой данных или таблицей, к которой применяются новые привилегии.

Указав на его месте *.*, можно установить привилегии для всех баз данных. Такое действие называется назначением *глобальных* привилегий. Тот же эффект достигается и указанием лишь *.

Чаще всего задаваться будут все таблицы в определенной базе данных — *dbname.* * *(имя_базы_данных.*)*, конкретная таблица — *dbname.tablename (имя_базы_данных.имя_таблицы)* либо определенные столбцы — *dbname.tablename* и список необходимых столбцов в заполнителе *columns.* Все перечисленное представляет три других доступных уров-

ня привилегий: *базы данных, таблицы* и *столбца,* соответственно. Если при выдаче этой команды используется какая-то конкретная база данных, параметр *tablename* сам по себе будет истолкован как "таблица в текущей базе данных".

В качестве значения *user_name* должно стоять имя пользователя, под которым пользователь должен входить в MySQL. Помните, что оно не должно совпадать с регистрационным именем, под которым входите в систему вы. В MySQL *user_name* может включать в себя и имя хоста, что весьма удобно для того, чтобы различать пользователей, скажем, **laura** (имеется ввиду **laura@localhost**) и **laura@somewhere.com**. Это очень полезная вещь, поскольку часто пользователи на различных доменах имеют одни и те же имена. Кроме того, повышается степень защищенности, поскольку имеется возможность указать, откуда пользователи могут заходить и к каким базам данных или таблицам могут иметь доступ.

Параметр *password* следует заменить на пароль, необходимый для входа. Руководствуйтесь общими правилами использования паролей. О безопасности мы поговорим чуть позже, отметим лишь, что пароль не должен быть легко угадываемым. Не стоит употреблять слово со словаря или совпадающее с именем пользователя. В идеале пароль должен включать в себя буквы верхнего и нижнего регистров и небуквенные символы.

Опция **WITH GRANT OPTION,** если указана, дает право пользователю передавать свои привилегии другим.

Привилегии хранятся в четырех системных таблицах, принадлежащих базе данных с именем mysql. Эти четыре таблицы называются так: mysql.user, mysql.db, mysql.tables_priv и mysql.columns_priv; они напрямую относятся к упомянутым ранее четырем уровням привилегий. Как альтернатива команде **GRANT,** можно непосредственно править эти таблицы. Более детально это обсуждается в главе 11.

Типы и уровни привилегий

В MySQL существуют три основных типа привилегий: привилегии, которые можно давать обычным пользователям; привилегии, которые нужны только администраторам, и множество специальных привилегий. Любой пользователь может получить любые привилегии, тем не менее, администраторы редко делятся с кем-либо своими правами, и причина тому — принцип наименьших привилегий.

Пользователям необходимы привилегии только для баз данных и таблиц, с которыми они работают. Доступ к базе данных mysql должен быть закрыт для всех, кроме администратора, ибо именно в ней хранятся учетные записи пользователей, пароли и т.п. (Эта база рассматривается в главе 11.)

Привилегии для обычных пользователей сводятся к набору определенных команд SQL и правам на их выполнение. Более детально команды SQL рассматриваются в следующей главе. Сейчас же мы лишь описываем, что эти команды делают. Привилегии показаны в табл. 8.1. Элемент столбца Applies To (Применяется к) перечисляет объекты, которым могут быть присвоены привилегии конкретного типа.

В контексте системной безопасности большая часть привилегий обычных пользователей сравнительно безвредна. Привилегия **ALTER** используется при работе чуточку вне системы привилегий, например, для переименования таблиц, однако пользователям она бывает нужна очень часто. Обеспечение безопасности — это предмет вечного спора практичности и надежности. Можно принимать собственные решения касаемо **ALTER,** но не следует забывать, что пользователям ее доверяют достаточно часто.

Таблица 8.1 Привилегии для пользователей

Привилегия	*Применяется к*	Описание
SELECT	таблицам, столбцам	Разрешает пользователям выбирать строки (записи) в таблицах.
INSERT	таблицам, столбцам	Разрешает пользователям вставлять новые строки в таблицы.
UPDATE	таблицам, столбцам	Разрешает пользователям изменять значения в существующих строках таблиц.
DELETE	таблицам	Разрешает пользователям удалять существующие строки в таблицах.
INDEX	таблицам	Разрешает пользователям создавать и удалять индексы определенных таблиц.
ALTER	таблицам	Разрешает пользователям изменять структуру существующих таблиц, добавляя столбцы, переименовывая столбцы или таблицы и изменяя тип данных в столбцах.
CREATE	базам данных, таблицам	Разрешает пользователям создавать новые базы данных или таблицы. Если командой GRANT обозначена определенная база данных или таблица, пользователь может только создавать (CREATE) ту или иную базу данных или таблицу, что подразумевает первоначальное ее удаление.
DROP	базам данных, таблицам	Разрешает пользователям удалять базы данных или таблицы.

Помимо привилегий, перечисленных в табл. 8.1, существует еще привилегия **REFERENCES,** правда, она в настоящий момент не используется, и привилегия **GRANT,** которая выдается в команде **WITH GRANT OPTION.**

В табл. 8.2 описаны привилегии, необходимые администраторам.

Таблица 8.2 Привилегии для администраторов

Привилегия	***Описание***
RELOAD	**Позволяет администратору перезагружать таблицы привилегий и подавлять привилегии, хосты, журнальные файлы и таблицы.**
SHUTDOWN	**Позволяет администратору останавливать сервер MySQL.**
PROCESS	**Позволяет администратору просматривать и удалять процессы на сервере.**
FILE	**Позволяет помещать в таблицы данные из файлов и** наоборот.

Эти привилегии можно выдавать не только администраторам, но прежде чем так поступить, следует трижды подумать. У обычного пользователя не должно возникать необходимости применять привилегии **RELOAD, SHUTDOWN** или **PROCESS.**

С привилегией **FILE** дела обстоят несколько иначе. Она очень удобна для пользователей, поскольку экономит время, позволяя извлекать данные из файла вместо того, чтобы набирать их заново. С другой стороны, у пользователя есть возможность загрузить любой файл, который сервер MySQL в состоянии увидеть, включая базы данных других пользователей и файлы с паролями. Выдавайте ее с осторожностью или предлагайте пользователю загружать данные вместо него.

Существуют две специальных привилегии, которые приводятся в табл. 8.3.

Таблица 8.3 Специальные привилегии

Привилегия	*Описание*
ALL	Предоставляет все привилегии, перечисленные в табл. 8.1 и 8.2. Вместо ALL можно также написать ALL PRIVILEGES.
USAGE	Не предоставляет никаких привилегий. Подобным образом можно подключить пользователя, дать ему возможность входить в систему, но без разрешения что-либо делать. Как правило, со временем такой пользователь все же получает какие-то привилегии.

Команда REVOKE

Противоположностью команде **GRANT** является команда **REVOKE.** Она отнимает у пользователя привилегии и по синтаксису сходна с командой **GRANT:**

```
REVOKE privileges [(columns)]
ON item
FROM user_name
```

Если указан параметр **WITH GRANT OPTION,** его действие можно отменить таким образом:

```
REVOKE GRANT OPTION
ON item
FROM user_name
```

Примеры использования команд GRANT и REVOKE

Для предоставления привилегий администратору наберите:

```
mysql> grant all
    -> on *
    -> to fred identified by 'mnb123'
    -> with grant option;
```

Эта команда дает пользователю Fred с паролем mnb123 все привилегии для всех базах данных с правом передачи привилегий другим пользователям.

Если такой пользователь в системе не нужен, отнимите у него привилегии:

```
mysql> revoke all
    -> on *
    -> from fred;
```

Теперь подключим обычного пользователя безо всяких привилегий:

```
mysql> grant usage
    -> on books.*
    -> to sally identified by 'magic123';
```

Поговорив с Салли (Sally), мы узнаем что-то о ее намерениях и решаем выдать ей необходимые привилегии:

```
mysql> grant select, insert, update, delete, index, alter, create, drop
    -> on books.*
    -> to sally;
```

Заметьте, для этого не требуется указывать пароль Sally.

Если мы пришли к выводу, что Салли что-то натворила в базе данных, ее права можно ограничить:

```
mysql> revoke alter, create, drop
    -> on books.*
    -> from sally;
```

Позже, когда ей не нужно будет пользоваться базой данных, можно отобрать у нее вообще все привилегии:

```
mysql> revoke all
    -> on books.*
    -> from sally;
```

Установка пользователя для доступа из Web

Для соединения PHP-сценариев с MySQL потребуется настроить пользователя. В этом случае также можно применить принцип наименьших привилегий: на что стоит предоставить права сценариям?

В большинстве случаев сценариям понадобится проводить над строками таблиц только такие операции: **SELECT, INSERT, DELETE и UPDATE.** Можно поступить следующим образом:

```
mysql> grant select, insert, delete, update
    -> on books.*
    -> to bookorama identified by 'bookorama123';
```

Не забывайте о безопасности! Такой пароль, конечно, никуда не годится.

Если вы используете службу Web-хостинга, можно предположить, что вы будете иметь доступ к другим пользовательским привилегиям для базы данных, которую создадут для вас. Вам выдадут одно и то же *имя_пользователя* и *пароль* для работы из командной строки (настройка таблиц и т.п.) и для подключения Web-сценариев к MySQL (запросы в базу данных). Это катастрофически снижает безопасность. Пользователя с таким уровнем привилегий можно установить так:

```
mysql> grant select, insert, update, delete, index, alter, create, drop
    -> on books.*
    -> to bookorama identified by 'bookorama123';
```

Теперь можете приступить к настройке второго пользователя.

выход *из системы пользователя root*

Покинуть монитор MySQL можно, набрав **quit.** После стоит войти в систему как пользователь из Web и убедиться, что все работает должным образом.

Использование требуемой базы данных

Если вы дошли до этой стадии, то должны находиться в системе под учетной записью MySQL пользовательского уровня и быть готовыми к тестированию примера кода независимо от того, кто его установил — вы или администратор Web-сервера.

После входа в систему сначала потребуется определить базу данных, с которой необходимо работать. Это можно сделать, набрав:

```
mysql> use dbname;
```

где ***dbname*** — имя соответствующей базы данных.

Можно и не набирать команду use, но тогда следует указать базу данных при входе в систему:

```
mysql dbname -h hostname -u username -p
```

В этом примере воспользуемся базой данных books:

```
mysql> use books;
```

После ввода этой команды MySQL должен выдать такой ответ:

Database changed

Если перед началом работы база данных не была выбрана, MySQL ответит сообщением об ошибке:

```
ERROR 1046: No Database Selected
```

Создание таблиц баз данных

Следующий этап настройки базы данных — создание таблиц. Это делается при помощи SQL-команды **CREATE TABLE.** Общая форма оператора **CREATE TABLE** выглядит следующим образом:

```
CREATE TABLE tablename(columns)
```

Заполнитель ***tablename*** необходимо заменить на название таблицы, которую требуется создать, а *columns* — на разделенный запятыми список столбцов в таблице.

Каждый столбец должен иметь имя, за которым следует тип данных.

Рассмотрим схему "Book-O-Rama":

Customers(CustomerID, Name, Address, City)

Orders(OrderID, CustomerID, Amount, Date)

Books(ISBN, Author, Title, Price)

Order_Items(OrderID, ISBN, Quantity)

Book_Reviews(ISBN, Reviews)

Листинг 8.1 содержит SQL-код для создания этих таблиц, при этом подразумевается, что база данных books уже создана. Этот код можно найти на сопровождающем CD-ROM в файле **chapter8/bookorama.sql**.

SQL-код создания таблиц запускается следующим образом:

```
> mysql -h host -u bookorama books -p < bookorama.sql
```

В данном случае очень удобно использовать переназначение файлов, поскольку предполагается, что до выполнения SQL-код редактируется в любом текстовом редакторе.

Листинг 8.1 bookorama.sql - SQL-код создания таблиц для "Book-O-Rama"

```
create table customers
( customerid int unsigned not null auto_increment primary key,
  name char(30) not null,
  address char(40) not null,
  city char(20) not null
);

create table orders
( orderid int unsigned not null auto_increment primary key,
  customerid int unsigned not null,
  amount float(6,2),
  date date not null
);
```

```
create table books
(   isbn char(13) not null primary key,
    author char(30),
    title char(60),
    price float(4,2)
);
create table order_items
( orderid int unsigned not null,
  isbn char(13) not null,
  quantity tinyint unsigned,
  primary key (orderid, isbn)
);
create table book_reviews
(
  isbn char(13) not null primary key,
  review text
);
```

Каждая из этих таблиц создается отдельным оператором **CREATE TABLE.** Как видите, создаются все таблицы из схемы, со столбцами, разработанными в предыдущей главе. Каждый столбец после своего названия содержит тип данных. Некоторые столбцы содержат и другие спецификаторы.

Что означают другие ключевые слова

NOT NULL означает, что все строки таблицы должны иметь значение в этом атрибуте. Если NOT NULL не указано, поле может быть пустым (NULL).

AUTO_INCREMENT — специальная возможность MySQL, которую можно задействовать в числовых столбцах. Если при вставке строк в таблицу оставлять такое поле пустым, MySQL автоматически генерирует уникальное значение идентификатора. Это значение будет на единицу больше максимального значения, уже существующего в столбце. В каждой таблице может быть не больше одного такого поля. Столбцы с AUTO_INCREMENT должны быть проиндексированными.

PRIMARY KEY после имени столбца определяет, что этот столбец является первичным ключом для таблицы. Данные в этом столбце должны быть уникальными. MySQL автоматически индексирует этот столбец. Заметьте, что ранее, при использовании его с **customerid** в таблице **customers, без AUTO_INCREMENT** не обошлось. Автоматический индекс по первичному ключу сберегает индекс, требуемый **AUTO_INCREMENT.**

Указывать **PRIMARY KEY** после названия столбца следует лишь тогда, когда мы имеем дело с первичным ключом в виде одиночного столбца. Альтернативным вариантом является конструкция **PRIMARY KEY** в конце описания таблицы **order_items.** В последнем случае первичный ключ включает два столбца.

UNSIGNED после целочисленного типа означает, что его значение может быть либо положительным, либо нулевым.

Что означают типы столбцов

Рассмотрим первую таблицу:

```
create table customers
( customerid int unsigned not null auto_increment primary key,
  name char(30) not null,
  address char(40) not null,
  city char(20) not null
);
```

При создании любой таблицы необходимо принять решение касаемо типов столбцов.

В таблице **customers,** как обозначено в схеме, существует четыре столбца. Первый, **customerid**, — это первичный ключ, который определен непосредственно. Согласно нашему решению, он будет представляться целым числом (тип данных int), причем unsigned. Кроме того, нам помогает особенность **auto_increment**, так что MySQL может поработать за нас — уже одной заботой меньше.

Все остальные столбцы будут содержать строковые типы данных. Для них выбран тип char. Он определяет поля фиксированной ширины. Ширина указывается в скобках, поэтому, например, *имя* состоит из 30 символов.

Этот тип данных всегда будет назначать 30 символов для имени, даже если не все символы будут использоваться. Для соблюдения требуемого размера MySQL добавит к данным соответствующее количество пробелов. Альтернативным типом данных является varchar, который использует только необходимый объем (плюс один байт). Таким образом, существует небольшой компромисс — varchar использует меньше места, зато char работает быстрее.

Для реальных клиентов, с реальными именами и адресами, ширина этих столбцов наверняка окажется недостаточной.

Обратите внимание, что все столбцы объявлены **NOT NULL.** Это минимальная оптимизация, в результате которой система будет работать немного быстрее.

Более подробную информацию по оптимизации можно найти в главе 11.

Некоторые операторы **CREATE** отличаются по синтаксису. Взгляните на таблицу **orders:**

```
create table orders
( orderid int unsigned not null auto_increment primary key,
  customerid int unsigned not null,
  amount float(6,2),
  date date not null
);
```

Столбец **amount** хранит числа с плавающей запятой (тип float). Для большинства типов данных с плавающей запятой можно определить ширину отображения данных и количество десятичных разрядов. В данном случае итог по заказу хранится в долларах, поэтому выбрана сравнительно большая ширина отображения итога (6) и два десятичных разряда для представления центов.

Столбец **date** имеет тип данных **date.**

В данной таблице указано, что все столбцы должны быть **NOT NULL,** кроме **amount.** Почему? Когда в базу данных вносится заказ, его необходимо сохранить в таблице **orders,** добавить заказные элементы в **order_items** и только тогда подсчитать сумму заказа. На этапе создания заказа сумма заказа не известна, поэтому ей можно присвоить значение **NULL.**

В таблице **books** обладает похожими характеристиками:

```
create table books
(   isbn char(13) not null primary key,
    author  char(30),
    title  char(60),
    price  float(4,2)
);
```

В этом случае не требуется генерировать первичный ключ, потому что номера ISBN получаются в другом месте. Остальные поля оставлены **NULL,** поскольку в первую

очередь книжный магазин узнает ISBN, а потом уже название книги, авторов и цену. Таблица **order_items** демонстрирует применение первичных ключей со множеством столбцов:

```
create table order_items
( orderid int unsigned not null,
  isbn char(13) not null,
  quantity tinyint unsigned,
  primary key (orderid, isbn)
);
```

Количество экземпляров конкретной книги обозначено как **TINYINT UNSIGNED;** этот параметр может принимать значения от 0 до 255.

Как уже упоминалось ранее, первичные ключи со множеством столбцов должны определяться специальным параметром первичного ключа, что как раз используется в нашем случае.

И наконец, рассмотрим таблицу **book_reviews**:

```
create table book_reviews
(
  isbn char(13) not null primary key,
  review text
);
```

Здесь присутствует новый тип данных, о котором мы еще не говорили. Он предназначен для объемных текстов, например, статей. Существует несколько вариантов данного типа, и они рассматриваются позже.

Чтобы понять процесс создания таблиц в деталях, стоит начать с идентификаторов и названий столбцов в целом, а потом уже перейти к типам данных. Для начала взглянем на созданную нами базу данных.

Просмотр базы данных с помощью команд SHOW и DESCRIBE

Войдите в монитор MySQL и начните работу с базой данных **books.** Таблицы в базе можно просмотреть, набрав:

```
mysql> show tables;
```

MySQL отобразит список таблиц базы данных:

```
+------------------+
| Tables in books  |
+------------------+
| book_reviews     |
| books            |
| customers        |
| order_items      |
| orders           |
+------------------+
5 rows in set (0.06 sec)
```

Командой **show** можно пользоваться и для просмотра списка баз данных:

```
mysql> show databases;
```

Команда **DESCRIBE** дает возможность увидеть дополнительную информацию по конкретной таблице, например, **books:**

```
mysql> describe books;
```

MySQL выдаст информацию, передаваемую пользователем во время создания базы:

```
+--------+-------------+------+-----+---------+----------+
| Field  | Type        | Null | Key | Default | Extra    |
+--------+-------------+------+-----+---------+----------+
| isbn   | char(13)    |      | PRI |         |          |
| author | char(30)    | YES  |     | NULL    |          |
| title  | char(60)    | YES  |     | NOLL    |          |
| price  | float(4,2)  | YES  |     | NULL    |          |
+--------+-------------+------+-----+---------+----------+
4 rows in set (0.05 sec)
```

Эти команды полезны, если требуется вспомнить, какие типы столбцов используются, или если работа осуществляется с чужой базой данных.

Идентификаторы MySQL

Существует четыре вида идентификаторов MySQL — базы данных, таблицы, столбцы (с ними мы уже знакомы) и псевдонимы (о них поговорим в следующей главе).

Базы данных в MySQL отображают базовые файловые структуры, а таблицы — файлы. Это прямым образом влияет на присваиваемые им имена, а также на чувствительность этих имен к регистру — если в установленной операционной системе (ОС) имена файлов и каталогов чувствительны к регистру, то и базы данных и таблицы не останутся к нему равнодушными (как, например, в UNIX), иначе — нет (Windows). Имена столбцов и псевдонимы к регистру не чувствительны, однако нельзя в одном и том же SQL-операторе использовать версии с различными регистрами.

К слову, расположение каталогов и файлов, содержащих данные, будет таким, каким оно установлено в конфигурации. Проверить их расположение можно с помощью **mysqladmin:**

```
mysqladmin variables
```

Краткий список возможных идентификаторов приведен в табл. 8.4. Единственное дополнительное исключение — в идентификаторах нельзя использовать ASCII(0) и ASCII(255) (честно говоря, трудно даже предположить, для чего они могли бы пригодиться).

Таблица 8.4 Идентификаторы MySQL

Тип	*Макс. длина*	*Чувствительность к регистру*	*Допустимые символы*
База данных	**64**	**так же, как в ОС**	**Все символы, допустимые в именах каталогов вашей ОС, за исключением символа /**
Таблица	**64**	**так же, как в ОС**	**Все символы, допустимые в именах файлов вашей ОС, за исключением символов / и .**
Столбец	**64**	**нет**	**Все**
Псевдоним	**255**	**нет**	**Все**

Эти правила являются предельно открытыми.

В идентификаторах MySQL 3.23.6 можно даже использовать зарезервированные слова и специальные символы всех видов; единственное ограничение — если вы все же применяете всякого рода странности, то их необходимо заключать в обратные кавычки (на большинстве клавиатур они находятся на клавише с тильдой (~) в верхнем левом углу). Например:

```
create database `create database`;
```

В предыдущих версиях MySQL (до 3.23.6) ограничений больше, и там такого делать нельзя.

Однако какой бы ни была свобода ваших действий, не стоит забывать о здравом смысле. То, что вы *можете* назвать базу данных **`create database`**, вовсе не означает, что вы так и *должны* делать. Здесь, как и в любой другой области программирования, в основе лежит один принцип — идентификаторы должны быть значимыми.

Типы данных столбцов

В MySQL определены три базовых типа столбцов: числовой, дата и время и строчный. Каждая из этих категорий подразделяется на множество типов. Мы кратко рассмотрим их в этой главе, а со всеми их преимуществами и недостатками разберемся в главе 11.

Каждый из трех типов обладает различной емкостью. Выбирая тип столбца, главное — выбрать наименьший, в котором помещаются данные.

У большинства типов данных при создании столбца выбранного типа можно указывать максимальную ширину отображения. В приведенных ниже таблицах для типов данных этот параметр обозначается как *M*. Если для выбранного типа он не обязателен, его приводят в квадратных скобках. Максимальным значением *M* является 255.

Вообще во всей книге необязательные значения заключаются в квадратные скобки.

Числовые типы

Числовые типы представляют собой либо целые числа, либо числа с плавающей запятой. Для чисел с плавающей запятой можно указывать количество цифр после десятичной точки. В нашей книге этот параметр обозначен как *D*. Максимальное значение, которое можно выбрать для *D* составляет 30, или $M-2$ (т.е. максимальная ширина отображения минус два — один символ для десятичной точки и второй для целой части числа), смотря что окажется меньшим.

Целый тип может также быть **UNSIGNED,** как показано в листинге 8.1.

Всем числовым типам можно присвоить атрибут **ZEROFILL.** Такие значения будут отображаться на экране с ведущими нулями.

Целочисленные типы показаны в табл. 8.5. Обратите внимание, что диапазоны приводятся как для чисел со знаком, так и для беззнаковых чисел (в двух строках).

Таблица 8.5 Целочисленные типы данных

Тип	*Диапазон*	*Память (байт)*	*Описание*
TINYINT [(M)]	-127..**128** **или** 0..255	**1**	**Очень маленькие целые числа**
SMALLINT [(M)]	-32768..32767 **или** 0..65535	**2**	**Маленькие целые числа**
MEDIUMINT **[(M)]**	-8388608..8388607 **или** 0..16777215	**3**	**Средней величины целые числа**
INT [(M)]	$-2^{31}..2^{31}-1$ **или** $0..2^{32}-1$	**4**	**Обычные целые числа**
INTEGER [(M)]			**Синоним INT**
BIGINT [(M)]	$-2^{63}..2^{63}-1$ **или** $0..2^{64}$ **- 1**	**8**	**Большие целые числа**

Типы данных с плавающей запятой показаны в табл. 8.6.

Таблица 8.6 Типы данных с плавающей запятой

Тип	*Диапазон*	*Память (байт)*	*Описание*
FLOAT *(precision)*	зависит от precision (точности)	различна	Может использоваться для определения чисел с плавающей запятой одинарной или двойной точности.
FLOAT [(M, D)]	±1.175494351E-38 ±3.402823466E+38	4	Числа с плавающей запятой одинарной точности. Эквивалентно FLOAT(4), но только с указанной шириной отображения и количеством десятичных разрядов.
DOUBLE [(M, D)]	±1.7976931348623157E +308 ±2.2250738585072014E -308	8	Числа с плавающей запятой двойной точности. Эквивалентно FLOAT(8), но только с указанной шириной отображения и количеством десятичных разрядов.
DOUBLE PRECISION [(M, D)]	-"-		Синоним DOUBLE [(M, D)].
REAL [(M, D)]	-"-		Синоним DOUBLE [(M, D)].
DECIMAL [(M[, D])]	различен	M+2	Число с плавающей запятой, сохраненное как **char.** Диапазон зависит от M, ширины отображения.
NUMERIC [(M, D)]	-"-		Синоним DECIMAL.

Типы даты и времени

MySQL поддерживает определенное количество типов даты и времени, которые показаны в табл. 8.7. С помощью этих типов можно вводить данные либо в строчном, либо в числовом форматах. Нет ничего сложного в установке столбца типа **TIMESTAMP** в определенной строке равным дате и времени последней операции, выполненной над этой строкой, даже если не делать это вручную. Это предельно полезно для фиксации транзакций.

Таблица 8.7 Типы данных даты и времени

Тип	*Диапазон*	*Описание*
DATE	1000-01-01 9999-12-31	Дата. Отображается в виде YYYY-MM-DD (ГГГГ-ММ-ДД).
TIME	-838:59:59 838:59:59	Время. Отображается в виде HH:MM:SS (ЧЧ:ММ:СС). Легко заметить, что область значительно шире, чем может когда-либо пригодиться.
DATETIME	1000-01-01 00:00:00 9999-12-31 23:59:59	Дата и время. Отображается в виде YYYY-MM-DDHH:MM:SS (ГГГГ-ММ-ДДЧЧ:ММ:СС).

Тип	Диапазон	Описание
TIMESTAMP [(M)]	1970-01-01 00:00:00 Какой-то момент в 2037 году	Метка времени, полезная для отчетов по транзакциям. Формат отображения зависит от значения M (см. ниже табл. 8.8). Верхнее значение диапазона зависит от ограничений UNIX.
YEAR [(2|4)]	70-69 (1970-2069) 1901-2155	Год. Можно определить двух- или четырехциферный формат. Каждый из них, как показано, обладает своим диапазоном.

Таблица 8.8 перечисляет возможные типы отображения для формата TIMESTAMP.

Таблица 8.8 Типы отображения для формата TIMESTAMP

Тип	Формат отображения
TIMESTAMP	**YYYYMMDDHHMMSS**
TIMESTAMP (14)	**YYYYMMDDHHMMSS**
TIMESTAMP (12)	**YYMMDDHHMMSS**
TIMESTAMP (10)	**YYMMDDHHMM**
TIMESTAMP (8)	**YYYYMMDD**
TIMESTAMP (6)	**YYMMDD**
TIMESTAMP (4)	**YYMM**
TIMESTAMP (2)	**YY**

Строковые типы

Строковые типы разделяются на три группы. Первая группа — простые старые строки, которые представляют собой короткие фрагменты текста. Это типы CHAR (символы с фиксированной длиной) и **VARCHAR** (символы с произвольной длиной). Ширину каждого из них можно регулировать. Столбцы с типом **CHAR** будут дополняться пробелами до максимальной ширины, независимо от размеров данных, в то время как в столбцах с типом **VARCHAR** ширина зависит от размеров данных. (Следует отметить, что MySQL усекает пробелы в конце текстовых строк у **CHAR** во время *извлечения* и у **VARCHAR** во время *сохранения.)* Имеет место компромисс, связанный с этими типами — пространство или скорость, о котором пойдет речь в главе 11.

Вторая группа — это типы **TEXT и BLOB.** Их размеры могут быть разными. Первый тип данных предназначен для более длинных текстовых фрагментов, второй — для двоичных данных. BLOB означает *binary large object (большой двоичный объект)* и может содержать любые данные, в том числе звуки и изображения.

На практике столбцы **TEXT и BLOB** одинаковы, за исключением того, что TEXT чувствителен к регистру, а **BLOB** — нет. Поскольку эти типы столбцов хранят большие массивы данных, они требуют более детального анализа, который выполняется в главе 11.

К третьей группе принадлежат два специальных типа SET и ENUM. Тип SET предназначен для того, чтобы определять, что значения в данном столбце принадлежат конкретному набору фиксированных значений. Значения столбца могут содержать несколько значений из набора. В определяемом наборе можно задать вплоть до 64 элементов.

ENUM представляет собой перечисление. Этот тип очень похож на SET, но только столбцы этого типа могут иметь всего лишь одно из фиксированных значений или NULL, а максимальное количество элементов в перечислении составляет — 65535.

В табл. 8.9, 8.10 и 8.11 приводится обзор строковых типов данных. Таблица 8.9 описывает простые строковые типы.

Таблица 8.9 Обычные строковые типы

Тип	*Диапазон*	*Описание*
[NATIONAL] CHAR (M) [BINARY]	**От 1 до 255 символов**	**Строки фиксированной длины *M*, где *M* находится между 1 и 255. Ключевое слово NATIONAL указывает на то, что должен использоваться набор символов, установленный по умолчанию. В MySQL по умолчанию так и происходит, однако стоит обратить на это внимание, поскольку это является частью ANSI-стандарта SQL. Ключевое слово BINARY указывает, что данные необходимо воспринимать как нечувствительные к регистру. (По умолчанию данные воспринимаются как чувствительные к регистру.)**
[NATIONAL] VARCHAR (M) [BINARY]	**От 1 до 255 символов**	**То же самое, за исключением того, что длина у VARCHAR произвольна.**

Таблица 8.10 описывает типы TEXT и BLOB. Максимальная длина поля TEXT в символах равна максимальному размеру файла в байтах, который может храниться в этом поле.

Таблица 8.10 Типы TEXT и BLOB

Тип	*Максимальная длина (в символах)*	*Описание*
TINYBLOB	2^8-1 **(т.е. 255)**	**Маленькое поле BLOB**
TINYTEXT	2^8-1 **(т.е. 255)**	**Маленькое поле TEXT**
BLOB	$2^{16}-1$ **(т.е. 65535)**	**Нормальное поле BLOB**
TEXT	$2^{16}-1$ **(т.е. 65535)**	**Нормальное поле TEXT**
MEDIUMBLOB	$2^{24}-1$ **(т.е. 16777215)**	**Среднее поле BLOB**
MEDIUMTEXT	$2^{24}-1$ **(т.е. 16777215)**	**Среднее поле TEXT**
LONGBLOB	$2^{32}-1$ **(т.е. 4294967295)**	**Большое поле BLOB**
LONGTEXT	$2^{32}-1$ **(т.е. 4294967295)**	**Большое поле TEXT**

Таблица 8.11 описывает типы ENUM и SET.

Таблица 8.11 Типы **ENUM** и **SET**

Тип	*Максимальное количество значений в наборе*	*Описание*
ENUM ('value1', 'value2',...)	**65535**	**Столбцы этого типа могут содержать только одно из перечисленных значений либо NULL.**
SET ('value1', 'value2',...)	**64**	**Столбцы этого типа могут содержать набор определенных значений либо NULL.**

Дополнительная информация

Дополнительные сведения о настройке баз данных находятся в интерактивном руководстве по MySQL, которое доступно по адресу **http://www.mysql.com/.**

Что дальше

Теперь уже ясно, как создавать пользователей, таблицы, базы данных, и, наконец, можно сосредоточиться на взаимодействии с базой данных. В следующей главе мы рассмотрим, как вносить данные в таблицы, обновлять и удалять их, и как отправлять запросы в базу данных.

Глава 9

Работа с базой данных MySQL

В этой главе будет рассматриваться язык структурированных запросов (SQL) и его применение для запросов в базы данных. На примере базы данных Book-O-Rama мы увидим, как вставлять, удалять и обновлять данные, и как задавать базе данных вопросы.

Темы, которые будут затронуты:

- Что такое SQL?
- Вставка данных в базу данных
- Извлечение данных из базы данных
- Объединение таблиц
- Обновление записей в базе данных
- Изменение таблиц после создания
- Удаление записей из базы данных
- Удаление таблиц

Начнем мы с того, что собой представляет SQL и чем он полезен.

Если вы еще не установили базу данных Book-O-Rama, мы рекомендуем установить ее, поскольку именно для этой базы данных будут формироваться запросы в данной главе. Инструкции по установке находятся в главе 8.

Что такое SQL?

SQL расшифровывается как *Structured Query Language (Язык структурированных запросов)*. Это наиболее стандартизированный язык для доступа к *системам управления реляционными базами данных (СУРБД)*. SQL используется для сохранения и извлечения данных в и из базы данных. Его применяют в таких системах баз данных как MySQL, Oracle, PostgreSQL, Sybase, а также Microsoft SQL Server.

Для SQL существует стандарт ANSI, и такие системы как MySQL используют этот стандарт. Конечно, не без того, чтобы добавить в них что-нибудь свое. Система привилегий MySQL — наглядный тому пример.

Наверняка доводилось слышать о *языках определения данных (Data Definition Languages, DDL),* которые используются для определения баз данных, и о *языках манипулирования данными (Data Manipulation Languages, DML),* применяемых для запросов баз данных. SQL обладает основами и тех, и других языков. В главе 8 мы наблюдали, как определяются данные в SQL (через DDL), так что это уже должно быть понятно. DDL используется в самом начале установки базы данных.

Аспект DML используется в SQL намного чаще, поскольку именно таким образом сохраняются и извлекаются реальные данные из баз данных.

Вставка данных в базу данных

Прежде чем всерьез приняться работать с базой данных, в ней необходимо сохранить какие-нибудь данные. Наиболее приемлемый способ предполагает использование оператора SQL **INSERT.**

Не забывайте, что СУРБД содержит таблицы, в которых, в свою очередь, находятся строки данных, организованные в столбцы. Каждая строка в таблице обычно описывает какой-либо реальный объект или отношение, а значения столбцов в этой строке хранят информацию о реальном объекте. Оператор **INSERT** можно использовать для внесения строк с данными в базу данных.

Типичная форма оператора **INSERT** выглядит так:

```
INSERT [INTO] table [(column1, column2, column3, ...)] VALUES
(value1, value2, value3, ...);
```

Например, чтобы вставить запись в таблицу Customers базы данных Book-O-Rama, можно набрать:

```
insert into customers values
   (NULL, "Julie Smith", "25 Oak Street", "Airport West");
```

Как видите, мы заменили ***table*** реальным именем таблицы, в которую требуется внести данные, а ***values*** — необходимыми значениями. Значения в данном примере заключены в двойные кавычки. В MySQL строки в любом случае должны быть помещены в пару одинарных или двойных кавычек. (В нашей книге встречаются оба варианта.) Числа и даты в кавычках не нуждаются.

С оператором **INSERT** связано несколько интересных моментов.

Указанные значения будут использованы для того, чтобы заполнять столбцы таблицы по порядку. Если необходимо заполнить только отдельные столбцы, или если вы хотите сами указать их порядок — можно поместить список столбцов в ту часть оператора, которая относится к столбцам. Например:

```
insert into customers (name, city) values
("Melissa Jones", "Nar Nar Goon North");
```

Такой подход полезен, когда о какой-либо записи есть лишь частичная информация или если несколько полей записи необязательны. Аналогичного эффекта можно достичь, прибегнув к следующему синтаксису:

```
insert into customers
set name="Michael Archer",
       address="12 Adderley Avenue",
       city="Leeton";
```

Вы, вероятно, заметите, что, добавляя Julie Smith, мы присвоили столбцу **customerid** нулевое значение (NULL), а при внесении других клиентов этот столбец просто игнорируется. Когда мы создавали базу данных, **customerid** был первичным ключом для таблицы Customers, так что это может показаться странным. В любом случае, поле было обозначено как **AUTOINCREMENT**. Это значит, что если вставить строку с нулевым значением или без значения, MySQL сгенерирует следующее число в автоинкрементной последовательности и вставит его автоматически. Это может пригодиться.

В таблицу можно также вставлять несколько строк сразу. Каждая строка должна быть заключена в скобки, разделенные запятыми.

Дабы объяснение было доходчивым, приведем несколько простых примеров. Это семейство операторов **INSERT** как раз иллюстрирует вышеизложенный подход мультистрочной вставки. Сценарий, выполняющий пример, находится на сопровождающем книгу CD-ROM, в файле **\chapter9\book_insert.sql**. Этот сценарий показан в листинге 9.1.

Листинг 9.1. book_insert.sql — **SQL-код для заполнения данными таблицу Book-O-Rama**

```
use books;

insert into customers values
   (NULL, "Julie Smith", "25 Oak Street", "Airport West"),
   (NULL, "Alan Wong", "1/47 Haines Avenue", "Box Hill"),
   (NULL, "Michelle Arthur", "357 North Road", "Yarraville");

insert into orders values
   (NULL, 3, 69.98, "02-Apr-2000"),
   (NULL, 1, 49.99, "15-Apr-2000"),
   (NULL, 2, 74.98, "19-Apr-2000"),
   (NULL, 3, 24.99, "01-May-2000");

insert into books values
   ("0-672-31697-8", "Michael Morgan", "Java 2 for Professional Developers",
       34.99),
   ("0-672-31745-1", "Thomas Down", "Installing Debian GNU/Linux", 24.99),
    ("0-672-31509-2", "Pruitt, et al.", "Sams Teach Yourself GIMP in
       24 Hours", 24.99),
    ("0-672-31769-9", "Thomas Schenk", "Caldera OpenLinux System
       Administration Unleashed", 49.99);

insert into order_items values
   (1, "0-672-31697-8", 2),
   (2, "0-672-31769-9", 1),
   (3, "0-672-31769-9", 1),
   (3, "0-672-31509-2", 1),
   (4, "0-672-31745-1", 3);

insert into book_reviews values
   ("0-672-31697-8", "Morgan's book is clearly written and goes well beyond
                      most of the basic Java books out there.");
```

Сценарий можно выполнить, запустив его через MySQL следующим образом:

```
>mysql -h host -u bookorama -p < book_insert.sql
```

Извлечение данных из базы данных

Рабочей лошадкой MySQL является оператор **SELECT.** Он извлекает данные из базы данных, выбирая из таблицы строки, которые отвечают заданному критерию поиска. Оператор **SELECT** сопровождает немалое количество опций и вариантов использования.

Основная же его форма такова:

```
SELECT items
FROM tables
[ WHERE condition ]
[ GROUP BY group_type ]
[ HAVING where_definition ]
[ ORDER BY order_type ]
[ LIMIT limit_criteria ] ;
```

Рассмотрим каждую конструкцию. Однако сперва давайте взглянем на поиск как таковой, без дополнительных опций, когда требуется просто найти определенные элементы в определенной таблице. Обычно это столбцы из таблиц. (Кроме того, они могут быть результатами любых выражений MySQL. Далее в этой главе будут рассматриваться наиболее употребимые.) Этот поиск запрашивает данные столбцов name (имя) и city (город) таблицы Customers:

```
select name, city
from customers;
```

Если введены данные из листинга 9.1, результат поиска будет таким:

```
+-------------------+----------------------+
| name              | city                 |
+-------------------+----------------------+
| Julie Smith       | Airport West         |
| Alan Nong         | Box Hill             |
| Michelle Arthur   | Yarraville           |
| Melissa Jones     | Nar Nar Goon North   |
| Michael Archer    | Leeton               |
+-------------------+----------------------+
```

Как видите, получена таблица с выбранными элементами — name и city — из указанной таблицы Customers. Эти данные собраны из всех подходящих строк таблицы Customers.

Можно указывать столько столбцов таблицы, сколько душе угодно, главное — вписать их по порядку после ключевого слова **select.** Кроме того, можно указывать и другие элементы. Очень неплохую помощь может оказать оператор группового символа *, который расширяет область запроса до уровня всех столбцов указанной таблицы (или таблиц). Например, чтобы извлечь все столбцы и строки из таблицы order_items, можно поступить так:

```
select *
from order_items;
```

вследствие чего получим вот такой ответ:

```
+----------+---------------+----------+
| orderid  | isbn          | quantity |
+----------+---------------+----------+
|        1 | 0-672-31697-8 |        2 |
|        2 | 0-672-31769-9 |        1 |
|        3 | 0-672-31769-9 |        1 |
|        3 | 0-672-31509-2 |        1 |
|        4 | 0-672-31745-1 |        3 |
+----------+---------------+----------+
```

Извлечение *данных по определенному критерию*

Чтобы получить доступ к подмножеству строк в таблице, следует указать критерий выбора. В этом нам поможет конструкция **where.** Например,

```
select *
from orders
where customerid = 3;
```

выбирает все столбцы из таблицы заказов, но только из строк с **customerid,** равным 3. Вот результат:

```
+----------+------------+---------+------------+
| orderid  | customerid | amount  | date       |
+----------+------------+---------+------------+
|        1 |          3 |   69.98 | 0000-00-00 |
|        4 |          3 |   24.99 | 0000-00-00 |
+----------+------------+---------+------------+
```

Конструкция **where** устанавливает критерий выбора определенных строк. В нашем случае выбраны строки с **customerid,** равным 3. Одиночный знак равенства используется для проверки на равенство — обратите внимание, что это немного отличается от PHP, и если работать и с тем, и с другим, вполне можно запутаться.

Вдобавок к равенству, MySQL поддерживает целое семейство операторов и регулярных выражений. Наиболее употребимые в конструкции **where** перечислены в табл. 9.1. Обратите внимание, что список далеко не полон, и если понадобится что-нибудь, отсутствующее в нем, обратитесь к руководству по MySQL.

Таблица 9.1 Полезные знаки операции сравнения для конструкции WHERE

Оператор	***Название*** *(если применимо)*	***Пример***	***Описание***
=	равенство	customerid=3	**Проверяет, являются ли два значения равными**
>	больше	amount>60.00	**Проверяет, больше ли одно значение другого**
<	меньше	amount<60.00	**Проверяет, меньше ли одно значение другого**
>=	больше или равно	amount>=60.00	**Проверяет, больше или равно одно значение по отношению к другому**
<=	**меньше или равно**	amount<=60.00	**Проверяет, меньше или равно одно значение по отношению к другому**
!= или О	**не равно**	**quantity !=0**	**Проверяет, не равны ли два значения**
IS NOT NULL	**адрес не равен нулю**		**Проверяет, имеет ли поле значение**
IS NULL	**адрес равен нулю**		**Проверяет, не имеет ли поле значения**
BETWEEN	**величина между 0 и 60.00**		**Проверяет, значение больше или равно минимальному и меньше или равно максимальному**
IN	**город содержится** ("Carlton", **"Moe")**		**Проверяет, содержится ли значение в определенном множестве**
NOT IN	**город не содержится ("Carlton", "Moe")**		**Проверяет, не содержится ли значение в определенном множестве**

Оператор	Название (если применимо)	Пример	Описание
LIKE	соответствие	name like ("Fred %")	Проверяет, отвечает ли значение образцу, используя простые механизмы соответствия SQL
NOT LIKE	соответствие	name not like ("Fred %")	Проверяет, не соответствует ли значение образцу
REGEXP	регулярное выражение	name regexp	Проверяет, соответствует ли значение регулярному выражению

Три последних строки таблицы относятся к **LIKE и REGEXP.** Это формы соответствия образцу.

LIKE использует простой механизм соответствия SQL. Образец может состоять из обычного текста плюс % (знак процента) для указания совпадения с любым количеством символов и _ (символ подчеркивания) для указания совпадения с одним символом. В MySQL соответствия не чувствительны к регистру. Например, **'Fred %'** найдет любое значение, которое начинается с **'fred '**.

Ключевое слово **REGEXP** используется для соответствия регулярных выражений. MySQL использует регулярные выражения в стиле POSIX. Вместо **REGEXP** можно применять и **RLIKE**, что является синонимом. Регулярные выражения **POSIX** также применяются и в PHP. Подробнее о них можно узнать в главе 4.

Можно проверять несколько критериев сразу, объединяя их операциями **AND** или **OR.** Например,

```
select *
from orders
where customerid = 3 or customer = 4 ;
```

Извлечение данных из нескольких таблиц

Часто для получения ответа от базы данных на заданный вопрос могут потребоваться данные нескольких таблиц. Например, если необходимо узнать, кто из клиентов осуществлял в этом месяце заказы, придется просмотреть таблицы Customers и Orders. Если нужно узнать, что конкретно они заказали, нельзя обойти вниманием и таблицу Order_Items.

Эти данные находятся в разных таблицах, поскольку относятся к разным реальным объектам. Это один из принципов хорошей разработки базы данных, и об этом упоминалось в главе 7.

Для объединения этой информации в SQL потребуется выполнить операцию, называемую *объединением*. Подразумевается объединение двух и больше таблиц с тем, чтобы сохранялись отношения между данными. Если, например, необходимо посмотреть, какие заказы сделал клиент Julie Smith, то сначала потребуется просмотреть таблицу Customers и найти в ней CustomerID Julie, после чего — таблицу Orders на предмет заказов, сделанных этим CustomerlD.

Хотя на первый взгляд операция объединения достаточно проста, на самом деле это один из наиболее сложных и тонких разделов SQL. В MySQL существует несколько разных типов объединения, каждый из которых предназначен для определенной цели.

Простое объединение двух таблиц

Начнем с поиска Julie Smith, которую мы уже не раз упоминали:

```
select orders.orderid, orders.amount, orders.date
from customers, orders
where customers.name = 'Julie Smith'
and customers.customerid = orders.customerid;
```

Результат запроса будет таким:

```
+----------+--------+------------+
| orderid  | amount | date       |
+----------+--------+------------+
|        2 |  49.99 | 0000-00-00 |
+----------+--------+------------+
```

Здесь стоит отметить несколько моментов.

Во-первых, для ответа на этот запрос необходима информация из двух таблиц, поэтому в списке перечислены обе.

Также мы определили тип объединения, возможно даже не зная его. Запятая между названиями таблиц эквивалентна словам **INNER JOIN** или **CROSS JOIN.** Такой тип соединения еще называют *полным объединением* или *Декартовым произведением* таблиц. Это означает: "Возьми указанные таблицы и сделай из них одну большую. В большой таблице должна быть строка для любой возможной комбинации строк из каждой таблицы, указанной в списке, имеют они смысл или нет". Другими словами, получаем таблицу, в которой каждая строка таблицы Customers сопоставляется каждой строке таблицы Orders независимо от того, какой клиент сделал какой заказ.

В большинстве случаев смысла в этом немного. Как правило, нам нужны строки, которые в самом деле совпадают, т.е. когда конкретные заказы совпадают с теми клиентами, которые их производили.

Это достигается путем помещения в конструкцию **WHERE** *условия объединения.* Это особый тип условного оператора, который объясняет, какие атрибуты показывают отношения между двумя таблицами. В данном случае наше условие соединения было таким:

```
customers.customerid = orders.customerid
```

что предписывает MySQL выводить в таблицу *с* результатами только соответствия Customerid из таблицы Customers CustomerID из таблицы Orders.

Внеся это условие в запрос, мы получили объединение другого типа — *объединение по равенству (equi-join).*

Обратите внимание на точечную нотацию, которой мы воспользовались для уточнения конкретного столбца конкретной таблицы. Так, **customers.customerid** относится к столбцу **customerid** из таблицы Customers, а **orders.customerid** — к столбцу **customerid** из таблицы Orders.

Точечная нотация необходима, когда имена столбцов неоднозначны, что случается, если одни и те же имена встречается в нескольких таблицах.

Как расширение его можно использовать для различения имен столбцов из разных баз данных. В нашем примере обозначение выглядит как table.column (таблица.столбец). Можно указать и иначе — database.table.column (база_данных.таблица.столбец), например, для проверки условия наподобие

```
books.orders.customerid = other_db.orders.customerid
```

С другой стороны, точечную нотацию можно применять и для всех ссылок на столбцы в запросе. Это частенько избавляет от лишней головной боли, особенно когда запросы становятся все более сложными. MySQL этого не требует, но удобочитабельные запросы — это не так уж и плохо. Если вы заметили, мы придерживаемся этого принципа во всех наших примерах, взять хотя бы вот такое условие

```
customers.name = 'Julie Smith'
```

Столбец **name** присутствует только в таблице customers, поэтому его необязательно указывать, но так, в общем-то, понятнее.

Объединение трех и более таблиц

Объединение более двух таблиц не сложнее объединения двух. Главное правило таково — таблицы нужно объединять попарно, учитывая условия объединения. Это можно представить в виде отношений данных между первой таблицей, второй и третьей.

Например, если требуется узнать, кто из клиентов заказал книги по Java (возможно, с целью того, чтобы отправить им информацию о новой книге по Java), необходимо отследить эти отношения в рамках небольшого количества таблиц.

Необходимо будет найти клиентов, разместивших, по крайней мере, один заказ, который выражен в **order_items** книгой по Java. Из таблицы Customers перебираемся в таблицу Orders, используя **customerid**, как и в предыдущих случаях. Из таблицы Orders в таблицу Order_Items, используя **orderid.** Из Order_Items — в таблицу Books за нужной книгой, руководствуясь номером ISBN. После того как все связи установлены, можем запросить книги со словом Java в названии и получить в результате имена клиентов, которые купили какую-либо из этих книг.

Посмотрим на запрос, который приведет все это в исполнение:

```
select customers.name
from customers, orders, order_items, books
where customers.customerid = orders.customerid
and orders.orderid = order_items.orderid
and order_items.isbn = books.isbn
and books.title like '%Java%' ;
```

Этот запрос выдаст следующий результат:

```
+-------------------+
| name              |
+-------------------+
| Michelle Arthur   |
+-------------------+
```

Обратите внимание, что были отслежены данные из четырех разных таблиц, а чтобы сделать это с помощью объединения по равенству, понадобились три разных условия объединения. Обычно каждой паре таблиц требуется одно условие объединения, таким образом, количество условий объединения на единицу меньше количества объединяемых таблиц. Это правило большого пальца может пригодиться при отладке запросов, которые работают неустойчиво. Проверьте свои условия объединения и убедитесь в том, что вы все время следовали намеченному пути, от того, что вы *уже* знаете — к тому, что *нужно* узнать.

Поиск несоответствующих строк

Другой распространенный тип соединения в MySQL — объединение по остатку.

В предыдущих примерах отбирались только те строки, в которых наблюдалось соответствие между таблицами. Однако могут потребоваться и строки, в которых нет

соответствия — например, нужно найти клиентов, которые не сделали ни одного заказа, или книги, которые никто не заказывал.

Самый простой вариант ответа на такой вопрос в MySQL — использование объединения по остатку, которое будет искать строки по указанному условию объединения двух таблиц. Если в указанной таблице нет подходящей строки, эта строка добавляется к результату, но с нулевым значением.

Взглянем на пример:

```
select customers.customerid, customers.name, orders.orderid
from customers left join orders
on customers.customerid = orders.customerid;
```

Данный запрос SQL использует объединение по остатку для таблиц Customers и Orders. Его синтаксис в отношении условий объединения несколько иной; условие объединения указывается в специальной конструкции **ON** оператора SQL.

Вот результат запроса:

```
+------------+------------------+-----------+
| customerid | name             | orderid   |
+------------+------------------+-----------+
|          1 | Julie Smith      |         2 |
|          2 | Alan Hong        |         3 |
|          3 | Michelle Arthur  |         1 |
|          3 | Michelle Arthur  |         4 |
|          4 | Melissa Jones    |      NULL |
|          5 | Michael Archer   |      NULL |
+------------+------------------+-----------+
```

Результат показывает, что для клиентов Melissa Jones и Michael Archer нет соответствующих **orderid,** поскольку их **orderid** имеют значения **NULL.**

Если необходимо найти исключительно тех клиентов, которые ничего не заказывали, этого можно достичь, проверив их на значение NULL в поле первичного ключа правой таблицы (в данном случае, **orderid),** поскольку строки с реальными значениями не могут иметь значение **NULL:**

```
select customers.customerid, customers.name
from customers left join orders
using  (customerid)
where orders.orderid is null;
```

И вот результат:

```
+------------+-----------------+
| customerid | name            |
+------------+-----------------+
|          4 | Melissa Jones   |
|          5 | Michael Archer  |
+------------+-----------------+
```

Вероятно, вы обратили внимание на то, что в этом примере условие объединения обладает несколько другим синтаксисом. Объединение по остатку воспринимает как синтаксис ON, как было в первом примере, так и **USING,** как было во втором. Синтаксис **USING** не предполагает указания таблицы атрибута объединения, и по этой причине, если вы хотите пользоваться таким синтаксисом, столбцы в обеих таблицах должны называться одинаково.

Использование других имен таблиц: псевдонимы

Часто бывает очень удобно, а порой и необходимо обращаться к таблицам под другими именами. Такие имена называются *псевдонимами (aliase)*. Их можно создать в самом начале запроса, а потом пользоваться по мере необходимости. Псевдонимы очень удобны, все равно, что ярлык на рабочем столе. Взгляните, как выглядит достаточно объемный запрос, рассмотренный нами ранее, переписанный с использованием псевдонимов:

```
select c.name
from customers as c, orders as o, order_items as oi, books as b
where c.customerid = o.customerid
and o.orderid = oi.orderid
and oi.isbn = b.isbn
and b.title like '%Java%' ;
```

Указывая таблицы, которые мы собираемся использовать, мы вставляем конструкцию **as,** присваивая таблице псевдоним. Кроме того, псевдонимы можно давать столбцам, однако к этому мы вернемся после того, как рассмотрим обобщенные функции.

Табличные псевдонимы необходимы в случае объединения таблицы с самой собой. Это звучит гораздо более угрожающе и мистически, чем есть на самом деле. А такой подход очень удобен для поиска строк в той же таблице, в которой есть и другие нужные нам значения. Если требуется найти клиентов, живущих в одном городе, скажем, с целью создания читательского клуба, можно присвоить одной и той же таблице (Customers) два разных псевдонима:

```
select c1.name, c2.name, c1.city
from customers as c1, customers as c2
where c1.city = c2.city
and c1.name != c2.name
```

Мы делаем вид, что таблица Customers — это на самом деле две разные таблицы, cl и c2, и выполняем объединение по столбцу City. Второе условие, **cl.name != c2.name**, необходимо для того, чтобы в результате запроса не выдавалось соответствие клиента самому себе.

Резюме по типам объединений

В табл. 9.2 перечислены разные типы объединений. Мы рассмотрели только самые основные, хотя существуют еще несколько объединений.

Таблица 9.2 Типы объединений в MySQL

Название	*Описание*
Декартово произведение	Все комбинации всех строк во всех таблицах. В случае применения между именами таблиц ставят запятые и не употребляют конструкцию **WHERE.**
Полное объединение	Аналогично предыдущему.
Перекрестное объединение	Аналогично предыдущему. Также может использоваться с указанием ключевых слов **CROSS JOIN** между названиями объединяемых таблиц.
Внутреннее объединение	Семантически эквивалентно запятой. Может использоваться с указанием ключевых слов **INNER JOIN.** Без условия **WHERE** эквивалентно полному объединению. Обычно при истинно внутреннем объединении задается условие **WHERE.**

Название	*Описание*
Объединение по равенству	Использует условное выражение со знаком = для соответствия в объединении строк из разных таблиц. В SQL в этом объединении применяется конструкция **WHERE.**
Объединение по остатку	Старается уравнивать строки в таблицах и выискивает несовпадающие строки со значениями **NULL.** В SQL используется с ключевыми словами **LEFT JOIN.** Предназначено для поиска отсутствующих значений. Аналогично можно употреблять **RIGHT JOIN.**

Извлечение данных в определенном порядке

Если строки, извлеченные по запросу, должны перечисляться в определенном порядке, для этого используется конструкция **ORDER BY** оператора SELECT. Эта особенность удобна для представления результатов запроса в удобочитабельном формате.

Конструкция **ORDER BY** применяется для сортировки строк в столбцах, указанных в конструкции **SELECT.** Например,

```
select name, address
from customers
order by name;
```

Такой запрос представит имена и адреса клиентов в алфавитном порядке по именам:

```
+------------------+---------------------+
| name             | address             |
+------------------+---------------------+
| Alan Hong        | 1/47 Haines Avenue  |
| Julie Smith      | 25 Oak Street       |
| Melissa Jones    |                     |
| Michael Archer   | 12 Adderley Avenue  |
| Michelle Arthur  | 357 North Road      |
+------------------+---------------------+
```

(Обратите внимание, что в таком случае, поскольку имена состоят из имени и фамилии, отсортированы они будут по имени. Если вы хотите сортировки по фамилии (которая стоит второй), нужно, чтобы имя и фамилия хранились в двух разных полях.)

По умолчанию порядок сортировки идет по возрастанию (от a до z или по возрастанию числовых значений). Это можно указать ключевым словом **ASC** (от англ. ascending):

```
select name, address
from customers
order by name asc;
```

Изменить порядок сортировки на обратный можно с помощью другого ключевого слова — **DESC** (от англ. descending):

```
select name, address
from customers
order by name desc;
```

Сортировать можно и по нескольким столбцам. Вместо названий можно использовать псевдонимы столбцов, и даже их порядковые номера (например, 3 — для третьего столбца в таблице).

Группировка и агрегирование данных

Нередко хочется знать, сколько строк присутствует в определенном наборе, или каково среднее значение какого-нибудь столбца — скажем, средняя цена в долларах каждого заказа. В MySQL имеется набор функций агрегирования, которые неплохо подходят для выполнения задач подобного рода.

Эти функции агрегирования можно применять как для таблицы в целом, так и для групп данных внутри таблицы.

Наиболее часто используемые функции перечислены в табл. 9.3.

Таблица 9.3 Обобщенные функции в MySQL

Название	Описание
AVG (столбец)	Средняя величина значений в определенном столбце.
COUNT (элементы)	При указании столбца выдается количество числовых (ненулевых) значений в этом столбце. Если перед названием столбца вставить слово **DISTINCT,** выдается только количество конкретных значений в столбце. Если указать **COUNT** (*) — подсчет строк будет производиться независимо от нулевых значений.
MIN (столбец)	Минимальное значение в столбце.
MAX (столбец)	Максимальное значение в столбце.
STD (столбец)	Стандартное отклонение значений в столбце.
STDDEV (столбец)	Аналогично предыдущему.
SUM (столбец)	Сумма значений в столбце.

Взглянем на несколько примеров, начиная с уже рассмотренного. Среднюю величину всех заказов можно высчитать так:

```
select avg (amount)
from orders;
```

Результат будет приблизительно таким:

```
avg (amount)
54.985002
```

Чтобы получить более детальную информацию, можно воспользоваться конструкцией **GROUP BY.** Это позволит посмотреть среднюю величину заказа по группам, например, по номеру клиента, что даст информацию о том, кто из клиентов делает самые крупные заказы:

```
select customerid, avg (amount)
from orders
group by customerid;
```

При использовании конструкции **GROUP BY** с функцией агрегирования это фактически меняет поведение функции. Вместо того чтобы выдавать среднюю величину заказов в таблице, такой запрос даст информацию по средней величине заказа каждого клиента (а если точнее, каждого **customerid)**

```
+------------+-------------+
I customerid | avg(amount) |
+------------+-------------+
I          1 I   49.990002 |
!          2 |   74.980003 |
I          3 |   47.485002 |
+------------+-------------+
```

Единственное, что стоит отметить при использовании функций группировки и агрегирования: если используется функция агрегирования или конструкция GROUP BY в ANSI SQL, в конструкции **SELECT** будут присутствовать только функции агрегирования и столбцы, указанные в конструкции **GROUP BY.** Если требуется использовать столбец в конструкции **GROUP BY,** он должен быть указан в конструкции **SELECT.**

На самом деле MySQL обеспечивает гораздо большую свободу действий, поддерживая *расширенный синтаксис,* который дает возможность убирать ненужные элементы из конструкции **SELECT.**

Вдобавок к группировке и агрегированию данных есть все шансы проверить результат агрегирования с использованием конструкции **HAVING.** Она следует сразу после конструкции **GROUP BY** и похожа на **WHERE,** но только применяется к группам и множествам.

Чтобы расширить предыдущий пример, скажем, получением информации, кто из клиентов произвел заказ в среднем больше чем на $50, можем воспользоваться следующим запросом:

```
select customerid, avg (amount)
from orders
group by customerid
having avg (amount) > 50;
```

Заметьте, конструкция **HAVING** обращается к группам. Такой запрос выдаст следующий результат:

```
+------------+--------------+
I customerid | avg(amount)  |
+------------+--------------+
|          2 |   74.980003  |
+------------+--------------+
```

Выбор количества отображаемый строк

Одна конструкция оператора **SELECT,** которая может оказаться особенно полезной в Web-приложениях — это конструкция **LIMIT.** Ее используют для указания, сколько строк результата следует отображать. Необходимы два параметра: номер строки, с которой следует начать, и количество строк.

Следующий запрос иллюстрирует применение **LIMIT:**

```
select name
from customers
limit 2, 3;
```

Запрос можно интерпретировать так: "Выбрать имена среди клиентов, в результате отобразить три строки, начиная со строки 2". Не забывайте, что нумерация строк начинается с нуля.

Это очень удобная конструкция для Web-приложений. Ее принцип точно такой же, как и в случае, когда покупатель листает каталог и хочет видеть на одной странице только 10 пунктов.

Обновление записей в базе данных

Помимо того, что данные необходимо извлекать из базы данных, их еще необходимо периодически обновлять. Например, иногда требуется повысить цены на книги в базе данных. Это можно сделать, используя оператор **UPDATE.**

Типовая форма этого оператора выглядит так:

```
UPDATE tablename
SET column1=expression1, column2=expression2,
[WHERE condition]
[LIMIT number]
```

Основная идея заключается в обновлении таблицы с именем ***tablename,*** изменяя каждый указанный столбец ***column*** соответствующим выражением ***expression.*** Конструкцией **WHERE UPDATE** можно ограничить до работы с определенными строками, а конструкцией **LIMIT** обозначить количество строк, которые нужно обновить.

Давайте рассмотрим несколько примеров.

Если мы хотим повысить цену всех книг на 10%, можно воспользоваться оператором **UPDATE** без конструкции WHERE:

```
update books
set price=price*1.1;
```

Если же требуется изменить одну строку, скажем, адрес некоторого клиента, можно поступить таким образом:

```
update customers
set address = '250 Olsens Road'
where customerid = 4;
```

Изменение таблиц после создания

Кроме обновления данных в строках, может потребоваться изменить структуру самой таблицы в базе данных. Для этого применяется очень гибкий оператор **ALTER TABLE.** Базовая его форма такова:

```
ALTER TABLE tablename alteration [, alteration ...]
```

В ANSI SQL один оператор **ALTER TABLE** может осуществить только одно преобразование, а вот в MySQL подобных ограничений нет. Разные конструкции преобразования могут изменять разные аспекты таблицы.

Типы преобразования, осуществляемые оператором **ALTER TABLE,** перечислены в табл. 9.4.

Таблица 9.4 Возможные преобразования в операторе ALTER TABLE

Синтаксис	*Описание*
ADD [COLUMN] *column_description* **[FIRST | AFTER** *column]*	**Добавить новый столбец в указанное место (если место не указано, столбец добавляется в конец). Обратите внимание,** *column_description* **требует имени и типа, точно так же, как при работе с оператором CREATE.**
ADD [COLUMN] (*column_description, column_description, ...)*	**Добавить один или несколько столбцов в конец таблицы.**
ADD INDEX [index] *(column, ...)*	**Добавить индекс в указанный столбец (столбцы) таблицы.**
ADD PRIMARY KEY *(column, ...)*	**Сделать указанный столбец (столбцы) первичным ключом таблицы.**
ADD UNIQUE [index] *(column, ...)*	**Добавить уникальный индекс в указанный столбец (столбцы) таблицы.**
ALTER [COLUMN] *column* {SET **DEFAULT** *value* \ **DROP** DEFAULT}	**Добавить или удалить значение по умолчанию определенного столбца.**
CHANGE [COLUMN] *column new_column_description*	**Изменить столбец с именем** *column* **так, чтобы он получил указанное описание. Это можно использовать для изменения имени столбца, поскольку** *column_description* **включает в себя имя.**
MODIFY [COLUMN] *column_description*	**Похоже на CHANGE. Используется для изменения типов столбцов, но не имен.**
DROP [COLUMN] *column*	**Удалить указанный столбец.**
DROP PRIMARY KEY	**Удалить первичный индекс (не столбец!).**
DROP INDEX *index*	**Удалить указанный индекс.**
RENAME[AS] *new_table_name*	**Переименовать таблицу.**

Рассмотрим наиболее типичные случаи употребления **ALTER TABLE.**

Частенько случается так, что вы вдруг осознаете: какой-то столбец "недостаточно велик", чтобы вместить в себе необходимые данные. Например, в нашей таблице Customers имена и фамилии могут иметь длину до 30 символов. И вскоре может оказаться, что некоторые имена и фамилии слишком длинны и сохраняются в таблице в искаженном виде. Однако это можно исправить, изменив тип данных столбца, после чего он сможет принимать имена и фамилии длиной до 45 символов.

```
alter table customers
modify name char(45) not null;
```

Другая часто встречающаяся необходимость связана с добавлением столбца. Представьте, что в каждом регионе существует свой налог с продаж, поэтому магазину Book-O-Rama приходится учитывать этот налог, но делать это раздельно, поскольку он везде разный. В таблицу Orders потребуется добавить столбец налога:

```
alter table orders
add tax float(6,2) after amount;
```

Иногда какой-нибудь столбец может оказаться лишним. Удалим столбец, который мы только что добавили:

```
alter table orders
drop tax;
```

Удаление записей из базы данных

Удалять строки из базы данных очень просто. При этом можно воспользоваться оператором **DELETE,** который выглядит следующим образом:

```
DELETE FROM table
[WHERE condition] [LIMIT number]
```

Если просто записать

```
DELETE FROM table;
```

то удаляются все строки в таблице, так что будьте осторожны! Обычно ненужными оказываются какие-то определенные строки и они указываются при помощи конструкции **WHERE.** Это может случиться, если какая-то книга больше не продается или кто-либо из клиентов длительное время ничего не заказывает:

```
delete from customers
where customerid=5;
```

Конструкция **LIMIT** ограничивает количество удаляемых строк.

Удаление таблиц

Временами нужно избавиться от целой таблицы. В этом хорошую службу сослужит оператор **DROP TABLE.** Он очень прост:

```
DROP TABLE table;
```

Он удаляет все строки из таблицы и вместе с ними саму таблицу, так что будьте с ним поосторожней.

Удаление целой базы данных

Можно пойти еще дальше и удалить целую базу данных! Вот вам соответствующий оператор - **DROP DATABASE:**

```
DROP DATABASE database;
```

В результате удаляются все строки, таблицы, индексы и сама база данных. Не стоит и говорить, что с этим оператором нужно вести себя, как на минном поле.

Дополнительная информация

В этой главе был рассмотрен обычный SQL, с которым приходится иметь дело при работе с базами данных MySQL. В последующих двух главах мы расскажем, как соединить MySQL с PHP, чтобы получать доступ к базе данных из Web. Кроме этого, будут рассмотрены некоторые дополнительные технологии применения MySQL.

Если хотите получить дополнительную информацию по SQL, обратитесь к истокам, почитайте о стандартах ANSI SQL, обратившись по адресу:

```
http://www.ansi.org/
```

Дополнительные возможности MySQL по сравнению с ANSI SQL описаны на Web-сайте MySQL:

```
http://www.mysql.com/
```

Что дальше

В главе 10 будет показано, как получать доступ к базе данных Book-O-Rama через Web.

Глава 10

Доступ к базе данных MySQL из Web с помощью PHP

В нашей предыдущей работе с PHP для хранения и извлечения данных мы использовали двумерные файлы. Когда данный вопрос рассматривался в главе 2, говорилось о том, что реляционные базы данных намного проще, безопаснее и эффективнее выполняют операции хранения и извлечения в Web-приложении. Теперь, поработав с MySQL над созданием базы данных, можно приступить к подключению этой базы данных к внешнему Web-интерфейсу.

В этой главе будет показано, как получить доступ к базе данных Book-O-Rama из Web, используя PHP. Вы узнаете, как читать базу и вносить в нее данные, и как фильтровать потенциально опасные входные данные.

В главе рассматриваются следующие вопросы:

- Как работает архитектура Web-баз данных
- Основные шаги выполнения запросов к базе данных через Web
- Установка соединения
- Получение информации о доступных базах данных
- Выбор базы данных
- Выполнение запроса к базе данных
- Получение результатов запроса
- Отсоединение от базы данных
- Внесение новой информации в базу данных
- Обеспечение безопасности базы данных
- Прочие полезные функции PHP-MySQL
- Другие интерфейсы PHP-баз данных

Как работает архитектура Web-баз данных

В главе 7 мы в общих чертах выяснили, как работает архитектура Web-баз данных. Еще раз напомним эти шаги:

1. Web-браузер пользователя выдает HTTP-запрос определенной Web-страницы. Например, пользователь ищет в Book-O-Rama все книги, написанные Майклом Морганом, используя HTML-форму. Страница с результатами поиска будет называться results.php.
2. Web-сервер принимает запрос на results.php, извлекает этот файл и передает на обработку механизму PHP.
3. Механизм PHP приступает к разбору сценария. Сценарий содержит команду соединения с базой данных и выполнения запроса (поиска книг). PHP открывает соединение с MySQL-сервером и отправляет соответствующий запрос.
4. Сервер MySQL принимает запрос к базе данных, обрабатывает его и отправляет результат (список книг) обратно механизму PHP.
5. Механизм PHP завершает выполнение сценария, что обычно включает в себя форматирование результатов запроса в HTML. После этого результат в виде HTML возвращается Web-серверу.
6. Web-сервер пересылает HTML в браузер, и пользователь имеет возможность просмотреть запрошенный список книг.

У нас имеется база данных MySQL, поэтому можем подготовить код PHP, чтобы выполнялись предыдущие шаги. Начнем с поисковой формы. Это простая HTML-форма. Код для этой формы показан в листинге 10.1.

Листинг 10.1 search.html — поисковая страница для базы данных Book-O-Rama

```
<html>
<head>
   <title>Book-O-Rama Catalog Search</title>
</head>

<body>
   <hl>Book-0-Rama  Catalog  Search</hl>

   <form action="results.php" method="post">
      Choose Search Type:<br>
      <select name="searchtype">
         <option value="author">Author
         <option value="title">Title
         <option value="isbn">ISBN
      </select>
      <br>
      Enter Search Term:<br>
      <input name="searchterm" type=text>
      <br>
      <input type=submit value="Search">
   </form>

</body>
</html>
```

Действительно, это достаточно простая HTML-форма. На рис. 10.1 показан ее внешний вид.

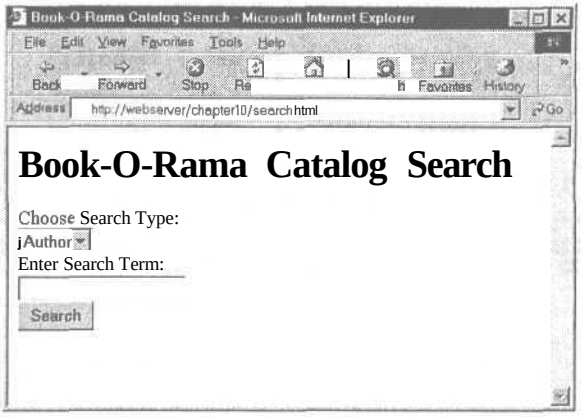


РИСУНОК 10.1
Поисковая форма предельно обобщена, поэтому книгу можно искать по названию, автору или номеру ISBN.

После того как пользователь нажмет на кнопке Search, вызывается сценарий results.php. Все это представлено в листинге 10.2. В данной главе мы рассмотрим, что делает упомянутый сценарий и как он работает.

Листинг 10.2 results.php - извлекает результаты запроса в базе данных MySQL и форматирует их для целей отображения

```
<html>
<head>
   <title>Book-O-Rama Search Results</title>
</head>
<body>
<h1>Book-O-Rama Search Results</h1>
<?
   trim($searchterm);
   if (!$searchtype || !$searchterm)
   {
        echo "You have not entered search details.  Please go back and
            ➥try again.";
        exit;
   }

   $searchtype = addslashes ($searchtype);
   $searchterm = addslashes ($searchterm);

   @ $db = mysql_pconnect("localhost", "bookorama", "bookorama");

   if (!$db)
   {
       echo "Error: Could not connect to database.  Please try again later.";
       exit;
   }

   mysql_select_db("books");
   $query = "select * from books where " . $searchtype." like
       '%".$searchterm."%'";
   $result = mysql_query($query);
   $num_results = mysql_num_rows ($result);

   echo "<p>Number of books found: ".$num_results."</p>";
```

```
    for ($i=0; $i <$num_results; $i++)
    {
        $row = mysql_fetch_array($result);
        echo "<p><strong>". ($i+1) .". Title: ";
        echo htmlspecialchars( stripslashes($row["title"]));
        echo "</strong><br>Author: ";
        echo htmlspecialchars (stripslashes($row["author"]));
        echo "<br>ISBN: ";
        echo htmlspecialchars (stripslashes($row["isbn"] ));
        echo "<br>Price: " ;
        echo htmlspecialchars (stripslashes($row["price"]));
        echo "</p>";
    }

?>

</body>
</html>
```

Рисунок 10.2 иллюстрирует результаты использования этого сценария для выполнения поиска.

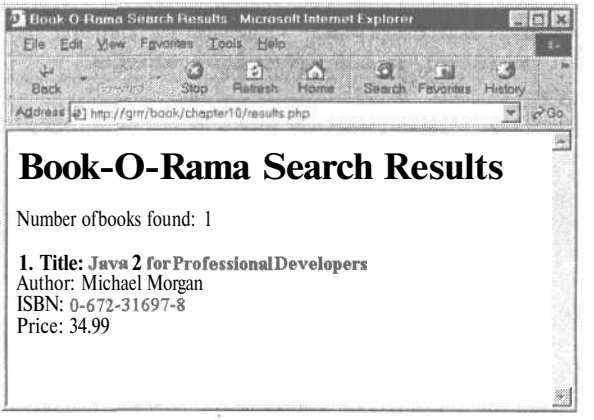


РИСУНОК 10.2
Результаты поиска в базе данных книг по Java с использованием сценария results.php представлены на Web-странице.

Основные шаги выполнения запросов к базе данных через Web

В любом сценарии, который обеспечивает доступ к базе данных из Web, имеется несколько базовых шагов:

1. Проверка и фильтрация данных, исходящих от пользователя.
2. Установка соединения с требуемой базой данных.
3. Передача запроса в базу данных.
4. Получение результатов.
5. Представление результатов пользователю.

То же самое делает и сценарий results.php, и сейчас мы исследуем каждый из этих этапов.

Проверка и фильтрация данных, исходящих от пользователя

Сначала необходимо убрать все лишние пробелы по краям слова, которые мог случайно набрать пользователь. Справиться с этим поможет функция **trim()**, применяемая к **$searchterm** (критерий поиска).

```
trim($searchterm);
```

Следующий этап — убедиться, что пользователь указал критерий и тип поиска. Заметьте, это проверяется лишь тогда, когда критерий поиска не содержит лишние пробелы. Если поменять эти этапы местами, может возникнуть ситуация, когда критерий вроде и введен, сообщения об ошибке быть не должно, однако критерий содержит только пробелы, которые полностью удаляются функцией **trim()**:

```
if (!$searchtype || !$searchterm)
{
   echo "You have not entered search details. Please go back and
        ➥try again." ;
   exit;
}
```

В этом случае выдается сообщение о том, что критерий поиска не введен.

Мы проверили переменную **$searchtype** даже в том случае, когда она поступает из оператора SELECT. Вас может заинтересовать, зачем проверять входные данные. Не забывайте, что база данных может иметь не один интерфейс. Например, Amazon располагает большим количеством филиалов, которые используют свои поисковые интерфейсы. Вследствие того, что пользователи могут заходить с разных рабочих станций, возникает и потенциальная угроза безопасности.

В случае задействования данных, которые вводят другие пользователи, необходимо тщательно фильтровать вводимые данные от управляющих символов. Как вы, наверное, помните, в главе 4 говорилось о функциях **addslashes()** и **stripslashes()**. Если записывать данные, введенные пользователем, в базу данных типа MySQL, следует вызывать **addslashes()**, а при возврате пользователю выходных данных — **stripslashes()**.

В нашем случае к критерию поиска применяется функция addslashes():

```
$searchterm = addslashes ($searchterm);
```

К данным, исходящим из базы, применяется stripslashesQ. Введенные данные не содержат косых линий, равно как и управляющих символов. То есть вызов **stripslashes()** не поможет. При построении Web-интерфейса для базы данных велики шансы того, что потребуется вносить данные о новых книгах, а детали, внесенные пользователем, могут содержать специальные символы. Сохраняя их в базе данных, мы обращаемся к addslashes(), а это означает, что при извлечении данных необходимо будет вызывать **stripslashes()**. Обратите внимание, что это хорошая привычка.

Функцию **htmlspecialchars()** применяют для кодировки символов, которые в HTML имеют особое значение. В наших тестовых данных нет амперсандов (&), знаков "меньше" (<), "больше" (>), двойных кавычек ("), однако в названиях многих замечательных книг может повстречаться амперсанд. Использование этой функции застраховывает от грядущих ошибок.

Установка соединения

Для подключения к серверу MySQL в сценарии присутствует такая строка:

```
@ $db = mysql_pconnect("localhost", "bookorama", "bookorama");
```

Для подключения к базе данных используется функция **mysql_pconnect()** со следующим прототипом:

```
int mysql_pconnect( [string host [:port] [:/socketpath] ] ,
                    [string user] , [string password] );
```

Потребуется указать имя узла *(host),* на котором размещен сервер MySQL, имя пользователя *(user),* чтобы войти в него, и пароль *(password).* Все это в принципе необязательно и если не указать все вышеперечисленное, функция воспользуется значениями по умолчанию — локальная машина вместо узла, имя пользователя, под которым запущен PHP, и пустой пароль.

В случае успеха функция вернет идентификатор связи с базой данных (который следует сохранить для дальнейшего использования), а в случае неудачи — значение false. Результат не стоит игнорировать, поскольку без соединения с базой данных работа невозможна. Это делает Следующий код:

```
if (!$db)
{
   echo "Error: Could not connect to database. Please try again later.";
   exit;
}
```

Как альтернативу, можно использовать другую функцию, которая делает практически то же самое — **mysql_connect().** Единственное отличие состоит в том, что **mysql_connect()** устанавливает *постоянное* соединение с базой данных.

Обычное соединение с базой данных закрывается, когда сценарий завершает свое выполнение или когда обращается к функции **mysql_close().** Постоянное соединение остается открытым и после того, как сценарий выполнен, а функцией **mysql_close()** его закрыть нельзя.

Может возникнуть вопрос, для чего это нужно. Ответ таков: соединение с базой данных предполагает некоторые непроизводительные затраты, что требует времени. Когда вызывается **mysql_pconnect()**, прежде чем она попытается подключиться к базе данных, она автоматически проверит, нет ли уже открытого постоянного соединения. Если есть, она не станет открывать новое. Это и время экономит, и предотвращает перегрузку сервера.

Однако если PHP выполняется как CGI, то постоянное соединение окажется не таким уж и постоянным. (Каждый вызов сценария PHP запускает новую копию механизма PHP и закрывает ее, когда сценарий завершает свою работу. Это, в свою очередь, также закрывает любое постоянное соединение.)

Помните, что количество соединений в MySQL, которые существуют одновременно, ограничено. Границу устанавливает параметр **max_connections.** Его задача (как и родственного ему параметра Apache **MaxClients)** — заставить сервер отвергать новые запросы на соединение, когда ресурсы узла заняты или когда программное обеспечение не функционирует.

Значения этих параметров можно изменять, редактируя файл конфигурации. Чтобы настроить MaxClients в Apache, следует править файл httpd.conf. Настройка **max_connections** в MySQL осуществляется за счет редактирования файла my.conf.

Если вы пользуетесь постоянными соединениями, и практически каждой странице на вашем сайте требуется доступ к базе данных, вам, пожалуй, понадобится постоянное соединение для каждого процесса Apache. Если же используются значения параметров, принятые по умолчанию, могут возникнуть определенные сложности. По

умолчанию Apache допускает до 150 соединений, а MySQL — только 100. В особо напряженное время соединений может не хватить. Поэтому лучше всего настроить параметры так, чтобы у каждого процесса Web-сервера было свое соединение, конечно, с оглядкой на технические возможности применяемых аппаратных средств.

Выбор базы данных

Работая с MySQL из командной строки, необходимо указывать, какая база данных нужна:

```
use books;
```

То же самое необходимо и при подключении из Web. Это может сделать PHP-функция **mysql_select_db()**:

```
mysql_select_db("books") ;
```

Прототип этой функции выглядит так:

```
int mysql_select_db(string database, [int database_connection] ) ;
```

В результате будет использоваться база данных с именем ***database.*** Можно также использовать соединение с базой данных, для которого требуется выполнить эту операцию (в нашем случае **$db),** однако, если его не указать, будет использоваться последнее открытое соединение. Если открытое соединение не существует, оно открывается по умолчанию, как если бы вызывалась **mysql_connect().**

Выполнение запроса к базе данных

Чтобы осуществить запрос, можно воспользоваться функцией **mysql_query()**. Однако прежде запрос необходимо настроить:

```
$query = "select * from books where ".$searchtype." like
    '%".$searchterm."%'";
```

В этом случае будет отыскиваться значение, введенное пользователем (**$searchterm**), в поле, которое указал пользователь (**$searchtype**). Вы, наверное, обратили внимание на то, что мы употребили **like,** отдав ему предпочтение перед **equal** — толерантность никогда не бывает излишней.



ПОДСКАЗКА

Не забывайте, что *запрос, отправляемый вами в MySQL, не требует в конце точки с запятой, в* отличие от запроса, который вводится в среде монитора MySQL.

Теперь можно выполнить запрос:

```
$result = mysql_query ($query) ;
```

Прототип функции **mysql_query()** таков:

```
int mysql_query(string query, [int database_connection] );
```

В функцию передается запрос, который должен быть выполнен; можно также передать еще и соединение с базой данных (в нашем случае **$db).** Если его не указать, будет использоваться последнее открытое соединение. Если такового нет, функция откроет соединение точно так же, как при выполнении **mysql_connect()**.

Возможно, вместо этого захочется воспользоваться функцией **mysql_db_query()**. Рассмотрим ее прототип:

```
int mysql_db_query(string database, string query,
   [int database_connection] ) ;
```

Почти то же самое, только здесь можно указать базу данных, в которой требуется производить поиск. В каком-то смысле это комбинация функций **mysql_select_db()** и **mysql_query()**.

Обе функции возвращают идентификатор результата (что позволяет получить результаты поиска) в случае успеха и значение false в случае неудачи. Идентификатор результата следует сохранить (так же, как в нашем случае с **$result),** чтобы извлечь из него некоторую пользу.

Получение результатов запроса

Разнообразие функций дает возможность получить результат различными способами. Идентификатор результата — это ключ доступа к строкам, возвращенным запросом, которых может быть нуль, одна и более.

В нашем примере использовались две функции: **mysql_numrows()** и **mysql_fetch_array()**.

Функция **mysql_numrows()** сообщает количество строк, которые возвращает запрос. В нее следует передать идентификатор результата:

```
$num_results = mysql_num_rows($result) ;
```

Это полезно знать, если планируется обрабатывать или отображать результаты. Зная их количество, можно организовать цикл:

```
for ($i=0; $i <num_results; $i++)
{
        // обработка результатов
}
```

На каждой итерации цикла происходит вызов **mysql_fetch_array().** Цикл не будет выполняться, если нет строк. Эта функция берет каждую строку из списка результата и возвращает ее в виде ассоциативного массива, с ключом как именем атрибутива и значением как соответствующим значением массива:

```
$row = mysql_fetch_array($result) ;
```

Имея **$row** в ассоциативном массиве, можно пройти каждое поле и должным образом его отобразить:

```
echo "<br>ISBN: ";
echo stripslashes($row["isbn"]);
```

Как уже упоминалось, **stripslashes()** вызывают для того, чтобы "подчистить" значение, прежде чем отображать его пользователю.

Существуют несколько вариантов получения результата из идентификатора результата. Вместо ассоциативного массива можно воспользоваться нумерованным массивом, применив **mysql_fetch_row()**:

```
$row = mysql_fetch_row($result);
```

Значения атрибутов будут храниться в каждом порядковом значении **$row[0], $row[1] и т.д.**

С помощью функции **mysql_fetch_object()** можно выбрать строку внутрь объекта:

```
$row = mysql_fetch_object($result);
```

После этого к каждому атрибуту можно получить доступ через **$row->title**, **$row->author и т.д.**

Каждый из этих вариантов подразумевает выборку строки за раз. Другой вариант — получить доступ, используя **mysql_result()**. Для этого потребуется указать номер строки (от 0 до количества строк минус 1) и название поля, например:

```
$row = mysql_result($result, $i, "title");
```

Название поля можно задать в виде строки (либо в форме **"title"** либо в форме **"books.title"**) или номером (как в **mysql_fetch_row()**). Не стоит смешивать **mysql_result()** с другими функциями выборки.

Строчно-ориентированные функции выборки намного более эффективны, нежели **mysql_result()**, так что лучше использовать одну из них.

Отсоединение от базы данных

Для закрытия непостоянного соединения применяется функция:

```
mysql_close( database_connection);
```

Однако в этом нет особой необходимости, поскольку с завершением выполнения сценария соединение закроется автоматически.

Внесение новой информации в базу данных

Внесение новой информации очень похоже на получение существующей. Нужно пройти те же шаги — установить соединение, отправить запрос и проверить результаты. Только в данном случае вместо **SELECT** будет использоваться **INSERT.**

Хоть все вроде и просто, но на пример взглянуть никогда не помешает. На рис. 10.3 показана обычная HTML-форма для добавления новых книг в базу данных.

HTML-код этой страницы приведен в листинге 10.3.

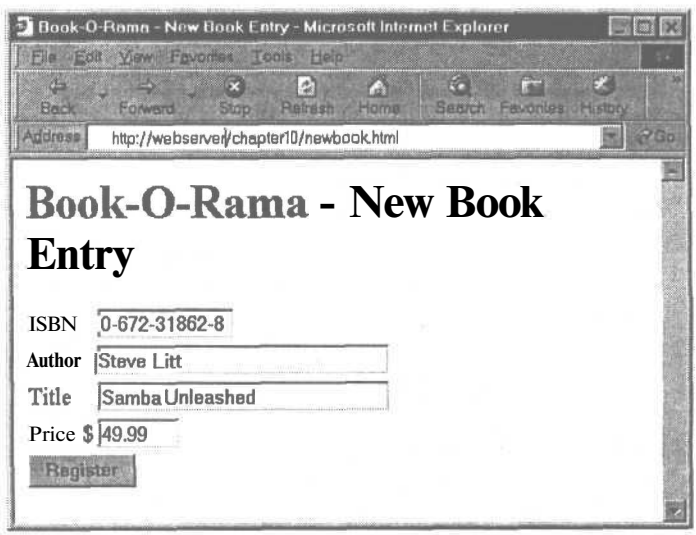


РИСУНОК 10.3
Интерфейс для добавления новых книг в базу данных, который может пригодиться в магазине Book-O-Rama.

Листинг 10.3 newbook.html — HTML-код страницы добавления новых книг

```
<html>
<head>
   <title>Book-O-Rama - New Book Entry</title>
</head>
<body>
   <h1>Book-O-Rama - New Book Entry</h1>

   <form action="insert_book.php" method="post">
      <table border=0>
         <tr>
            <td>ISBN</td>
              <tdXinput type=text name=isbn maxlength=13 size=13><br></td>
         </tr>
         <tr>
            <td>Author</td>
            <td> <input type=text name=author maxlength=30 size=30Xbr></td>
         </tr>
         <tr>
            <td>Title</td>
            <td> <input type=text name=title maxlength=60 size=30Xbr></td>
         </tr>
         <tr>
            <td>Price $</td>
            <td><input type=text name=price maxlength=7 size=7XbrX/td>
         </tr>
         <tr>
            <td colspan=2><input type=submit value="Register"></td>
         </tr>
      </table>
   </form>
</body>
</html>
```

Результаты заполнения этой формы передаются в **insert_book.php**, а сценарий, занимающийся деталями, выполняет определенную аутентификацию и пытается записать данные в базу данных. Код для сценария представлен в листинге 10.4.

Листинг 10.4 insert_book.php — сценарий записывает новые книги в базу данных

```
<html>
<head>
   <title>Book-O-Rama Book Entry Results</title>
</head>
<body>
<h1>Book-O-Rama Book Entry Results</h1>
<?
   if (!$isbn || !$author || !$title I I ! $price)
   {
       echo "You have not entered all the required details.<br>"
             ."Please go back and try again.";
       exit;
   }

   $isbn   = addslashes($isbn);
   $author = addslashes($author);
   $title  = addslashes($title) ;
   $price  = doubleval($price);

   @ $db = mysql_pconnect("localhost", "bookorama", "bookorama");
```

```
    if  (!$db)
    {
        echo "Error: Could not connect to database.  Please try again
    later.";
        exit;
    }

    mysql_select_db("books");
    $query = "insert into books values
                ('".$isbn."', '".$author."', '".$title."','".$price."')";
    $result = mysql_query($query);
    if  ($result)
         echo mysql_af f ected_rows ( ) . " book inserted into database.";
?>

</body>
</html>
```

Результаты успешного добавления книги в базу данных показаны на рис. 10.4.

После изучения кода **insert_book.php** станет ясно, что он во многом похож на код сценария для извлечения данных из базы. Мы проверяем, чтобы все поля формы были заполнены и отформатированы с помощью **addslashes()** перед внесением данных в базу:

```
$isbn = addslashes ($isbn);
$author = addslashes ($author);
$title = addslashes ($title);
$price = doubleval ($price);
```

Поскольку цены хранятся в базе в виде чисел с плавающей запятой, символы наклонной черты они содержать не должны. Это же достигается при помощи функции **doubleval()**, которая отфильтрует все неподходящие символы в числовом поле. Мы рассматривали **doubleval()** в главе 1. Эта же функция позаботится и обо всех символах валюты, которые пользователь может печатать при заполнении формы.

Мы снова соединяемся с базой данных, используя **mysql_pconnect()**, и настраиваем запрос. В данном случае это **INSERT.**

```
$query = "insert into books values
   (' ".$isbn." ', ' ".$author." ', ' ''.$title." ', ' ''.$price.'' ')";
$result = mysql_query($query);
```

Выполнение происходит не без помощи **mysql_query()**.

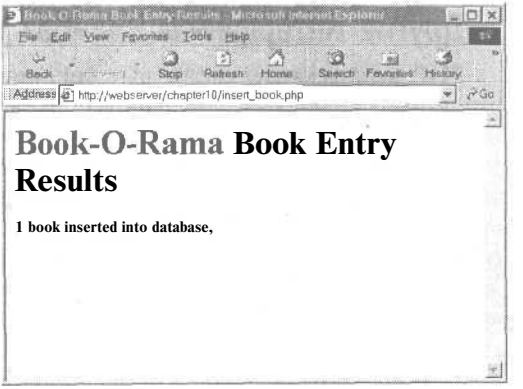


РИСУНОК 10.4
Сценарий завершен успешно и он сообщает о том, что книга добавлена в базу данных.

Одно существенное различие между **INSERT и SELECT** заключается в использовании **mysql_affected_rows()**:

```
echo mysql_affected_rows(). " book inserted into database.";
```

В предыдущем сценарии функция **mysql_num_rows()** применялась для определения количества строк, которые будет возвращать **SELECT.** При написании запросов, которые изменяют базу данных, например, **INSERT, DELETE, UPDATE,** следует использовать **mysql_affected_rows()**.

Мы рассмотрели основы использования баз данных MySQL из PHP. Взглянем еще на некоторые полезные функции, не упомянутые ранее.

Прочие полезные функции PHP-MySQL

Рассмотрим кратко несколько полезных функций PHP-MySQL.

Освобождение ресурсов

Если во время выполнения сценария возникают проблемы, связанные с нехваткой памяти, может пригодиться функция **mysql_free_result()**. Вот ее прототип:

```
int mysql_free_result(int result);
```

Она вызывается с идентификатором результата:

```
mysql_free_result($result);
```

В итоге освобождается память, занимаемая результатом. Очевидно, что до обработки результата эта функция вызываться не должна.

Создание и удаление баз данных

Для создания новой базы данных MySQL из PHP-сценария применяется функция **mysql_create_db()**, а для удаления базы данных — **mysql_drop_db()**.

Рассмотрим прототипы этих функций:

```
int mysql_create_db(string database, [int database_connection] );
int mysql_drop_db(string database, [int database_connection] );
```

Обе функции используют имя базы данных и соединение. Если соединения нет, будет использоваться последнее открытое. Функции создают либо удаляют указанную базу данных. В случае успеха функции возвращают значение true, а в случае неудачи — false.

Другие интерфейсы PHP-баз данных

PHP поддерживает различные библиотеки, что дает возможность подключаться к огромному количеству баз данных, включая Oracle, Microsoft SQL Server, mSQL и PostgreSQL.

В целом принципы подключения и подачи запросов любой из этих баз данных одни и те же. Названия функций могут быть разными, разные базы данных могут иметь разную функциональность, но если вы можете подключиться к MySQL, то другие базы вряд ли поставят вас в безвыходное положение.

Если необходимо использовать базу данных, которая не имеет специфической библиотеки, доступной в PHP, можно прибегнуть к обобщенным функциям ODBC.

ODBC — это открытый интерфейс доступа к базам данных и является стандартом подключения к базам данных. Функциональность ODBC нельзя назвать чересчур широкой, однако на то имеются вполне очевидные причины: либо универсальная совместимость, либо задействование специфических возможностей каждой системы.

Вдобавок к PHP-библиотекам, доступны такие классы абстракции баз данных, как Metabase, что позволяет использовать одни и те же названия функций в каждом типе базы данных.

Дополнительная информация

Дополнительно о соединении MySQL и PHP можно прочитать в соответствующих разделах руководства по PHP и MySQL.

Информация об ODBC доступна по адресу:

```
http://www.whatis.com/odbc.htm
```

Информация по Metabase находится на Web-сайте:

```
http://phpclasses.upperdesign.com/browse.html/package/20
```

Что дальше

В следующей главе мы более детально рассмотрим вопросы администрирования MySQL и поразмышляем о путях оптимизации работы вашей базы данных.

Глава 11

Дополнительные возможности MySQL

В этой главе рассматриваются некоторые дополнительные возможности MySQL: привилегии, безопасность и оптимизация.

Вот темы, которые будут затронуты:

- Детальное описание системы привилегий
- Обеспечение безопасности баз данных MySQL
- Получение дополнительной информации о базах данных
- Ускорение использования базы данных при помощи индексов
- Советы по оптимизации
- Различные типы таблиц

Детальное описание системы привилегий

Ранее, в главе 8, исследовались вопросы подключения пользователей и присвоения им привилегий. При этом применялась команда **GRANT.** Если вы собираетесь администрировать базу данных MySQL, необходимо четко понимать, что именно делает **GRANT** и каким образом.

Оператор **GRANT** оказывает влияние на таблицы в специальной базе данных, которая называется **mysql**. Информация о привилегиях хранится в пяти таблицах этой базы данных. Поэтому, присваивая пользователям привилегии в базах данных, будьте осторожны с выдачей доступа к базе данных **mysql.**

Следует отметить, что команда **GRANT** доступна в MySQL только начиная с версии 3.22.11.

Посмотреть содержимое базы данных **mysql** можно, зарегистрировавшись в системе как администратор и набрав:

```
use mysql;
```

После можно просмотреть таблицы этой базы, набрав:

```
show tables;
```

как обычно.

В результате отображается нечто подобное:

```
+------------------+
| Tables in mysql  |
+------------------+
| columns_priv     |
| db               |
| host             |
| tables_priv      |
| user             |
+------------------+
```

В каждой из этих таблиц содержится информация о привилегиях. Их еще называют *таблицами прав*. Их функции могут быть несколько разными, но задача у всех одна – определять, что позволено, а что не позволено делать пользователям. В каждой таблице содержится два типа поля: контекстные поля, которые идентифицируют пользователя, хост и часть базы данных, и поля привилегий, определяющие, какие действия может выполнять пользователь, исходя из своего контекста.

Таблица **user** (пользователь) предназначена для определения, может ли пользователь подключаться к серверу MySQL и обладает ли он привилегиями администратора. Таблицы db и host определяют, к каким базам данных пользователь может иметь доступ. Таблица **tables_priv** — какие таблицы в базе данных разрешается использовать, а **columns_priv** — к каким столбцам в таблицах имеется доступ.

Таблица user

В этой таблице содержатся данные о глобальных привилегиях пользователя, как то: имеет ли право пользователь подключаться к серверу MySQL вообще и есть ли у него какие-либо привилегии глобального уровня, т.е. привилегии, которые распространяются на все базы данных в системе.

Оператор **describe user;** отображает структуру этой таблицы.

Схема таблицы **user** приведена в табл. 11.1.

Таблица 11.1 Схема таблицы user в базе данных mysql

Поле	*Тип*
Host	char(60)
User	char(16)
Password	char(16)
Select_priv	enum('N','Y')
Insert_priv	enum('N','Y')
Update_priv	enum('N','Y')
Delete_priv	enum('N','Y')
Create_priv	enum('NYY')
Drop_priv	enum('N','Y')

Поле	*Тип*
Reload_priv	enum('N','Y')
Shutdown_priv	enum('N','Y')
Process_priv	enum('N','Y')
File_priv	enum('N','Y')
Grant_priv	enum('N','Y')
References_priv	enum('N','Y')
Index_priv	enum('N','Y')
Alter_priv	enum('N','Y')

Каждая строка в этой таблице соответствует набору привилегий пользователя, который регистрируется из определенного хоста с применением имени и пароля. В таблице имеются *поля с диапазоном,* которые описывают диапазон для других полей, называемых *полями привилегий.*

Приведенные в таблице привилегии (равно как и те, которые будут рассматриваться далее) полностью согласуются с привилегиями, предоставляемыми с использованием **GRANT** (см. главу 8). Например, **select_priv** напрямую соответствует привилегиям по выполнению оператора **SELECT.**

Если пользователь обладает определенной привилегией, значением в столбце типа будет Y. Точно так же, если данной привилегии у пользователя нет, значением столбца будет N.

Все привилегии, перечисленные в таблице **user,** являются глобальными, т.е. они применяются *ко всем базам данных в системе* (включая и базу **mysql**). Большинство значений в столбце, равных Y, будут иметь администраторы, тогда как для пользователей будет характерно изобилие значений N. Типовые пользователи обладают правами для совершенно конкретных баз данных, а не для всех существующих таблиц в системе.

Таблицы db и host

Большинство привилегий для рядовых пользователей системы размещаются в таблицах **db** и **host.**

Таблица **db** определяет, какие пользователи к каким таблицам и из каких хостов могут получить доступ. Перечисленные в таблице привилегии применяются к любой базе данных.

Таблица **host** дополняет таблицу db. Если пользователь должен соединяться с некоторой базой данных, используя при этом множество хостов, в таблице **db** для него не будет указано ни одного хоста. Наоборот, с таким пользователем будет связан набор записей в таблице **host,** по одной для каждого возможного хоста.

Схемы таблиц **db** и **host** показаны, соответственно, в табл. 11.2 и 11.3.

Таблица 11.2 Схема таблицы db в базе данных mysql

Поле	*Тип*
Hosf	char(60)
Db	char(64)
User	char(16)

Поле	Тип
Select_priv	enum('N','Y')
Insert_priv	enum('N','Y')
Update_priv	enum('N','Y')
Delete_priv	enum('N','Y')
Create_priv	enum('N','Y')
Drop_priv	enum('N','Y')
Grant_priv	enum('N','Y')
References_priv	enum('N','Y')
Index_priv	enum('N','Y')
Alter_priv	enum('N','Y')

Таблица 11.3 Схема таблицы host в базе данных mysql

Поле	*Тип*
Host	char(60)
Db	char(64)
Select_priv	enum('N','Y')
Insert_priv	enum('N','Y')
Update_priv	enum('N','Y')
Delete_priv	enum('N','Y')
Create_priv	enum('N','Y')
Drop_priv	enum('N','Y')
Gran+_priv	enum('N','Y')
References_priv	enum('N','Y')
Index_priv	enum('N','Y')
Alter_priv	enum ('N','Y')

Таблицы tables_priv и columns_priv

Эти таблицы предназначены для хранения привилегий, соответственно, на уровне таблицы и на уровне столбцов. Они работают подобно таблице **db** за исключением того, что обеспечивают привилегии для таблиц в определенной базе данных и для столбцов в определенной таблице.

Структура рассматриваемых таблиц несколько отличается от структуры таблиц **user, db и host.** Схемы таблиц **tables_priv и columns_priv** показаны в табл. 11.4 и 11.5 соответственно.

Таблица 11.4 Схема таблицы tables_priv в базе данных mysql

Поле	Тип
Host	char(60)
Db	char(64)
User	char(16)
Table_name	char(64)
Grantor	char(77)
Timestamp	timestamp(14)
Table_priv	set('Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter')
Column_priv	set('Select', 'Insert', 'Update', 'References')

Таблица 11.5 Схема таблицы columns_priv в базе данных mysql

Поле	Тип
Host	char(60)
Db	char(60)
User	char(16)
Table_name	char(60)
Column_name	char(59)
Timestamp	timestamp(14)
Column_priv	set('Select', 'Insert', 'Update', 'References')

Столбец **Grantor** таблицы **tables_priv** хранит информацию о пользователе, который выдал привилегию данному пользователю. Столбец **Timestamp** в обеих таблицах хранит информацию о дате и времени выдачи привилегии.

Управление доступом: как MySQL использует таблицы привилегий

MySQL использует таблицы привилегий с целью определения, что пользователю позволено делать, и совершает это в два этапа.

1. Проверка соединения. На этом этапе MySQL проверяет, есть ли у вас вообще право подключаться, исходя из данных таблицы **user,** как показано выше. Используется информация об имени пользователя, хосте и пароле. Если поле имени пользователя пусто, значит, подходит любое имя пользователя. Имя узла можно указать символом %. Этим символом можно заполнить целое поле, т.е. % будет означать "любой узел". Можно также включить этот символ в часть названия, например, **%.tangledweb.com.au** соответствует всем узлам, завершающимся на **.tangledweb.com.au**. Если поле пароля пустое, значит, пароль не требуется. С точки зрения безопасности лучше, конечно, избегать пустых полей с именем пользователя, символов шаблона в названиях хостов и пользователей без пароля.
2. Подтверждение запроса. Всякий раз когда соединение уже установлено и запрос отправлен, MySQL проверяет, есть ли у вас необходимый уровень привилегий для выполнения такого запроса. Система начинает с проверки глобальных при-

вилегий (по таблице **user**), и если их недостаточно, проверяет таблицы **db** и **host.** Если привилегий все равно не хватает, MySQL проверит таблицу **priv_table** и, в конце концов, таблицу **columns_priv**.

Обновление привилегий: когда изменения вступают в силу?

При запуске и после ввода операторов **GRANT и REVOKE** сервер MySQL автоматически читает таблицы привилегий.

В любом случае, поскольку теперь мы знаем, где и как хранятся эти привилегии, мы можем редактировать их вручную. Однако если обновлять данные таким образом, сервер MySQL *не заметит, что данные изменились.*

Необходимо сообщить серверу, что произошли изменения, и для этого существуют три способа. Можно набрать:

```
FLUSH PRIVILEGES;
```

в командной строке MySQL (для чего необходимо быть администратором). Это наиболее типичный способ обновления привилегий.

Существует и альтернатива:

```
mysqladmin flush-privileges
```

ИЛИ

```
mysqladmin reload
```

из вашей операционной системы.

После этого при следующем подключении пользователя будут проверяться его привилегии глобального уровня; привилегии уровня базы данных будут проверяться при встрече оператора use, а привилегии уровня таблицы и столбцов — при новом запросе пользователя.

Обеспечение безопасности баз данных MySQL

Безопасность очень важна, особенно когда вы начинаете подключать базу данных MySQL к Web-сайту. В этом разделе мы поговорим о мерах предосторожности, которые следует соблюдать в плане защиты базы данных.

MySQL с точки зрения операционной системы

Если вы работаете в UNIX-подобной операционной системе, то запускать MySQL-сервер **(mysqld)** как привилегированному пользователю не рекомендуется, поскольку помимо полного набора привилегий это дает пользователю MySQL право читать и записывать файлы в любом месте операционной системы. Мы затронули очень немаловажный момент; именно таким способом чаще всего взламывали Web-сайты Apache. (К счастью, взломщики оказались "недюжинными джентльменами" и единственное, чего они добились — это ужесточения мер безопасности.)

Для этой цели лучше всего завести отдельного пользователя. Вдобавок, можно сделать каталоги (в которых хранятся физические данные) доступными только конкретному пользователю MySQL. Во многих установках сервер настроен на запуск под именем пользователя **mysql** в группе mysql.

В идеале, лучше, конечно устанавливать сервер MySQL за брандмауэром. Это даст возможность предотвратить несанкционированный доступ — проверьте, сможете ли вы подключиться к серверу извне через порт 3306. Это порт, на котором MySQL запускается по умолчанию и в брандмауэре он должен быть закрыт.

Пароли

Следите за тем, чтобы у всех пользователей (особенно, у привилегированного) были определены пароли, причем хорошо выбранные и регулярно изменяемые, как в операционной системе. Главное, что следует помнить в данном случае — пароли, составленные из словарных слов, весьма слабы. Лучше воспользоваться комбинациями из букв и цифр.

Если вы собираетесь хранить пароли в файлах сценариев, следите за тем, чтобы только тот пользователь, чей пароль находится в этом файле, мог его открыть. Речь идет о двух типичных случаях:

1. В сценарии mysql.server может потребоваться пароль привилегированного пользователя UNIX. Если это так, то пусть доступа к этому файлу не имеет никто, кроме него.
2. Пароль такого пользователя должен быть сохранен в PHP сценариях, используемых для подключения к базам данных. Это можно сделать безопасно, если поместить имя пользователя и пароль в файл с названием, скажем, dbconnect.php, который будет включаться по мере необходимости. Сценарий может храниться вне дерева Web-документов и быть доступным только определенному пользователю. Помните, что если поместить эти детали в a.inc или в файл с другим расширением в рамках Web-дерева, следует проявлять предельную осторожность и проверять, знает ли Web-сервер, что эти файлы требуется интерпретировать как PHP-код, чтобы детали нельзя было просмотреть в Web-браузере.

Не храните пароли своей базы данных в текстовых файлах. Пароли MySQL обычно так не хранят, однако зачастую в Web-приложениях сохраняют имена пользователей Web-сервера и их пароли. Шифровать пароли (односторонне) можно, используя MySQL-функции **PASSWORD()** и MD5(). Помните, что когда вы вставляете пароль (INSERT) в одном из этих форматов, выполняя SELECT (дабы обеспечить вход пользователя в систему), придется использовать эту функцию еще раз, чтобы проверить, какой пароль был введен пользователем.

Мы будем применять такую функциональность, когда приступим к реализации проектов в пятой части.

Привилегии пользователей

Знание — сила. Переспросите себя, понимаете ли вы систему привилегий MySQL и взаимосвязи между конкретными привилегиями? Не давайте пользователю привилегий больше, чем необходимо. Это можно проверить, исследовав таблицы привилегий.

Если говорить конкретнее, то не давайте привилегий PROCESS, FILE, SHUTDOWN и RELOAD никому, кроме администратора, разве что в случае крайней необходимости. Привилегию PROCESS можно использовать для того, чтобы увидеть, что делают и набирают другие пользователи, включая и пароли. Привилегия FILE используется для чтения и записи файлов в операционной системе (включая, например, /etc/password в системе UNIX).

Привилегию GRANT также следует выдавать осторожно, поскольку она предоставляет пользователям возможность делить свои привилегии с другими.

При подключении новых пользователей предоставляйте им доступ на подключение к базе только с тех хостов, с которых они могут заходить на нее. Если есть пользователь jane@localhost, это замечательно, а вот просто jane можно встретить где угодно, и она может зайти откуда угодно и не факт, что это именно та jane, которую вы

знаете и любите. По тем же причинам избегайте употребления групповых символов в названиях узлов.

Повысить уровень безопасности можно также, используя в таблице **host** IP-адреса вместо доменных имен. Это поможет защититься как от взломщиков, так и от ошибок в DNS. Это можно осуществить, запуская демон MySQL с опцией **--skip-name-resolve,** что подразумевает замену всех значений столбцов хостов на IP-адреса или локальный хост.

Еще одна альтернатива — запускать mysqld с опцией **--secure**. Таким образом можно проверить разрешенные IP-адреса, преобразовываются ли они обратно в имя узла. (По умолчанию эта опция включена, начиная с версии 3.22.)

Кроме того, стоит следить за тем, чтобы непривилегированные пользователи не имели доступа к программе **mysqladmin** на Web-сервере. Поскольку она выполняется из командной строки, она также во многом связана с привилегиями на уровне операционной системы.

Тонкости Web

При подключении базы данных MySQL к Web возникают вопросы, касающиеся некоторых аспектов безопасности.

Есть особый смысл в том, чтобы создать пользователя специально для Web-соединений. Такому пользователю можно выдать минимум необходимых привилегий, не предоставляя возможности выполнять **DROP, ALTER и CREATE. SELECT** можно выдавать исключительно для таблиц каталогов, а **INSERT** — для таблиц заказов. Это отличный пример применения принципа наименьших привилегий.

ПРЕДУПРЕЖДЕНИЕ

В последней главе говорилось о применении PHP-функций **addslashes()** и **stripslashes()** с целью избавления от проблемных символов в строках. Не забывайте об этом. Равно как и о том, что перед отправкой что-либо в MySQL стоит проделать чистку основных данных. Если помните, с помощью функции **doubleval()** мы добивались того, чтобы числовые данные были действительно числовыми. Это часто упускают из виду — люди помнят о том, что нужно использовать **addslashesl),** но забывают проверять числовые данные.

Всегда проверяйте данные, исходящие от пользователя. Даже в HTML-форме, состоящей из полей выбора и переключателей какой-нибудь пользователь может попытаться изменить URL, чтобы взломать ваш сценарий. Кроме того, стоит проверять и размер входящих данных.

Если пользователи вводят пароли или конфиденциальные данные, которые будут храниться в вашей базе данных, помните, что они передаются от браузера к серверу в текстовом формате, если конечно вы не используете SSL (Secure Sockets Layer – слой безопасных сокетов). Более детально SSL обсуждается позже.

Получение дополнительной информации о базах данных

До сих пор для того, чтобы узнать, какие таблицы имеются в базе данных и какие столбцы определены в этих таблицах, применялись **SHOW и DESCRIBE.**

Получение информации с помощью SHOW

Ранее использовалось

```
SHOW TABLES;
```

для того, чтобы получить список таблиц в базе данных.

Оператор

```
show databases;
```

отображает список доступных баз данных. После этого можно ввести **SHOW TABLES** и увидеть список таблиц в одной из этих баз данных:

```
show tables from books;
```

Если **SHOW TABLES** применяется без указания базы данных, отображаются таблицы используемой базы.

Когда таблицы известны, список столбцов можно получить следующим образом:

```
show columns from orders from books;
```

Если параметр базы данных не включен, оператор **SHOW COLUMNS** обратится к используемой базе. Можно также воспользоваться и нотацией **table.column**:

```
show columns from books.orders;
```

Еще одна полезная особенность оператора **SHOW** заключается в возможности просмотра привилегий пользователя. Набрав

```
show grants for bookorama;
```

мы увидим вот такой ответ:



РИСУНОК 11.1 *Результат ввода SHOW GRANTS.*

Показанные операторы **GRANT** не обязательно используются для выдачи привилегий определенному пользователю, однако текущий уровень его привилегий вы, без сомнения, сможете лицезреть.

ПРИМЕЧАНИЕ

Оператор **SHOW GRANTS** появился в MySQL 3.23.4; если у вас установлена более ранняя версия, такой вариант не сработает.

Есть много вариаций оператора **SHOW.** Их список представлен в табл. 11.6.

Таблица 11.6 Синтаксис оператора SHOW

Вариант	*Описание*
SHOW DATABASES [LIKE *база_данных*]	Выводит список доступных баз данных, с возможностью выбора подобных имени *база_данных*
SHOW TABLES [FROM *база_данных*] [LIKE *таблица*]	Выводит список таблиц базы данных, которые используются в данный момент, *а* при конкретном указании — из базы данных с именем *база_данных*. Возможна сортировка таблиц по именам.
SHOW COLUMNS FROM *таблица* [FROM *база_данных*] [LIKE *столбец*]	Выводит список всех столбцов указанной таблицы базы данных, которая используется в данный момент. Можно указывать также базу данных и названия столбцов. Вместо **SHOW COLUMNS** можно использовать **SHOW FIELDS.**

Вариант	*Описание*
SHOW INDEX FROM *таблица* [FROM *база_данных*]	Отображает детальную информацию по всем индексам конкретной таблицы используемой базы данных или детали индексов базы данных, если она указана. Также можно использовать **SHOW KEYS.**
SHOW STATUS [LIKE *состояние_процесса*]	Выдает информацию о количестве системных процессов, например, количество параллельных процессов. Конструкция **LIKE** применяется для поиска по именам процессов, например, по критерию **'Thread%'** можно найти **'Threads_cached', 'Threads_connected', 'Threads_running'.**
SHOW VARIABLES [LIKE *название_переменной*]	Отображает имена и значения системных переменных MySQL, например, номер версии. Конструкция **LIKE** может использоваться для уточнения критерия поиска таким же образом, как и **SHOW STATUS.**
SHOW [FULL] PROCESSLIST	Выводит список всех выполняющихся процессов в системе, т.е. всех обрабатываемых запросов. Большинство пользователей могут просматривать, какие процессы они запустили, однако если пользователь обладает привилегией **PROCESS,** он может видеть процессы других пользователей, в том числе просматривать и пароли, если они использовались в запросе. По умолчанию запросы усекаются до ста символов. Опция **FULL** позволяет увидеть запрос целиком.
SHOW TABLE STATUS [FROM *база_данных*] [LIKE *база_данных*]	Выводит информацию о каждой таблице, используемой в данный момент, или о таблице, имя которой указано. Можно использовать сопоставление с универсальным шаблоном. Информация включает в себя тип таблицы и последнее время обновления таблицы.
SHOW GRANTS FOR *пользователь*	Отображает операторы **GRANT,** необходимые для выдачи указанному пользователю текущего уровня привилегий.

Получение информации о столбцах с помощью DESCRIBE

Альтернативой оператору **SHOW COLUMNS** является **DESCRIBE,** подобный оператору **DESCRIBE** в базе данных Oracle. Базовый синтаксис его таков:

```
DESCRIBE таблица [столбец];
```

Это обеспечивает отображение информации обо всех столбцах таблицы или о конкретном столбце, если он указан. По желанию в названии столбца можно воспользоваться универсальным шаблоном.

Как запросы работают с EXPLAIN

Оператор **EXPLAIN** можно использовать в двух вариациях. Можно набрать:

```
EXPLAIN таблица;
```

Это выдаст результат, похожий на результат использования **DESCRIBE** ***таблица*** или **SHOW COLUMNS FROM** ***таблица.***

Второй, более интересный вариант заключается в применении **EXPLAIN** для того, чтобы посмотреть, как MySQL выражает в числовом отношении запрос **SELECT.** В этом случае достаточно перед оператором **SELECT** поместить слово **explain.**

Оператором **EXPLAIN** можно воспользоваться, когда необходимо выполнить сложный запрос или когда процесс обработки запроса занимает намного больше времени, чем требуется. При построении достаточно сложного запроса можно проверить его загодя, выполнив команду **EXPLAIN** перед тем, как, собственно, начать обработку запроса. Результат выполнения упомянутой команды может натолкнуть на мысли по оптимизации работы базы данных MySQL, если, конечно, в этом есть необходимость. Кроме того, **EXPLAIN** очень полезен в плане самообучения.

Например, запустим следующий запрос в нашей базе данных Book-O-Rama. Результат будет выглядеть подобно показанному на рис. 11.2.

```
explain
select customers.name
from customers, orders, order_items, books
where customers.customerid = orders.customerid
and orders.orderid = order_items.orderid
and order_items.isbn = books.isbn
and books.title like '%Java%';
```

```
+-------------+--------+---------------+-------------+---------+-------------------+------+-------------+
| table       | type   | possible_keys | key         | key_len | ref               | rows | Extra       |
+-------------+--------+---------------+-------------+---------+-------------------+------+-------------+
| orders      | ALL    | PRIMARY       | NULL        |    NULL | NULL              |    4 |             |
| order_items | ref    | PRIMARY       | PRIMARY     |       4 | orders.orderid    |    1 | Using index |
| customers   | ALL    | PRIMARY       | NULL        |    NULL | NULL              |    3 | where used  |
| books       | eq_ref | PRIMARY       | PRIMARY     |      13 | order_items.isbn  |    1 | where used  |
+-------------+--------+---------------+-------------+---------+-------------------+------+-------------+
```

РИСУНОК 11.2 ***Результат выполнения оператора EXPLAIN.***

Поначалу это может смутить, однако пользы от этой информации немало. Рассмотрим каждый из столбцов этой таблицы.

В первом столбце, **table,** показаны таблицы, которые используются для ответа на запрос. Каждая строка результата выдает дополнительную информацию о том, каков же вклад каждой конкретной таблицы в общее дело поиска. В нашем примере задействованы таблицы **orders, order_items, customers и books.** (Это стало известно сразу после просмотра запроса.)

Столбец **type** показывает, как используется конкретная таблица в запросе. Набор значений, которые может иметь столбец, перечислены в табл. 11.7. Значения поданы по времени выполнения запроса, от самого быстрого до самого медленного. Это может подсказать, сколько строк из каждой таблицы потребуется читать для того, чтобы выполнить запрос.

Таблица 11.7 Возможные типы объединений, как они показаны в результате EXPLAIN

Тип	*Описание*
const или system	Информация из таблицы читается один раз. Это типично для случаев, когда в таблице находится одна строка. Тип **system** используется, когда это системная таблица, в других случаях применяется **const.**
eq_ref	При любом наборе строк из других таблиц объединения из этой таблицы читается одна строка. Это применяется, когда объединение использует в таблице все части индекса, причем индекс равен **UNIQUE** или первичному ключу.
ref	При любом наборе строк из других таблиц объединения читается набор совпадающих строк из данной таблицы. Применяется в тех случаях, когда объединение по своим условиям не может выбрать одну строку, например, когда в объединении задействована только часть ключа или если это не **UNIQUE** и не первичный ключ.

Тип	*Описание*
range	При любом наборе строк в других таблицах объединения читается набор строк из данной таблицы, попадающий в определенную область.
index	Сканируется индекс целиком.
ALL	Сканируется каждая строка в таблице.

В предыдущем примере одна из таблиц объединена с использованием **eq_ref (books),** а другая — с использованием **ref (order_items),** однако две других объединены с использованием **ALL,** т.е. с просмотром каждой строки таблицы.

Столбец **rows** выступает в качестве поддержки данной опции. Он выводит в список количество строк каждой таблицы, которое необходимо сканировать для выполнения запроса. Когда запрос выполнен, значения можно перемножить, чтобы получить общее количество сканированных строк. Мы делаем это, поскольку объединение — суть произведение строк разных таблиц (см. главу 9). Помните, что имеется в виду количество сканированных строк, а не выданных в результате, т.е. это просто оценка, MySQL не будет знать конкретного числа, пока не выполнит запрос.

Очевидно, что чем меньше это число, тем лучше. В настоящее время в нашей базе данных количество данных ничтожно мало, однако когда база данных начинает возрастать, подобный запрос в процессе выполнения замедляется. Через минуту мы к этому вернемся.

Столбец **possible_keys** выводит, как вы уже, наверное, догадались, список ключей, которые MySQL может использовать для объединения таблицы. В нашем случае все возможные ключи — первичные.

Столбец key — это либо ключ из таблицы, которую использует MySQL, либо **NULL,** если не используется ни один ключ. Вы заметите, что несмотря на наличие первичных ключей в таблицах **orders и customers,** в запросе может не быть ни одного ключа. Как это исправить, мы узнаем через минуту.

Столбец **key_len** показывает, какова длина использованного ключа. С помощью этого столбца можно узнать, задействован ключ целиком или частично. Это может пригодиться, если существуют ключи, состоящие из нескольких столбцов. В нашем примере, там где использовались ключи **(order_items и books),** они использовались целиком.

Столбец **ref** показывает столбцы, для извлечения строк из которых применялись ключи.

И наконец, столбец **extra** содержит массу полезной информации о том, как прошло объединение. Возможные значения этого столбца перечислены в табл. 11.8.

Таблица 11.8 Возможные значения столбца extra, как показано в результате выполнения explain.

Значение	*Объяснение*
Not exists (Не существует)	Запрос оптимизирован для использования **LEFT JOIN**
Range checked for each record (Проверка диапазона для каждой записи)	Для каждой строки в наборе из других таблиц объединения искать наилучший индекс, если таковые имеются.
Using filesort (Использование сортировки)	Для сортировки данных потребуются два просмотра (вполне естественно, это занимает в два раза больше времени).

Значение	*Объяснение*
Using index (Использование индекса)	Вся информация из таблицы поступает из индекса, т.е. сами строки не просматриваются.
Using temporary (Использование временной таблицы)	Для выполнения этого запроса потребуется создание временной таблицы.
WHERE used	Для отбора строк используется конструкция **WHERE.**

Есть несколько вариантов устранения проблем, которые выползают наружу после выполнения **explain.**

Во-первых, проверьте типы столбцов, чтобы убедиться, что они не изменились. В особенности это относится к ширине столбцов. Если ширина столбцов разная, индексы не смогут уравнять такие столбцы. Можно изменить типы столбцов либо четко запланировать это с самого начала проектирования базы данных.

Во-вторых, можно указать оптимизатору объединений проинспектировать распределение ключей, оптимизировав с помощью утилиты **myisamchk** объединения более эффективно. Для этого потребуется набрать:

```
>myisamchk --analyze путь_к_базе_данных_mysql/таблица
```

Несколько таблиц можно проверить, указав их имена в командной строке, либо воспользовавшись:

```
>myisamchk --analyze путь_к_базе_данных_mysql/*.MYI
```

Следующим образом можно проверить все таблицы во всех базах данных (результат показан на рис. 11.3):

```
>myisamchk --analyze путь_к_базе_данных_mysql/*/*.MYI
```

```
+-------------+--------+---------------+---------+---------+-------------------------+------+-------------+
| table       | type   | possible_keys | key     | key_len | ref                     | rows | Extra       |
+-------------+--------+---------------+---------+---------+-------------------------+------+-------------+
| books       | ALL    | PRIMARY       | NULL    |    NULL | NULL                    |    4 | where used  |
| order_items | index  | PRIMARY       | PRIMARY |      17 | NULL                    |    8 | where used; Using index |
| orders      | eq_ref | PRIMARY       | PRIMARY |       4 | order_items.orderid     |    1 |             |
| customers   | eq_ref | PRIMARY       | PRIMARY |       4 | orders.customerid       |    1 |             |
+-------------+--------+---------------+---------+---------+-------------------------+------+-------------+
```

РИСУНОК **11.3** *Результат EXPLAIN после выполнения myisamchk.*

Несложно заметить, что критерий оценки запроса существенно изменился. Теперь мы используем только *все (all)* строки одной таблицы **(books),** чего вполне достаточно. В частности, для двух таблиц используется **eq_ref**, а для трех — **index.** MySQL также задействует полный ключ для **order_items** (17 символов против 4 прежних).

Возможно, внимание привлечет и тот факт, что количество используемых строк увеличилось. Это можно объяснить тем, что данных в нашей базе на нынешний момент мало. Количество строк, вынесенных в список, это всего лишь оценка — попытайтесь выполнить какой-либо запрос и убедитесь в этом. Если этих чисел нет, руководство по MySQL рекомендует использовать прямое объединение и листинг таблиц в конструкции **FROM** в другом порядке.

В-третьих, может возникнуть желание добавить в таблицу новый индекс. Если запрос медленный или общий, задумайтесь над этим всерьез. Если же запрос одноразовый и он не будет запускаться еще раз, игра не стоить свеч, если только не замедлятся остальные процессы. Как добавить новый индекс, мы узнаем из следующей главы.

Ускорение использования базы данных при помощи индексов

Если вы попали в вышеописанную ситуацию, когда столбец **possible_keys** команды **EXPLAIN** содержит несколько значений **NULL,** выполнение запроса можно ускорить за счет добавления в таблицу индекса. Если используемый в конструкции **WHERE** столбец допускает индексацию, можно создать для такой таблицы новый индекс, воспользовавшись **ALTER TABLES,** например:

```
ALTER TABLE таблица ADD INDEX (столбец);
```

Советы по оптимизации

Вдобавок к перечисленным способам оптимизации, представим несколько способов, которые могут в целом заметно улучшить производительность базы данных MySQL.

Оптимизация проектирования

Главное, что может потребоваться в базе данных — это чтобы она была как можно меньше. Этого можно достичь вместе с улучшенным проектированием, которое уменьшает избыточность. Второй вариант связан с применением наименьшего из возможных типов данных в столбцах. Где это только возможно, уберите нулевые значения и сделайте первичные ключи как можно короче.

Избегайте неоднородной длины столбцов, где это возможно (например, **VARCHAR, TEXT, BLOB).** Таблицы с фиксированной длиной полей работают быстрее, однако вместе с тем могут занимать большее пространство.

Права доступа

Помимо выполнения советов из предыдущей части, касающихся **EXPLAIN,** скорость выполнения запроса можно увеличить, упростив права доступа. Ранее обсуждалось, как перед выполнением запросов осуществляется сверка с системой прав доступа. Чем проще этот процесс, тем быстрее выполняется запрос.

Оптимизация таблиц

Если таблица использовалась в течение некоторого времени, данные вследствие обновлений и удалений могут фрагментироваться. Фрагментация увеличивает время поиска данных в таблице. Это можно исправить с помощью оператора:

```
OPTIMIZE TABLE имя_таблицы;
```

или набрав:

```
>myisamchk -r таблица
```

в командной строке.

Утилиту **myisamchk** можно задействовать для сортировки индекса таблицы и данных, соответствующих этому индексу, например:

```
>myisamchk --sort-index --sort-records=1 путь_к_базе_данных_mysql/*/*.MYI
```

Применение индексов

Индексы используются для ускорения поиска. Пусть они остаются простыми; не стоит создавать индексы, которые в запросах не используются. Узнать, какие индексы используются в запросах, можно с помощью **EXPLAIN,** как было показано ранее.

Использование значений, принятых по умолчанию

Где только возможно, используйте для столбцов значения, принятые по умолчанию, пополняйте их только теми данными, которые отличаются от принятых по умолчанию. Это уменьшает время выполнения оператора INSERT.

Применяйте постоянные соединения

Этот совет касается Web-баз данных. Мы уже говорили об этом, так что теперь просто напоминаем.

Другие советы

Есть еще много хитростей, с помощью которых можно ускорять работу в конкретных ситуациях. Web-сайт MySQL предлагает довольно неплохой набор дополнительных советов. Загляните по адресу:

```
http://www.mysql.com
```

Различные типы таблиц

Еще один момент, который стоит обсудить, это существование различных типов таблиц. Тип таблицы можно выбрать при ее создании, используя:

```
CREATE TABLE таблица TYPE=тип . . .
```

Возможные типы таблиц:

- **MyISAM.** Этот тип принят по умолчанию и применяется до сих пор. Основа его — ISAM, что расшифровывается как *Indexed Sequential Access Method (индексно-последовательный метод доступа),* который представляет собой стандартный метод хранения записей и файлов.
- **HEAP.** Таблицы этого типа хранятся в памяти, причем их индексы хэшируются. Вследствие этого таблицы типа **HEAP** очень быстрые, но в случае малейших неполадок все данные окажутся утерянными. Такие характеристики делают таблицы типа **HEAP** идеальными для хранения временных и выводимых данных. В операторе **CREATE TABLE** необходимо указать **MAX_ROWS,** иначе такие таблицы "съедят" всю доступную память. В них нельзя определять столбцы **BLOB, TEXT и AUTO INCREMENT.**
- BDB. Эти таблицы безопасны для транзакций, т.е. они обеспечивают **COMMIT и ROLLBACK.** В работе они медленнее таблиц **MyISAM** и основываются на Berkely DB. В момент написания книги они еще находились на стадии отладки в MySQL 3.23.21, а для работы с ними потребуется дополнительная загрузка кода из Web-сайта MySQL.

Таблицы такого типа могут пригодиться, если требуется скорость и безопасность транзакций.

Загрузка данных из файла

Еще одна полезная особенность MySQL, о которой мы не говорили, это оператор **LOAD DATA INFILE.** Его можно использовать для загрузки табличных данных из файла. Выполняется этот оператор очень быстро.

Это очень гибкая команда со множеством опций, однако типичное использование выглядит приблизительно так:

```
LOAD DATA INFILE "newbooks.txt" INTO TABLE books;
```

Эта команда прочитает строчные данные из файла **newbooks.txt** в таблицу **books.** По умолчанию поля данных в файле должны разделяться знаками табуляции и заключаться в одинарные кавычки, а строки должны разделяться символами новой строки (\n). Специальные символы должны обозначаться наклонной чертой (\). Все эти характеристики конфигурируются различными опциями оператора **LOAD** — за детальной информацией обращайтесь к руководству по MySQL.

Чтобы задействовать оператор **LOAD DATA INFILE,** пользователь должен обладать привилегией **FILE,** о которой упоминалось ранее.

Дополнительная информация

В этих главах по MySQL внимание фокусировалось на использовании тех частей системы, которые работают с Web и соединяют MySQL с PHP.

Если требуется получить дополнительную информацию по приложениям, не имеющим отношения к Web, или об администрировании MySQL, посетите Web-сайт MySQL по адресу:

```
http://www.mysql.com
```

Что дальше

Мы рассмотрели основы PHP и MySQL. В главе 12 мы обсудим вопросы электронной коммерции и аспекты безопасности Web-сайтов, обладающих поддержкой баз данных.

Часть 3

Системы электронной торговли и безопасность

В этой части:

12 Эксплуатация сайта электронной коммерции

13 Вопросы безопасности в электронной коммерции

14 Аутентификация с помощью PHP и MySQL

15 Реализация безопасных транзакций в PHP и MySQL

Глава 12

Эксплуатация сайта электронной коммерции

В этой главе рассматриваются некоторые проблемы, связанные со спецификацией, проектированием, созданием и эксплуатацией сайта электронной коммерции. На основе исследования плана сайта и оценки возможных рисков будет предпринята попытка отыскать пути его самоокупаемости.

Итак, нас интересуют следующие вопросы:

- Цели, преследуемые в результате создания сайта электронной коммерции
- Типы коммерческих Web-сайтов
- Риски и угрозы
- Выбор стратегии

Преследуемые цели

Прежде чем с головой уходить в разработку Web-сайта и всех составляющих его компонентов, следует уяснить для себя, какие задачи этот сайт призван решать, и уже на основе этого составить более или менее подробный план решения этих задач.

В данной книге будем исходить из предположения, что строится коммерческий Web-сайт; следовательно, главная задача — обеспечение прибыли.

Существует немало способов зарабатывания денег через Internet. Через сеть можно рекламировать обычные услуги либо продавать обычные товары. Некоторые товары можно не только продавать, но и доставлять через сеть. Не все Web-сайты предназначены для непосредственного извлечения дохода. Они могут обеспечивать определенную поддержку несетевых коммерческих операций либо рассматриваться в качестве более дешевой их альтернативы.

Типы коммерческих Web-сайтов

Коммерческие Web-сайты выполняют, как правило, одну или несколько из перечисленных ниже функций:

- Распространение информации за счет публикации сетевых брошюр
- Прием заказов
- Предоставление услуг или информационной продукции
- Предоставление дополнительных товаров и услуг
- Снижение расходов

Зачастую отдельный компонент Web-сайта может выполнять несколько из перечисленных функций. Ниже следует описание каждой из функций с указанием обычных способов применения с целью извлечения дохода или других целей.

Данный раздел книги призван помочь читателю сформулировать цели, преследуемые созданием Web-сайта: для чего он предназначается и каким образом каждая его функция будет содействовать расширению бизнеса.

Сетевые брошюры

В начале 90-х большинство Web-сайтов представляли собой сетевые брошюры или средства продаж. Сетевые брошюры и поныне остаются одним из наиболее популярных типов коммерческих Web-сайтов, удобным для первоначального набега на Web или недорогих упражнений в рекламе.

Брошюрный сайт может выполнять множество функций — от вывода в форме Web-страниц визитных карточек до предоставления обширной информации коммерческого характера. В любом случае, назначение такого сайта и смысл его существования — привлечение интереса клиентов к конкретному бизнесу.

Web-сайты подобного типа не приносят прямого дохода, но способствуют росту доходов, получаемых обычными средствами.

Разработка сайта упомянутого вида требует решения нескольких технических проблем, характерных, впрочем, и для других ситуаций. Речь идет о предотвращении следующих условий:

- Непредоставление важной информации
- Невыразительное представление
- Отсутствие реакции на обратную связь
- Несвоевременное обновление сайта
- Отсутствие отслеживания результатов работы

Непредоставление важной информации

Какие сведения понадобятся посетителю данного сайта? В зависимости от того, что ему уже известно, он может затребовать подробное описание товара, но может довольствоваться телефонами и адресами компании.

На многих сайтах полезная либо наиболее важная информация в принципе отсутствует. Минимум сведений, которые необходимо предоставить пользователю — это подсказки в принятии решений, географическая принадлежность предоставляемых служб, а также адреса и номера телефонов.

Невыразительное представление

"В Internet никто не увидит, что ты пес" — гласит старая поговорка [1]. Однако если мелкие предприятия ("псы") могут выглядеть в Internet весьма представительно, крупные зачастую бывают представлены на невыразительных Web-сайтах как мелкие и непрофессиональные.

Web-сайт любой компании должен соответствовать самым высоким стандартам. Текст должен быть написан и вычитан профессионалами, прекрасно в владеющими языком, графика — отчетливой, выразительной и быстро загружаемой. Вообще говоря, на коммерческом сайте следует очень тщательно отнестись к выбору графики и цветов, чтобы они соответствовали тому образу компании, который предстоит создать. Особая осторожность требуется в выборе анимации и звука; впрочем, без крайней необходимости их и вовсе не следует использовать.

Конечно, страницы сайта не могут выглядеть совершенно одинаково на любых машинах, во всех операционных системах и браузерах; тем не менее, они должны быть доступными для подавляющего большинства пользователей и выводиться на экран без ошибок.

Отсутствие реакции на обратную связь

В Web, как и в реальном мире, хорошее обслуживание важно ничуть не меньше, чем привлечение и удержание клиентов. Многие компании — как мелкие, так и крупные — грешат тем, что не отвечают своевременно и по существу на запросы клиентов, отправленные по адресам, указанным на Web-страницах. Клиент же, отправив сообщение по электронной почте, рассчитывает на более скорый ответ, нежели при обмене традиционными почтовыми отправлениями. Поэтому, чтобы не обидеть кого-либо невниманием, обработкой почты следует заниматься ежедневно.

Адреса электронной почты, указываемые на Web-страницах, должны принадлежать компании и ее подразделениям, но никак не отдельным сотрудникам. Иначе кто будет заниматься почтой, адресованной **fred.smith@company.com** после того, как Фред уволится? У сообщений, адресованных на **sales@company.com,** существенно больше шансов попасть к тому, кто придет на его место. К тому же таким сообщением может заняться сразу несколько специалистов, что увеличивает шансы благоприятного разрешения проблемы.

Несвоевременное обновление сайта

Необходимо внимательно следить за тем, чтобы информация, представленная на сайте, не устаревала. Содержимое страниц должно периодически обновляться, клиентам следует предоставлять сведения обо всех организационных изменениях в компании. Слишком запутанный сайт способен отвадить клиентов, поскольку не создает уверенности в корректности предоставляемой информации.

Во избежание застоя необходимо своевременно обновлять страницы либо формировать их динамически при помощи какого-либо языка сценариев — например, PHP. В этом случае достаточно запрограммировать в сценариях обращение к обновляемым источникам информации.

---

[1] Конечно, в отношении Internet "старая поговорка" звучит несколько двусмысленно. Это подпись под карикатурой Питера Стайнера (Peter Steiner), опубликованной впервые в номере журнала "Нью-Йоркер" от 5 июля 1993 г.

Отсутствие отслеживания результатов работы

Создание Web-сайта — только половина дела. Необходимо еще окупить затраченные усилия и финансовые средства. В частности, если сайт принадлежит крупной компании, ее руководство рано или поздно потребует отчета о том, насколь неоценим его вклад в процветание предприятия.

В маркетинговых мероприятиях, осуществляемых традиционными способами, крупные компании тратят десятки тысяч долларов на изучение рынка — причем как до начала этих мероприятий, так и после их завершения, чтобы иметь возможность судить об эффективности проделанной работы. Эта схема вполне пригодна и для оценки Web-проектов — по крайней мере, сравнительно крупных и обеспеченных солидным бюджетом.

Возможны также более простые и недорогие варианты:

Изучение журналов сервера. На Web-серверах хранится множество сведений о выполнении запросов. Большая часть этой информации совершено бесполезна, к тому же представлена в таком виде, что найти в ней что-либо осмысленное практически невозможно. Чтобы выудить из этого изобилия что-либо полезное, необходим анализатор журналов. Существуют две популярные программы такого рода, распространяемые бесплатно: Analog (загружается из сайта http://www.statslab.cam.ac.uk/~sretl/analog) и Webalizer (http://www.mrunix.net/webalizer). Можно также воспользоваться более совершенной коммерческой программой наподобие Summary (**http://www.summary.net**). Анализаторы журналов позволяют определять зависимость трафика от времени и посещаемость отдельных страниц.

Мониторинг продаж. Назначение сетевых брошюр заключается в способствовании росту продаж. Чтобы оценить, насколько успешно они справляются с этой задачей, достаточно сравнить уровень продаж до и после запуска Web-сайта. Разумеется, отчетливый результат трудно получить на фоне других маркетинговых мероприятий.

Организация обратной связи. Об отношении пользователей к Web-сайту можно узнать у самих пользователей, поместив на страницы формы обратной связи или указав соответствующие адреса электронной почты. Обсуждение можно стимулировать, установив для его участников небольшое поощрение — скажем, участие в какой-то лотерее.

Изучение типового пользователя. Надежную оценку сайта или даже прототипа можно выполнить при помощи целевых групп добровольцев. Добровольцам предлагается оценить сайт, затем их мнения и впечатления фиксируются в процессе интервью.

Если исследование методом целевых групп поручить профессионалу, способному оценить демографический и личностный срез будущего сообщества пользователей, а также профессионально проинтервьюировать всех участников эксперимента, это может потребовать немалых средств. Расходы можно свести к нулю, доверив работу непрофессионалу, но в этом случае трудно гарантировать адекватность моделируемых условий и предстоящей реальности.

Впрочем, подключение к эксперименту компании, занимающейся исследованиями рынка — не единственный вариант получения достоверных результатов методом целевых групп. Можно организовать собственные группы под руководством опытного модератора, умеющего работать с людьми и свободного от предубеждений. В

группу следует включать от шести до десяти человек. Чтобы модератор мог полностью сосредоточиться на своей работе, в помощь ему следует выделить протоколиста или секретаря. Точность результатов будет зависеть от того, насколь удачно подобран состав групп. Группа, составленная из друзей или сотрудников, вряд ли окажется особо полезной.

Прием *заказов на товары и услуги*

Если реклама в сети организована основательно, следующий шаг — предоставление пользователям возможности размещать заказы, не покидая сеть. Обычным продавцам прекрасно известно, как важно побудить клиента принять решение не откладывая. Чем больше человек размышляет, тем больше вероятность того, что он отложит покупку. Следовательно, в интересах продавца обслужить его как можно быстрее. Необходимость отключить модем и связаться с продавцом по телефону, а то и лично появиться в магазине — это препятствие на пути к завершению сделки. Поэтому, имея работоспособную систему рекламирования в сети, почему бы не дополнить ее системой совершения сделки купли-продажи!

Прием заказов через Web очень удобен для многих видов бизнеса. В конце концов, в заказах нуждается любое предприятие, поэтому предоставление клиентам возможности делать это через сеть если и не увеличит число последних, то, по меньшей мере, снизит нагрузку на продавцов. Естественно, это потребует создания динамического сайта со средствами приема платежей и, как следствие, дополнительных расходов, поэтому неплохо было бы оценить, насколько товары, которые предстоит продавать, удобны для представления при помощи средств электронной коммерции.

Товары, продаваемые через Internet — это, главным образом, книги и журналы, компьютеры и программное обеспечение, музыкальные записи, одежда, туристические путевки и билеты на развлекательные мероприятия.

Впрочем, не стоит отчаиваться, не увидев в приведенном перечне своего товара - здесь уже основательно утвердились многие солидные компании. Разумнее будет рассмотреть факторы, которые поспособствовали продвижению указанных товаров на сетевой рынок.

Товар для электронной коммерции должен быть нескоропортящимся, удобным в доставке; он должен быть достаточно дорог, чтобы стоимость доставки не составляла существенной добавки к цене, но не настолько, чтобы покупатель не мог решиться выложить деньги, не убедившись воочию в реальности покупки.

Для электронной коммерции больше подходят промышленные товары. Покупая, скажем, авокадо, мы стараемся выбрать лучшие плоды и не прочь иногда пощупать каждый. Ведь они не все одинаковы. Что же касается книг, CD-дисков или компьютерных программ, то здесь изделия одного наименования неотличимы друг от друга. Покупателю нет надобности ощупывать покупку.

Еще одно требование к товарам электронной коммерции: они должны быть ориентированы на пользователей Internet. Во время писания этой книги в данную категорию входили трудоустроенные молодые люди, имеющие доходы выше среднего и про-

---

[2] Use of Internet by Householders, Австралия, февраль 2000 г. (каталог № 8147.0), Австралийское бюро статистики.

живающие в довольно крупных городах [2]. Впрочем, со временем в категорию пользователей Internet войдет все население.

Существуют товары, которые ни разу не упоминались в отчетах об исследовании рынка электронной коммерции, но, тем не менее, достаточно перспективные в отношении упомянутого способа продаж. К примеру, Internet может оказаться идеальным средством продажи товара, ориентированного на определенный сегмент рынка.

Многие товары совершенно непригодны для электронной коммерции. Таковыми являются дешевые и скоропортящиеся изделия, например, бакалея. Ряд компаний все же пытались организовать их продажу через сеть, но без особого успеха. Существуют также товары, которые удобно рекламировать через сетевые брошюры, но продавать обычным образом. Это крупные и дорогостоящие изделия — автомобили, недвижимость и т.п. Такие покупки не делаются заочно и без тщательного взвешивания множества вариантов; кроме того, дело осложнятся проблемами доставки.

Ниже перечислены обстоятельства, которые могут помешать перспективному покупателю разместить заказ:

- Отсутствие ответов на заданные вопросы
- Недоверие
- Неудобство в использовании
- Сочетаемость

Любое из этих обстоятельств способно помешать покупателю сформировать заказ.

Вопросы, оставшиеся без ответа

Если потенциальный покупатель не найдет ответа на один из своих вопросов, он, по всей вероятности, покинет сайт без покупки. Из этого можно сделать несколько выводов. Прежде всего, Web-сайт должен быть хорошо организован, так, чтобы покупатель, посетивший его впервые, мог без труда найти все, что его интересует. Далее, посетителю необходимо предоставить всю информацию, которая только может потребоваться, но таким образом, чтобы уберечь его от интеллектуальных перегрузок. Пользователи Web не склонны к вдумчивому чтению — они, скорее, предпочитают беглый просмотр. Следовательно, тексты должны быть лаконичными. Существуют оценки максимально допустимых объемов размещаемой информации, эмпирически определенные для каждого из носителей рекламы. В отношении Web-сайтов действуют несколько иные правила. Здесь важны, главным образом, два параметра. Первый — стоимость сбора и обновления информации, и второй — разумное ее размещение, структурирование и снабжение ссылками, которые вместе позволят пользователю быстро и без труда во всем разобраться.

Web-сайт подобен продавцу, не требующему заработной платы и отдыха. Это, однако, не освобождает от необходимости правильно организовать обслуживание клиентов. Следует всячески поощрять вопросы от посетителей и отвечать на эти вопросы незамедлительно по телефону, электронной почте или иным средствам связи.

Доверие

Если посетителю Web-сайта не известна торговая марка представленной в нем компании, с какой стати он будет этой компании доверять? Ведь создать представительный сайт может кто угодно. Конечно, отсутствие доверия не может помешать чтению сетевой брошюры, но совсем иное дело — оформление заказа. В последнем случае юга-

енту совершенно необходимо знать, с кем он имеет дело — с организацией, заслуживающей уважения, или с уже упоминавшимся "псом".

Совершая покупки в сети, клиенты зачастую бывают озабочены рядом обстоятельств:

Как будет использоваться предоставленная клиентом персональная информация? Не станет ли она доступной для посторонних? Не будет ли использована для назойливой рекламы? Надежно ли будет храниться? Клиент должен быть уверен, что никто не злоупотребит его доверием, и как раз это называется политикой конфиденциальности. Описание этой политики должно быть легко доступно.

Достойный ли бизнес представлен на сайте? Если предприятие зарегистрировано в соответствующем учреждении, имеет физический адрес и телефонный номер, существует уже в течение нескольких лет, вполне можно рассчитывать на то, что оно не окажется очередными "рогами и копытами", обладающими лишь несколькими Web-страницами и, как максимум, почтовым ящиком. Перечисленные выше сведения обязательно должны быть представлены на страницах Web-сайта.

Что делать покупателю, недовольному покупкой? Каковы условия возврата денег недовольному покупателю? Кто в этом случае оплачивает расходы по доставке? До сих пор условия возврата покупки, заказанной по почте, были более либеральными, чем в случае приобретения непосредственно в магазине. Зачастую гарантируется безусловный возврат. Следует оценить рост расходов на возврат покупок и сравнить его с доходами за счет дополнительных клиентов, привлеченных либеральной политикой возврата. Каковой бы ни была эта политика, ее описание необходимо разместить на страницах сайта.

Может ли клиент доверить компании сведения о своей кредитной карточке? Именно это вызывает наибольшие опасения у посетителей Web-магазинов, а потому необходимо не только обеспечить надежную защиту информации при обработке кредитных карточек, но также продемонстрировать серьезное к этому отношение. Как минимум, сведения о карточках должны передаваться из браузера на сервер с использованием SSL (Secure Sockets Layer — протокол защищенных сокетов); на самом же сервере необходимо обеспечить должный уровень администрирования с соблюдением режима секретности. В последующих главах мы обсудим это более подробно.

Удобство использования

Клиенты различаются по уровню как компьютерной, так и общей грамотности, говорят на разных языках, не все обладают крепкой памятью и острым зрением. Отсюда следует, что Web-сайт должен быть по возможности простым. О проектировании пользовательского интерфейса сказано и написано немало; тем не менее, вот еще несколько небесполезных советов:

Web-сайт должен быть, по возможности, простым. Чем больше опций, рекламы и прочих отвлекающих элементов будет представлено на каждой странице, тем больше вероятность того, что пользователь во всем этом запутается.

Текст должен быть отчетливым. Не следует применять причудливые шрифты. Текст не должен быть слишком мелким; необходимо учитывать зависимость его размера от типа экрана.

Следует максимально упростить процесс формирования заказа. Как подсказывает интуиция и свидетельствует опыт, чем больше щелчков приходится делать кли-

енту для осуществления заказа, тем меньше вероятность того, что он выдержит эту процедуру до конца. Необходимо свести к минимуму количество операций, держа однако в уме патент США[3], выданный на операцию, требующую всего одного щелчка и именуемую 1-Click. Этот патент активно оспаривается владельцами многих Web-сайтов.

Постарайтесь не дать пользователям запутаться. Снабжайте страницы ориентирами и подсказками, по которым пользователь смог бы определять свое местоположение. Выделяйте цветом ссылки, которыми клиент успел воспользоваться.

Если клиенту предоставляется виртуальная покупательская тележка, в которую он "складывает" покупки, ссылка на нее должна присутствовать на экране в любой момент времени.

Совместимость

Web-сайт обязательно должен быть протестирован на браузерах различных типов и под управлением разных операционных систем. Если сайт не сумеет взаимодействовать с какими-либо браузером или операционной системой, то это не только будет свидетельствовать о том, что он сработан непрофессионально, но и приведет к потере части потенциальных клиентов.

Если сайт уже запущен в работу, для определения типов используемых клиентами браузеров можно прибегнуть к информации из журналов сервера. Опыт показывает, что для ПК с операционной системой Microsoft Windows достаточно тестировать Web-сайт на двух последних версиях браузеров Microsoft Internet Explorer и Netscape Navigator, для Apple Mac — на двух последних версиях Netscape Navigator, для Linux — на последней версии Netscape Navigator. Неплохо также выполнить тестирование в текстовом браузере Lynx.

Новейшие функции и средства следует применять только в том случае, если предполагается создание нескольких версий Web-сайта.

Предоставление услуг и цифровой продукции

Многие товары и услуги продаются через Web, но доставляются клиенту курьером, другие могут доставляться незамедлительно через сеть. Если услуга или товар могут быть переданы через модем, их заказ, оплата и доставка происходят незамедлительно, без вмешательства человека.

Наиболее характерный пример подобной продукции — информация. Часто она предоставляется бесплатно либо оплачивается за счет рекламы, иногда — по подписке; возможен и обычный способ оплаты — за единицу продукции.

К цифровой продукции относятся электронные книги и музыкальные записи в цифровых форматах наподобие MP3, изобразительная продукция из библиотечных фондов, переведенная в цифровую форму, а также программное обеспечение, которое также необязательно распространять только в виде записей на CD-ROM. Все это можно выгружать непосредственно из Internet.

---

[3] US Patent and Trademark Office Patent Number 5,960,411. Method and system for placing a purchase order via communications network (Патент США и право обладания торговой маркой № 5,960,411. Метод и система размещения заказа на приобретение товара через коммуникационную сеть.)

Среди услуг, предоставляемых таким способом — доступ к Internet, предоставление и обслуживание Web-сайтов, а также некоторые профессиональные службы, которые могут заменять экспертные системы.

Физическая доставка товара, заказанного через Web-сайт, обладает по сравнению с выгрузкой цифровой продукции как преимуществами, так и недостатками.

Физическая доставка требует расходов, в то время как стоимость выгрузки через Web практически равна нулю. Это означает, что расходы на доставку единицы цифровой продукции мало отличаются от расходов на доставку тысячи единиц. Конечно, это верно лишь до определенного предела, пока не потребуется установка дополнительного оборудования с целью увеличения пропускной способности.

Цифровая продукция, как и цифровые услуги, хорошо продаются как импульсивные покупки. Человек, заказавший физический товар, получит его через некоторое время — возможно, через несколько дней; выгрузка через Web длится несколько секунд или минут, но именно это и составляет проблему. Необходимость незамедлительной доставки не позволяет распределять эти операции с учетом суточных колебаний нагрузки. Результат — недостаточная защита от компьютерных мошенников и неэффективное использование ресурсов.

Цифровая продукция и цифровые услуги идеально подходят для электронной коммерции, но они составляют лишь ограниченную часть всех товаров и услуг.

Дополнительные товары и услуги

Существуют области успешного коммерческого использования Web, не связанные непосредственно с продажей товаров или услуг. Например, компании экспресс-доставки почты UPS **(www.ups.com)** и Fedex **(www.fedex.com)** предлагают службы слежения, непосредственно не предназначенные для извлечения прибыли, но составляющие добавочную стоимость основных услуг. Позволяя клиентам отслеживать состояние сделок и банковских счетов, компании получают преимущество перед конкурентами.

В эту же категорию входят дискуссионные группы для пользователей. Создание таких групп, в которых клиенты могли бы обсуждать проблемы, возникающие при пользовании изделиями компании, имеет надежное экономическое обоснование. Обмен опытом позволяет решать множество проблем, к тому же, подобная организация технической поддержки позволяет клиентам экономить на оплате дальней телефонной связи и никак не связана рамками рабочего времени. Что же касается компании, для нее эти группы — наиболее дешевый способ удовлетворения претензий клиентов.

Снижение расходов

Один из наиболее очевидных результатов использования Internet связано со снижением расходов, достигаемым за счет распространения информации по сети, более оперативной и дешевой связи, замены служб и централизации.

Web-сайт — наиболее экономичное средство доставки информации большому числу людей. Прейскуранты, каталоги, документированные процедуры, спецификации и Бог знает что еще значительно дешевле публиковать на Web-сайте, нежели печатать и рассылать бумажные копии, особенно, в условиях частого обновления информации. Что касается коммуникационных возможностей Web, то их можно использовать для распределения тендеров с быстрым получением результатов, для связи клиентов с оптовыми продавцами или изготовителями (минуя посредников) и т.п. В любом случае результат будет один: снижение расходов и рост прибыли.

Снижению расходов способствует также замена традиционных служб электронными. Пример смелого решения — сайт Egghead.com. Эта компания закрыла свою сеть

компьютерных магазинов и полностью сосредоточилась на электронной коммерции. Конечно, организация основательного сайта электронной коммерции стоит немалых денег, но это все же не идет ни в какое сравнение с расходами на поддержку более 70 розничных торговых точек. Несомненно, замена существующей службы связана с риском: теряются, как минимум, клиенты, не использующие Internet.

Снижению расходов способствует и централизация. Если компания имеет несколько офисов, расположенных на удалении друг от друга, ей приходится вносить многочисленные платежи за аренду, нести накладные расходы, содержать многочисленный штат сотрудников и обеспечивать оснащение рабочих мест. Предприятие электронного бизнеса может располагаться в одном месте и в то же время быть доступным из любой точки земного шара.

Риски и угрозы

Любой бизнес связан с риском вследствие конкуренции, воровства, неустойчивости общественных предпочтений, стихийных бедствий — список можно продолжать до бесконечности. Риски, связанные с электронной коммерцией, имеют свои особенности и источники, среди которых:

- Взломщики
- Невозможность привлечения компаньонов
- Отказы оборудования
- Сбои питания, коммуникационных линий или сети
- Зависимость о служб доставки
- Интенсивная конкуренция
- Сбои программного обеспечения
- Изменения в политике и налогообложении
- Ограниченная пропускная способность системы

ВЗЛОМЩИКИ

Наиболее популяризованная угроза электронной коммерции исходит от компьютерных злоумышленников — взломщиков. Любое предприятие подвергается угрозе незаконного нападения, крупные же предприятия электронной коммерции привлекают внимание компьютерных взломщиков разного уровня квалификации.

Причины этого внимания различны. В одних случаях это просто "спортивный" интерес, в других — жажда геростратовой славы, желание навредить, попытки хищения денег либо бесплатного приобретения товаров и услуг.

Защита сайта обеспечивается сочетанием следующих мер:

- Резервное копирование важной информации.
- Кадровая политика, позволяющая привлекать к работе только добросовестных людей и стимулировать добросовестность персонала. Наиболее опасны попытки взлома, исходящие изнутри компании.
- Использование программного обеспечения с возможностями защиты данных и своевременное его обновление.
- Обучение персонала определению целей и распознаванию слабых мест системы.
- Аудит и ведение журналов для обнаружения успешных и неудачных попыток взлома.

Как правило, взлом удается из-за легко угадываемого пароля, распространенных ошибок в конфигурации и несвоевременного обновления версий программного обеспечения. Для защиты от не слишком изощренного взломщика достаточно принятия относительно простых мер; на крайний случай, всегда должна быть резервная копия критичных данных.

Невозможность привлечения компаньонов

Атаки взломщиков вызывают наибольшие опасения, однако реальная опасность для электронной коммерции имеет чаще всего экономическую основу. Создание и маркетинг крупного сайта электронной коммерции требуют немалых средств. Компании предпочитают краткосрочные инвестиции, предполагая немедленный рост числа клиентов и доходов после утверждения торговой марки на рынке.

Во время написания этой книги Amazon.com (по некоторым оценкам, наиболее известный сайт розничных продаж) пятый год подряд работал в убыток. В первом квартале 2000 г. его убытки составили 99 миллионов долларов США. В цепи крупных провалов стоит упомянуть европейский филиал boo.com, сменивший владельца после потери за полгода почти 120 миллионов долларов США. При этом причина несчастий была не в недостаточном объеме продаж, просто компания тратила больше, чем зарабатывала.

Отказы оборудования

Совершенно очевидно, что отказ важной части одного из компьютеров компании, деятельность которой сосредоточена в Web, может нанести ей существенный вред. Защита от простоя сайтов, работающих с высокой нагрузкой или выполняющих важные функции, обеспечивается дублированием систем, благодаря чему выход из строя любого компонента не приводит к полному останову. Однако и здесь необходимо оценить потери от возможных простоев в сравнении с расходами на приобретение дополнительного оборудования.

Сбои питания, коммуникационных линий или сети, отказы службы доставки

Зависимость от Internet означает зависимость от множества взаимосвязанных поставщиков услуг, поэтому, если связь с остальным миром вдруг обрывается, не останется ничего иного, как только ждать ее восстановления. Та же ситуация с перебоями в электропитании и забастовками (или иными неприятностями) может возникнуть в компании, занимающейся доставкой.

При достаточном финансировании можно работать с несколькими поставщиками услуг. Это влечет дополнительные расходы, однако обеспечивает бесперебойность работы в условиях отказа одного из них. От кратких перебоев в электропитании можно защититься установкой источников бесперебойного питания.

Интенсивная конкуренция

В случае открытия киоска на улице оценка конкурентной среды не составляет труда — конкурентами будут все, кто торгует тем же товаром в пределах видимости. В случае электронной коммерции ситуация несколько сложнее. В зависимости от расходов на доставку, а также при условии колебаний курсов валют и различий в стоимости рабочей силы, конкуренты могут располагаться где угодно. Internet — в высшей степени конкурентная и активно развивающаяся среда. В популярных отраслях бизнеса новые конкуренты возникают почти ежедневно.

Риск, исходящий от конкуренции, с трудом поддается оценке. Здесь наиболее верная стратегия — поддержка современного уровня технологии.

Сбои программного обеспечения

Еще одна угроза для электронного бизнеса — сбои программ. Количество критических сбоев можно свести к минимуму за счет установки надежного программного обеспечения, тщательного тестирования после каждого случая замены неисправного оборудования и применения формальных процедур тестирования. Очень важно сопровождать тщательным тестированием любые нововведения в систему.

Для уменьшения вреда, наносимого сбоями программного обеспечения следует своевременно создавать резервные копии всех данных; при внесении каких-либо изменений необходимо сохранять прежние конфигурации программ; для быстрого обнаружения возможных неисправностей — вести постоянный мониторинг системы.

Изменения в политике и налогообложении

Во многих странах деятельность в сфере электронного бизнеса не определена (либо недостаточно определена) законодательно. Однако такое положение не может сохраняться вечно, и урегулирование вопроса породит ряд проблем, способных повлечь закрытие некоторых предприятий. К тому же всегда существует опасность повышения налогов.

В этой ситуации единственной разумной линией поведения будет внимательное отслеживание ситуации и приведение деятельности предприятия в соответствие с законами. Следует также изучить возможность лоббирования собственных интересов.

Ограниченная пропускная способность системы

На этапе проектирования системы обязательно следует предусмотреть возможности роста. Успех неразрывно связан с нагрузками, поэтому система должна допускать наращивание оборудования.

Ограниченного роста производительности можно достичь заменой оборудования, однако скорость даже самого совершенного компьютера имеет предел, поэтому в программном обеспечении должна быть предусмотрена возможность при достижении указанного предела распределять нагрузку на несколько систем. Например, СУБД должна обеспечивать одновременную обработку запросов от нескольких машин.

Наращивание любой системы не проходит безболезненно, однако своевременное ее планирование позволяет предвидеть многие неприятности и заранее их предупреждать.

Выбор стратегии

Многие полагают, что бурный рост Internet исключает возможность эффективного планирования. Мы придерживаемся противоположной точки зрения: планирование крайне необходимо именно ввиду изменчивости Internet. Определение целей и выбор стратегии позволяет предвидеть будущие изменения и реагировать на них с опережением.

Итак, мы разобрались с целями создания коммерческого Web-сайта и теперь в состоянии разработать собственную стратегию. В этой стратегии должна быть определена модель бизнеса. Зачастую модель — это нечто давно известное, но воспринимаемое нами как свежая и гениальная идея. Воспользоваться ли идеями, уже апробированными в Web, или идти собственным путем — это каждый должен решать для себя сам.

Что дальше

В следующей главе мы рассмотрим вопросы безопасности электронной коммерции: терминологию, угрозы и методы защиты.

Глава 13

Вопросы безопасности в электронной коммерции

В этой главе обсуждается роль безопасности в электронной коммерции. Мы обсудим, кто заинтересован в получении вашей информации, как ее можно получить, принципы создания стратегии, которая позволит избежать проблем подобного рода, а также некоторые технологии для гарантированной защиты Web-сайта, в числе которых шифрование, аутентификация и отслеживание.

В этой главе рассматриваются следующие темы:

- Насколь важна ваша информация?
- Угрозы безопасности
- Разработка стратегии защиты
- Достижение баланса между практичностью, производительностью и защищенностью
- Принципы аутентификации
- Использование аутентификации
- Основы шифрования
- Шифрование с закрытым ключом
- Шифрование с открытым ключом
- Цифровые подписи
- Цифровые сертификаты
- Защищенные Web-серверы
- Аудит и регистрация
- Брандмауэры
- Резервное копирование данных
- Физическая защита

Насколь важна ваша информация?

При рассмотрении средств защиты первое, что потребуется оценить — это важность того, *что* вы пытаетесь защитить. Необходимо принимать во внимание две стороны — важность для вас и важность для потенциальных взломщиков.

Очень соблазнительно думать, что наивысшая степень защиты необходима всегда и везде для всех сайтов, однако безопасность имеет свою цену. Перед тем как вы решите, сколько расходов и усилий оправдывают ваши средства защиты, неплохо было бы решить, сколько стоит ваша информация.

Очевидно, что ценность информации на компьютерах пользователя-любителя, банка и военной организации существенно различается. Точно так же есть различия, насколь далеко может зайти атакующий для получения доступа к этой информации. Насколько притягательно для взломщика содержимое вашего компьютера?

Вероятно, пользователи-любители располагают очень ограниченным временем, чтобы изучить защиту или поработать над защитой собственных систем. Учитывая, что на компьютерах любителей, как правило, хранится информация, ценность которой не велика для всех за исключением владельца, то атаки на такие системы будут нечастыми и ограниченными по усилиям. Однако пользователи подключенных к сети компьютеров должны предпринять значительные меры предосторожности. Даже если данные на компьютере не представляют никакого интереса, этот компьютер может быть использован как анонимная стартовая площадка для атак на другие системы.

Военные компьютеры являются очевидной целью для атак как индивидуалов, так и иностранных правительств. Поскольку атакующие правительства могут обладать внушительными ресурсами, благоразумно направить дополнительные инвестиции в персонал и другие ресурсы, дабы иметь гарантию, что в данном домене предприняты все меры предосторожности.

Если вы отвечаете за сайт электронной коммерции, то его привлекательность для взломщиков находится приблизительно посередине между двумя крайностями, описанными выше.

Угрозы безопасности

Чем вы рискуете на вашем сайте? Какие существуют угрозы извне?

Некоторые из угроз бизнесу в сфере электронной коммерции обсуждались в главе 12. Множество из этих угроз относятся к угрозам безопасности.

В зависимости от вашего Web-сайта, угрозы безопасности могут включать следующие моменты:

- Вскрытие конфиденциальных данных
- Потеря или уничтожение данных
- Изменение данных
- Отказ в обслуживании
- Ошибки программного обеспечения
- Отказ от обязательств

Давайте рассмотрим каждую из этих угроз.

Вскрытие конфиденциальных данных

Данные, хранимые на вашем компьютере или передаваемые на или с вашего компьютера, могут быть конфиденциальными. К ним относится информация, которую разрешено видеть только определенным людям, например, списки оптовых цен. Это может быть конфиденциальная информация, предоставленная клиентом, например, пароль, контактная информация или номер кредитной карточки.

Надеемся, что вы не храните на Web-сервере информацию, которая не предназначена для просмотра всеми посетителями сайта. Web-сервер — это неподходящее место для секретной информации. Если вы собираетесь сохранить на компьютере платежную ведомость или план захвата власти над всем миром, то благоразумнее Web-сервер для этого не использовать. Web-сервер по определению является общедоступным компьютером и должен содержать либо информацию, которую следует предоставить клиентам, либо информацию, только что собранную от клиентов.

Для снижения риска раскрытия данных потребуется ограничить число методов получения доступа к информации и количество людей, которые могут получить такой доступ. Этого можно добиться, если проектировать, памятуя о защите, должным образом конфигурировать сервер и серверные приложения, осторожно программировать, тщательно тестировать, удалять ненужные службы из Web-сервера и требовать аутентификации посетителей.

Следует тщательно проектировать, настраивать, кодировать и тестировать, чтобы снизить риск успешной криминальной атаки, и, что не менее важно, уменьшить шанс того, что ошибка оставит вашу информацию открытой для случайного чтения.

С целью уменьшения количества потенциальных слабых мест удалите ненужные службы с Web-сервера. Каждая служба, запущенная на сервере, обладает своими уязвимыми местами. Каждая служба должна обновляться, чтобы гарантировать, что известные всем уязвимости не присутствуют на вашем сервере. Службы, которые не используются, могут оказаться еще более опасными. Если вы никогда не используете команду rcp, то зачем устанавливать эту службу?[1] Если в процессе инсталляции вы укажете, что компьютер подключен к сети, то программа инсталляции, включенная в основные дистрибутивы Linux и Windows NT, установит на ваш компьютер массу служб, которые вам не нужны и должны быть удалены.

Аутентификация (authentication) означает требование посетителей доказать свою личность. Если система знает, кто делает запрос, она может решить, разрешен ли данной персоне доступ. Существует несколько возможных методов аутентификации, но в большинстве случаев применяются только пароли и цифровые подписи (digital signatures). Более подробно эти методы обсуждаются ниже.

Компания CD Universe представляет собой хороший пример стоимости раскрытия конфиденциальной информации как в денежном отношении, так и в отношении репутации. Как сообщалось, в конце 1999 г.. взломщик, называющий себя Maxus, связался с компанией CD Universe, утверждая, что обладает номерами 300000 кредитных карточек, которые были украдены с сайта компании. Он хотел получить выкуп в размере 100000 долларов США за то, что уничтожит номера кредитных карточек. Компания решила не платить и оказалась втянутой в запутанную историю, попавшую на первые полосы большинства газет, поскольку упомянутый Maxus раздал номера мелкими партиями другим людям для нелегального использования.

---

[1] Даже если вы используете rcp в данный момент, вероятно, следует удалить эту утилиту и применять вместо нее утилиту scp (secure copy — защищенное копирование).

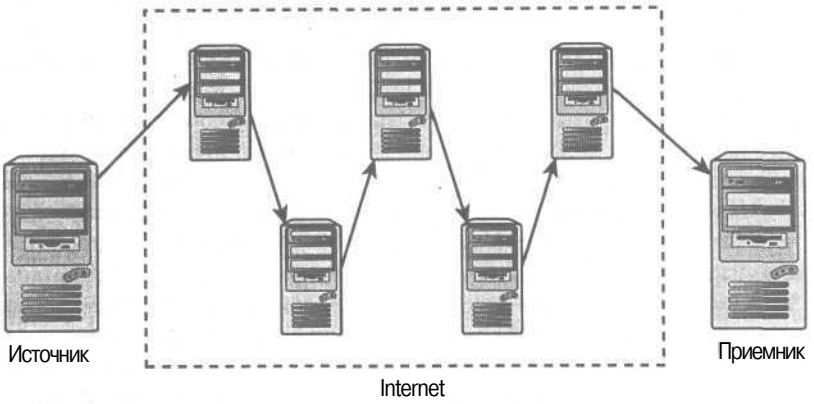


РИСУНОК 13.1 *Передача информации через Internet связана с переправкой ваших данных через множество потенциально уязвимых узлов.*

Данные подвергаются риску раскрытия также и во время прохождения по сети. Хотя сети на базе TCP/IP имеют множество хороших возможностей, которые сделали их стандартом де-факто для объединения различных сетей в Internet, защита в число этих хороших возможностей не входит. Протокол TCP/IP работает, разбивая ваши данные на пакеты и пересылая эти пакеты от одного компьютера к другому, пока пакеты не достигнут требуемой точки назначения. Это означает, что на пути ваши данные проходят через множество компьютеров, как показано на рис. 13.1. Любой из этих компьютеров имеет возможность просмотреть данные в момент прохождения через него.

Чтобы увидеть путь, по которому проходят данные от вашего компьютера в точку назначения, воспользуйтесь командой **traceroute** на UNIX-компьютерах или **tracert** на Windows-компьютерах. Эта команда выдаст адреса компьютеров, через которые проходят ваши данные на пути к узлу назначения. Для достижения узла в отдельной стране данные обычно должны пройти до десятка разных компьютеров. В случае узлов, находящихся вне вашей страны, может оказаться более 20 промежуточных узлов. Если ваша организация владеет сложной сетью, данные могут пройти только через пять узлов, перед тем как покинуть пределы здания.

Для защиты конфиденциальных данных можно зашифровать перед отправкой и расшифровать после получения. Web-серверы часто используют протокол Socket Secure Layer (SSL), разработанный компанией Netscape, тем самым обеспечивая безопасное взаимодействие для данных, путешествующих между Web-сервером и браузерами. Это достаточно дешевый и не требующий больших усилий способ защиты канала передачи, но поскольку вместо простой отправки и приема данных сервер должен еще их зашифровывать и расшифровывать, то существенно снижается количество посетителей в секунду, которых может обслуживать сервер.

Потеря или разрушение данных

Потеря данных может стоить значительно больше, чем просто их вскрытие. Если вы потратили месяцы на разработку сайта и сбор от пользователей данных и заказов, сколько в деньгах, репутации и времени, вам будет стоить потеря этой информации? Если у вас не было резервных копий данных, придется спешно переписать сайт и начать с самого начала.

Вероятно, взломщики все-таки *войдут* в вашу систему и отформатируют ваш жесткий диск. Очень похоже, что беззаботный программист или администратор случайно *удалит* что-то, и почти несомненно, что вы случайно *потеряете* жесткий диск.

Жесткие диски вращаются со скоростью тысяч оборотов в секунду, и время от времени все-таки отказывают. Согласно законам Мерфи, отказавший диск будет содержать самые важные данные, для которых дольше всего не выполнялось резервное копирование.

Во избежание потери данных, можно предпринять разнообразные действия. Защищайте свои серверы от взломщиков. Поддерживайте минимальным число людей, имеющих доступ к вашему компьютеру. Нанимайте только компетентных и внимательных сотрудников. Покупайте хорошие, качественные жесткие диски. Используйте массивы RAID с несколькими дисками. Такие массивы могут работать как один диск, но только более надежный.

Независимо от причин, существует только один реальный способ защиты от потери данных — резервное копирование. Резервное копирование — это не высшая математика. Наоборот, это утомительный, скучный и, наверно, неблагодарный процесс, однако он жизненно необходим. Удостоверьтесь, что данные регулярно копируются на резервные носители, и протестируйте процедуру резервного копирования, чтобы иметь уверенность в возможности восстановления данных. Также удостоверьтесь, что резервные копии хранятся за пределами ваших компьютеров. И хотя маловероятно, что ваш офис пострадает от пожара или еще какого-нибудь стихийного бедствия, хранение резервных копий вне офиса — это достаточно дешевая стратегия страховки.

Изменение данных

Хотя потеря данных может принести большой ущерб, не меньший, а то и больший ущерб случается по причине изменения данных. Что произойдет, если кто-то получил доступ к вашей системе и изменил файлы? Хотя полное удаление может быть замечено и исправлено с использованием резервной копии, как скоро вы сможете заметить несанкционированное изменение?

Изменение файлов может включать изменение данных и изменение выполняемых файлов. Мотивация взломщика при изменении файлов данных связана с его желанием "украсить" ваш сайт "настенными рисунками" или получить какие-либо привилегии. Замена выполняемых файлов их вредительскими версиями может открыть взломщику, получившему доступ к сайту, своего рода "черный ход" для последующих визитов.

Данные, которые передаются через сеть, можно защитить путем вычисления подписи (signature). Это не остановит злоумышленников от изменения данных, но если получатель проверит, что подпись все еще соответствует полученному файлу, он будет знать, что файл был изменен. Если во избежание несанкционированного просмотра данные шифруются, то это также существенно усложняет модификацию данных в пути без обнаружения.

Защита от изменений в файлах, хранящихся на сервере, требует применения предоставленных операционной системой возможностей контроля доступа к файлам. Воспользуйтесь этими возможностями для защиты системы от несанкционированного доступа. Используя механизм файловых разрешений, посетителям можно разрешить использовать систему, но не предоставлять им полный контроль над изменением системных файлов и файлов других пользователей. Отсутствие соответствующих систем разрешений — это одна из причин, почему Windows 98 и Windows 95 не подходят в качестве операционных систем для серверов.

Обнаружение изменений представляет собой сложную задачу. Если в какой-то момент становится понятно, что защита системы взломана, как можно определить,

изменены ли важные файлы или нет? Некоторые файлы, такие как файлы, хранящие базу данных, должны со временем изменяться. Множество других файлов должны оставаться неизменными с момента установки. Такие файлы меняются, только если вы сознательно обновляете их. Модификация файлов как программ, так и данных может осуществляться очень коварно, и хотя программы можно установить повторно, вы не сможете определить, какая из версий данных была "чистой".

Программное обеспечение определения целостности файлов наподобие Tripwire фиксирует информацию о важных файлах, которые находятся в известном безопасном состоянии, возможно даже немедленно после установки. Позже эта информация может быть использована для проверки целостности файла. Коммерческую и условно-бесплатную версию этого приложения можно выгрузить из сайта http://www.tripwire.com

Отказ в обслуживании

Одна из наиболее сложных с точки зрения защиты угроз — это угроза *отказа в обслуживании (Denial of Service, DoS)* Эта угроза появляется, когда чьи-то действия приводят к затруднению или невозможности для пользователей получить доступ к определенной службе. Кроме того, отказом в обслуживании считается задержка при доступе к критичной ко времени службе.

В начале 2000 г. произошло большое количество атак типа *распределенного отказа в обслуживании (Distributed Denial of Service, DDoS)* на известные и популярные Web-сайты. В список подвергнувшихся атакам сайтов входят Yahoo!, eBay, Amazon, E-Trade и Buycom. Упомянутые сайты приспособлены к потокам данных такого объема, о котором многие из нас могут только мечтать. И все-таки эти сайты уязвимы для атак DoS и были недоступны в течении нескольких часов. Хотя взломщики мало получают от полного отключения сайта, владелец сайта может терять деньги, время и репутацию.

Одна из причин, почему от атак подобного типа очень сложно защищаться, связана с многообразием методов атаки DoS. Список методов включает установку на целевом компьютере программы, которая будет использовать почти все процессорное время, *обратную массовую рассылку (reverse spamming)* либо применение одного из автоматизированных инструментов. Метод *обратной массовой рассылки (reverse spamming)* состоит в том, что кто-то инициирует поддельную *массовую рассылку (spam),* в которой в качестве отправителя указан атакуемый узел. Таким образом, целый сайт получит тысячи злых ответов, которые еще и потребуется обработать.

Существуют автоматизированные утилиты, позволяющие запустить распределенную атаку DoS на выбранный сервер. Даже не обладающий большими знаниями человек может исследовать несколько компьютеров на предмет известных уязвимостеи, получить доступ к какому-то из компьютеров и установить эту утилиту. Поскольку весь процесс является полностью автоматическим, атакующий может установить такую программу на узле меньше чем за пять секунд. Когда утилита установлена на достаточном количестве узлов, на все такие узлы передается команда "утопить" атакуемую цель потоком сетевых данных.

В общем, защищаться от атак DoS достаточно сложно. Однако, выполнив небольшие исследования, вы узнаете и закроете порты, используемые по умолчанию большинством утилит для атак DDoS. Ваш маршрутизатор может предоставлять механизмы, такие как ограничение процентной доли потока данных, использующего определенный протокол, например, ICMP. Значительно проще обнаружить узлы сети, которые используются для запуска атак других узлов, нежели защищать от атак собственный узел. Если каждый сетевой администратор смог бы сосредоточиться исключительно на наблюдении за собственной сетью, то исчезла бы проблема атак DDoS.

В связи с тем, что существует очень много возможных методов атак, единственный эффективный метод защиты — это наблюдение за нормальным потоком данных в сети и наличие группы экспертов, что позволит принять меры при появлении ненормальных отклонений.

Ошибки программного обеспечения

Существует вероятность, что в программном обеспечении, которое вы купили, получили или создали самостоятельно, присутствуют ошибки. Учитывая, что для разработки Web-проектов отводится очень мало времени, весьма и весьма вероятно, что в программном обеспечении имеются ошибки. Любой основанный на компьютерах бизнес предельно чувствителен к ошибкам программного обеспечения.

Ошибки в программах могут приводить к непредсказуемому поведению, включая недоступность службы, прорехи в защите, финансовые потери и просто плохое обслуживание клиентов.

Подобного рода ошибки связаны с несколькими причинами, которые придется найти. Общеизвестные причины включают неполные спецификации, ошибочные предположения, сделанные разработчиками, и неадекватное тестирование.

Неполные спецификации

Чем более разбросанной и неоднозначной будет документация по вашему проекту, тем вероятнее, что в результате конечный продукт будет содержать ошибки. И хотя может показаться излишним указывать в спецификации, что если кредитная карточка клиента отвергнута, то не следует отсылать ему счет, существует, по крайней мере, один крупнобюджетный сайт, который содержал такую ошибку. Чем меньше у ваших разработчиков опыта в создании систем, подобных разрабатываемой в данный момент, тем точнее должны быть спецификации.

Предположения, сделанные разработчиками

Проектировщики и программисты системы делают много предположений. Очень хочется надеяться, что они внесут в документацию все свои предположения, да и правильно сделают. Все-таки иногда люди делают некорректные предположения. Сюда входят предположения, что введенные данные будут правильными, не будут содержать необычных символов или размер данных не превысит определенного значения. Неверные предположения также могут включать предположения о времени, такие как вероятность выполнения двух конфликтующих действий в одно и то же время, или предположение о том, что процесс сложной обработки займет больше времени, чем решение простой задачи.

Подобные предположения легко не заметить, поскольку они всегда истинны. Взломщик может воспользоваться переполнением буфера, если программист сделал предположение о длине входных данных, или обычный пользователь может запутаться и оставить ваш сайт из-за того, что программист не предположил, что в имени пользователя может встретиться апостроф. Ошибки такого рода можно отыскать и исправить, используя комбинацию качественного тестирования и детального анализа кода.

Исторически сложилось, что прорехи в приложениях и операционных системах, используемые взломщиками, обычно связаны с переполнением буфера или состязательными условиями.

Некачественное тестирование

Практически невозможно протестировать все условия ввода данных на всех допустимых типах оборудования и во всех возможных операционных системах и пользовательских настройках. Это можно считать аксиомой для систем, работающих в Web.

На самом деле просто требуется хороший план тестирования, который обеспечит проверку всех функций приложения на представительском наборе распространенных типов компьютеров. Цель хорошо спланированного набора тестов должна состоять в том, чтобы, как минимум, один раз протестировать каждую строку кода созданного приложения. В идеале этот тест должен быть автоматизированным, чтобы существовала возможность без особых усилий запускать его на выбранном компьютере.

Наибольшая проблема тестирования заключается в том, что в нем нет ничего очаровательного, плюс тестирование нужно выполнять несколько раз. И хотя некоторым людям нравится ломать вещи, очень немногим понравится ломать одну и ту же вещь снова и снова. Исключительно важно вовлекать в тестирование не только разработчиков, но и посторонних. Одна из целей тестирования — найти ошибочные предположения, сделанные разработчиками. Новый человек, вероятнее всего, будет иметь другие предположения. И самое печальное, профессионалы редко горят желанием искать ошибки в собственных разработках.

Отказ от обязательств

Последний риск, который мы рассмотрим - • это *отказ от обязательств (repudiation)*. Название этого риска говорит само за себя. Примером из области электронной коммерции может послужить человек, который заказал товары на Web-сайте, а затем отказался санкционировать снятие денег с кредитной карточки. Также примером может послужить человек, который соглашается с чем-то в электронном письме, после чего он же требует, чтобы его письмо было забыто.

В идеальном случае финансовые операции должны обеспечивать обоим сторонам душевное спокойствие касаемо отказа от обязательств. Ни один из участников не должен отказываться от своей части операции, или, более точно, оба участника сделки должны последовательно и документально подтвердить действия другого участника для третьей стороны, такой как суд. На практике это редко случается.

Аутентификация предоставляет некоторую уверенность в том, с кем вы работаете. Если сертификат выдан авторитетным органом, то цифровой сертификат подлинности (digital certificate of authentication) может обеспечить большую уверенность.

Отправляемые каждой из сторон сообщения также необходимо защищать от подделки. Вы немногого добьетесь, доказав, что компания Corp Pty Ltd отправила вам сообщение. Еще потребуется доказать, что полученное вами* — это в точности то, что компания отослала. Как упоминалось ранее, цифровые подписи и шифрование существенно усложняют незаметное изменение ваших сообщений.

Для выполнения транзакций между сторонами, поддерживающими регулярные отношения, эффективным способом снижения количества отказов от обязательств являются цифровые сертификаты в комбинации с обменом зашифрованными или подписанными сообщениями. Такие методы неэффективны для одноразовых транзакций, например, начальной транзакции между сайтом электронной коммерции и неизвестным посетителем, обладающим кредитной карточкой.

Чтобы подтвердить перед посетителями свою добросовестность, занимающаяся электронной коммерцией компания должна быть готова предоставить органу сертификации, такому как VeriSign (http://www.verisign.com) или Thawte (http://

www.thawte.com), доказательства подлинности и несколько сотен долларов. Возжелает ли та же компания пренебречь посетителем, который отказывается получать сертификат подтверждения личности? Для транзакций с небольшими суммами торговцы, в основном, готовы смириться с определенной долей риска обмана или отказа от обязательств, дабы тем самым не терять потенциальных покупателей.

Начиная с 1997 г., альянс, включающий VISA и ряд других финансовых организаций, занимается продвижением стандарта безопасных электронных транзакций (Secure Electronic Transaction, SET). Одна из компонент системы SET — это возможность для держателей кредитных карточек получить цифровой сертификат в организациях, выдавших кредитную карточку. Если этот стандарт будет воплощен, это должно существенно уменьшить риск отказа от обязательств и риск других махинаций с кредитными карточками в Internet-транзакциях.

К сожалению, несмотря на то что спецификация SET существует уже несколько лет, со стороны банков наблюдаются лишь незначительные усилия в выдаче SET-совместимых сертификатов держателям кредитных карточек. Ни один розничный торговец не захочет отказывать всем покупателям, не использующим SET-совместимое программное обеспечение. Кроме того, сами покупатели не проявляют большого энтузиазма в установке и использовании упомянутого программного обеспечения. Если только все поголовно продавцы не станут требовать от покупателей наличия "цифровых кошельков" (digital wallet), трудно найти какие-то еще причины, из-за которых покупатели должны выстраиваться в очередь в своем банке, чтобы получить какое-то там программное обеспечение и тратить собственное время на его установку.

Достижение баланса между практичностью, производительностью, стоимостью и защитой

По своей природе Web — среда, переполненная рисками. Web создан для того, чтобы позволить большому количеству неизвестных пользователей запрашивать услуги от вашего компьютера. Большинство запросов будут вполне легальными запросами на получение Web-страниц, но если ваш компьютер подключен к Internet, людям ничто не мешает попытаться установить соединения другого типа.

Хотя весьма соблазнительно предположить, что требуется достигнуть наивысшего уровня защиты, подобный уровень не часто встречается. Если вы хотите быть по-настоящему защищенным, держите все ваши компьютеры выключенными, отключенными от всех сетей и закрытыми в сейфе. Чтобы сделать компьютеры чуть более полезными и доступными для клиентов, требуются некоторые послабления в защите.

Между защитой, практичностью, стоимостью и производительностью должен быть найден компромисс. Если сделать службу более защищенной, это может снизить ее практичность, например, если ограничить клиенту количество разрешенных действий или потребовать, чтобы клиент идентифицировал себя. Повышение защищенности также может повлиять на производительность компьютеров. Запуск программного обеспечения, например, системы шифрования, системы обнаружения проникновений, антивирусных программ, системы расширенной регистрации, для защиты системы требует дополнительных ресурсов. Потребуется значительно большая вычислительная мощность, чтобы обеспечить шифрованный, а не обычный сеанс связи с Web-сайтом. Подобного рода потери производительности можно компенсировать, потратив больше денег на более мощные компьютеры или на оборудование, специально созданное для шифрования данных.

Производительность, практичность и защищенность можно рассматривать как конкурирующие цели. Потребуется проанализировать необходимые уступки и принять осмысленное решение, дабы добиться компромисса. Компромисс можно будет найти в зависимости от того, какова ценность вашей информации, сколько пользователей вы собираетесь обслуживать и какие препятствия могут возникнуть со стороны легальных посетителей вашего сайта.

Разработка стратегии защиты

Стратегия защиты — это документ, описывающий следующее:

- Общую философию защиты в вашей организации
- Что следует защищать — программное обеспечение, оборудование, данные
- Кто отвечает за защиту
- Стандарты защищенности и метрика, измеряющая степень выполнения этих стандартов

Хорошим примером во время разработки стратегии защиты может послужить написание функциональных требований для программного обеспечения. Стратегия не должна затрагивать специфические реализации или решения; вместо этого в стратегии должны быть раскрыты цели защиты и требования к защите вашего окружения. Стратегия защиты не должна обновляться слишком часто.

В отдельном документе следует зафиксировать принципы того, как требования стратегии защиты должны быть реализованы в конкретном окружении. Эти принципы могут различаться для разных подразделений вашей организации. Данный документ представляет собой подробнейший проект или процедурное описание того, что на самом деле следует проделать, чтобы гарантировать заявленный уровень защиты.

Принципы аутентификации

Аутентификация — это попытка доказать, что кто-то на самом деле является тем, за кого себя выдает. Существует множество методов обеспечения аутентификации, но, как и в случае с общей защитой, чем больше метод защищен, тем сложнее его использовать.

Методы аутентификации включают пароли, цифровые подписи, биометрические показатели, такие как отпечатки пальцев, а также методы с использованием специального оборудования, например, интеллектуальные карточки (смарт-карты). В Web повсеместно используются только два метода — пароли и цифровые сертификаты.

Биометрические методы и методы, предполагающие применение специального оборудования, требуют наличия специализированных устройств ввода и привязывают санкционированных пользователей к компьютерам, на которых подобные устройства установлены. С этим можно согласиться или даже требовать для доступа ко внутренним системам организации, но такой подход сводит к нулю все преимущества систем, доступных в Web.

Пароли легко реализовать, просто использовать, и они не требуют наличия специального оборудования. Пароли обеспечивают некоторый уровень аутентификации, но не подходят в качестве единственного метода аутентификации для систем с сильной системой защиты.

Концепция паролей достаточно простая. Ваш пароль знают только вы и система. Если посетитель утверждает, что он — это вы и знает ваш пароль, то система ве-

рит в то, что посетитель — это вы. До тех пор пока никто не знает и не может угадать ваш пароль, система остается защищенной. Сами по себе пароли обладают несколькими потенциальными недостатками и не могут использоваться для надежной аутентификации.

Множество паролей несложно угадать. Если выбор пароля возложить на пользователя, то 50% пользователей выберет пароль, который совершенно легко угадать. Известно, что такими паролями являются, например, слова из словаря и имена пользователей в системе. За счет потерь в практичности можно заставить пользователей применять в паролях цифры и знаки пунктуации. Однако это приведет к тому, что у пользователей появятся сложности с запоминанием паролей. Конечно, может помочь обучение пользователей правилам выбора хороших паролей, но даже после обучения около 25% все равно выберут легко угадываемый пароль. Имеет смысл создать стратегию выбора паролей, которая запретит пользователям отдавать предпочтение легко угадываемым паролям. Для этого стратегия должна проверять пароль в словаре или требовать использовать в пароле несколько цифр, знаков пунктуации или комбинации прописных и строчных букв. Опасность подобного подхода в том, что строгие правила для паролей приведут к тому, что многие пользователи попросту не сумеют запомнить собственные пароли.

Трудно запоминающиеся пароли увеличивают вероятность того, что пользователь может записать, например, фразу "Пользователь: Fred Пароль: rover", на самоклеющемся листке и прилепить его на монитор.

Пользователей нужно учить не записывать пароли и не делать других глупых вещей, например, не сообщать пароль человеку, который позвонил по телефону и утверждает, что работает в системе.

Пароли могут перехватываться электронным путем. Используя программы перехвата клавиатурного ввода на терминалах или анализаторы сетевых протоколов (sniffers), взломщики могут перехватывать интересные пары имя пользователя-пароль. Шифрование сетевого потока данных позволяет снизить риск перехвата паролей.

Пароли, при всех своих недостатках, являются простым и относительно эффективным способом аутентификации пользователей. Пароли обеспечивают уровень защиты, который, возможно, не подойдет для министерства национальной безопасности, однако идеально подойдет для проверки состояния доставки товаров, заказанных клиентом.

Использование аутентификации

Механизмы аутентификации встроены в наиболее популярные Web-браузеры и серверы. Web-серверы могут потребовать имя пользователя и пароль у тех, кто желает выгрузить файлы из определенных каталогов на сервере.

Когда браузер получает запрос на ввод имени пользователя и пароля, он отображает диалоговое окно, подобное показанному на рис. 13.2.

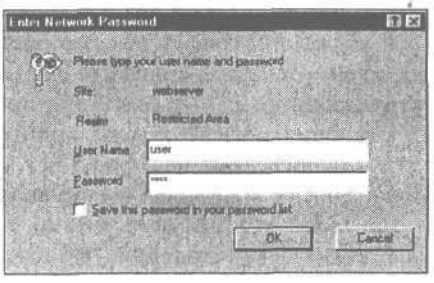


РИСУНОК 13.2. *Web-браузер требует от пользователей аутентификации при попытке посещения защищенного каталога на Web-сервере.*

Web-серверы Apache и Microsoft IIS позволяют подобным способом легко защитить весь сайт или его часть. В случае применения PHP и MySQL существует множество способов достичь того же эффекта. MySQL обеспечивает более быструю реакцию, нежели встроенная аутентификация. С помощью PHP можно реализовать более гибкую аутентификацию или представить запрос в более привлекательной форме.

Некоторые примеры аутентификации можно найти в главе 14.

Основы шифрования

Алгоритм шифрования — это математический процесс преобразования информации в строку данных, которые выглядят как случайные.

Данные, с которых вы начинаете, часто называют *простым (plain) текстом,* хотя для процесса шифрования не имеет значения, что представляет собой информация - реальный текст или данные другого вида. После шифрования получается *зашифрованный текст,* но, как правило, это мало напоминает текст. На рис. 13.3 при помощи простой блок-схемы показан процесс шифрования. Простой текст один раз передается в механизм шифрования, который мог бы быть даже механическим устройством наподобие машины Enigma во времена второй мировой. Сейчас почти все шифровальные машины — это компьютеры. Шифровальный механизм создает зашифрованный текст.



РИСУНОК 13.3. *Процесс шифрования получает на входе простой текст и преобразует его в зашифрованный текст, который выглядит как случайный.*

Чтобы создать защищенный каталог, для доступа в который на рис. 13.2 показано диалоговое окно аутентификации, мы воспользовались самым простым методом аутентификации, предоставленным сервером Apache. (В следующей главе применение этого метода рассматривается более подробно.) Этот метод шифрует пароли перед их сохранением. Мы создали пользователя с паролем **password**. Этот пароль был зашифрован и сохранен в виде строки **aWDuA3X3hmc2**. Несложно заметить, что простой и зашифрованный текст не содержат явно похожих частей.

Показанный метод шифрования не является обратимым. Многие пароли сохраняются с помощью необратимых алгоритмов шифрования. Для проверки корректности вводимого пароля расшифровывать сохраненный пароль не потребуется. Вместо этого вводимый пароль шифруется и результат сравнивается с сохраненной версией.

Многие, но не все процессы шифрования могут быть обратимыми. Обратимый процесс называют *дешифрацией.* На рис. 13.4 показан двунаправленный процесс шифрования.

Криптографии уже насчитывается 4000 лет, но своего совершеннолетия эта наука достигла в период второй мировой войны. С тех пор развитие криптографии повторяет развитие компьютерных сетей — сначала криптография использовалась только военными и финансовыми организациями, с 70-х годов криптография стала шире использоваться в коммерческих компаниях, а в 90-х годах криптография стала использоваться практически повсеместно. За последние несколько лет криптография прошла путь от концепции, которую люди видели в фильмах о второй мировой и в шпионских триллерах, до технологии, о которой каждый день можно читать в газетах и которая задействуется каждый раз, когда осуществляется приобретение чего-то через Web.

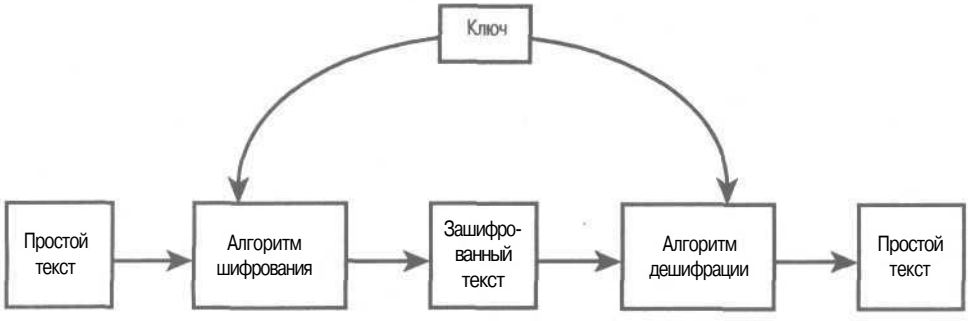


РИСУНОК 13.4. *Процесс шифрования получает на входе простой текст и преобразует его в зашифрованный текст, который выглядит как случайный. Процесс дешифрации получает на входе зашифрованный текст и преобразует его обратно в простой текст.*

Существует великое множество разных алгоритмов шифрования. Некоторые, например, DES, используют секретный, или закрытый ключ. Другие, например, RSA, используют открытый и отдельный закрытый ключи.

Шифрование с закрытым ключом

Шифрование с закрытым ключом (private key encryption) основано на том, что доступ к ключу имеет только авторизованный персонал. Этот ключ должен держаться в секрете. Если ключ попадет не в те руки, то похититель ключа сможет получить несанкционированный доступ к зашифрованной информации. Как показано на рис. 13.4, отправитель (который шифрует сообщение) и получатель (который дешифрует сообщение) имеют один и тот же ключ.

Наиболее широко используется алгоритм с закрытым ключом DES (Data Encryption Standard). Этот алгоритм, разработанный компанией IBM в 70-х годах, принят как американский стандарт для коммерческих и правительственных несекретных коммуникаций. Современные скорости вычислений на порядок превышают скорости вычислений в 70-х годах, поэтому алгоритм DES считается устаревшим как минимум с 1998 г.

Другие известные алгоритмы с закрытым ключом — это RC2, RC4, RC5, тройной DES (triple DES) и IDEA. Тройной DES-алгоритм достаточно защищен[2]. Этот алгоритм использует тот же метод шифрования, что и DES, но применяет его трижды, используя при этом до трех разных ключей. Простой текст шифруется с использованием первого ключа, дешифруется при помощи второго ключа, а затем шифруется с применением третьего ключа.

Явный недостаток алгоритмов с закрытым ключом состоит в том, что для отправки кому-либо защищенного сообщения необходимо располагать безопасным способом передачи этой персоне закрытого ключа. А если у вас есть безопасный метод передачи ключа, то почему не воспользоваться этим же методом для передачи сообщений?

Прорыв, к счастью, произошел в 1976 г., когда Дифи (Diffie) и Хелман (Hellman) опубликовали первый алгоритм с открытым ключом.

---

[2] Парадокс, но тройной DES всего лишь в два раз защищеннее обычного алгоритма DES. Если вам нужен алгоритм, который в три раза защищеннее обычного алгоритма DES, придется написать программу, которая применяет стандартный алгоритм DES четыре раза.

Шифрование с открытым ключом

Шифрование с открытым ключом (public key encryption) базируется на двух различных ключах — открытом и закрытом. Как показано на рис. 13.5, открытый ключ используется для шифрования сообщения, а закрытый — для дешифрации сообщения.

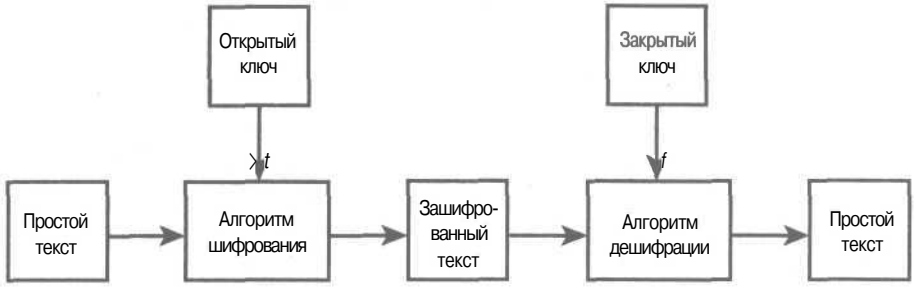


РИСУНОК 13.5. *Шифрование с открытым ключом использует различные ключи для шифрования и дешифрации.*

Преимущество этого подхода состоит в том, что, как следует из названия, открытый ключ можно свободно распространять. Любой человек, которому вы дали открытый ключ, может отправлять вам защищенное сообщение. Поскольку закрытым ключом обладаете только вы, только вы и сможете дешифровать сообщение.

Наиболее известный алгоритм с открытым ключом — это алгоритм RSA, который был разработан Ривестом (Rivest), Шамиром (Shamir) и Адельманом (Adelman) в Мичиганском технологическом институте (MIT) и опубликован в 1978 г. Ранее алгоритм RSA был частной собственностью, но срок действия патента закончился в сентябре 2000 г.

Огромным преимуществом алгоритмов с открытым ключом является возможность передачи открытого ключа по незащищенному каналу, не беспокоясь, что он будет прочитан третьей стороной. Несмотря на это, все еще повсеместно используются системы с закрытым ключом. Часто можно встретить гибридные системы. В таких системах алгоритм с открытым ключом используется для передачи закрытого ключа, который используется для обмена данными до конца сеанса связи. Эта дополнительная сложность компенсируется, так как алгоритмы с закрытым ключом в тысячи раз быстрее алгоритмов с открытым ключом.

Цифровые подписи

Цифровые подписи (digital signature) относятся к алгоритмам с открытым ключом, но с измененными ролями открытого и закрытого ключей. Отправитель может зашифровать и подписать сообщение своим закрытым ключом. Когда сообщение получено, получатель может дешифровать его, используя открытый ключ отправителя. Ввиду того что отправитель — это единственное лицо, обладающее доступом к закрытому ключу, то получатель четко знает, от кого получено сообщение, а также имеет уверенность, что сообщение не было изменено.

Цифровые подписи могут оказаться очень полезными. Они гарантируют получателю, что сообщение не было подделано, а также не позволяют отправителю отказаться от обязательств, отрицая факт отправки сообщения.

Важно заметить, что хотя сообщения шифруются, их может прочитать любой обладатель открытого ключа. Несмотря на то что используются те же методы и ключи, назначение шифрования в данном случае не запретить чтение, а предотвратить подделку и отказ от обязательств.

Поскольку алгоритмы с открытым ключом достаточно медленно работают на больших сообщениях, для повышения производительности используется алгоритм другого типа, называемый *хэш-функцией.*

Хэш-функция вычисляет дайджест сообщения, или хеш-значение, для каждого данного сообщения. Совершенно не важно, какое значение генерирует алгоритм. Важно, что результат этой функции является детерминированным, т.е. результат будет одним и тем же каждый раз, когда на вход передаются одинаковые данные. Кроме того, важно, что результат имеет небольшой размер и алгоритм быстро работает.

Наиболее известные хэш-функции — это MD5 и SHA.

Хэш-функция генерирует дайджест, соответствующий определенному сообщению. Располагая сообщением и его дайджестом, можно убедиться, не подделывалось ли сообщение, но только в том случае, если дайджест не был подделан вместе с ним.

И, наконец, обычный способ создания цифровой подписи — это создание при помощи быстрой хэш-функции дайджеста для всего сообщения, а затем шифрование только короткого дайджеста с использованием медленного алгоритма с открытым ключом. Теперь подпись можно отправить вместе с сообщением через нормальный, незащищенный канал связи.

После получения можно проверить подлинность подписанного сообщения. Подпись дешифруется с помощью открытого ключа отправителя. Для сообщения генерируется хэш-значение с помощью той же хэш-функции, которую использовал отправитель. Если дешифрованное хэш-значение совпадает с вычисленным значением, это значит, что сообщение действительно прислано отправителем и при пересылке не изменялось.

Цифровые сертификаты

Хорошо бы иметь возможность проверять, что сообщение не было изменено, а последовательность сообщений вся полностью пришла с определенного компьютера или от определенного пользователя. Для коммерческого взаимодействия было бы даже лучше иметь возможность привязать пользователя или сервер к какому-то реальному и правовому понятию, такому как физическое или юридическое лицо.

Цифровой сертификат объединяет в цифровой подписанной форме открытый ключ и детальную информацию о человеке или организации. Получив сертификат, вы получаете открытый ключ другой стороны для отправки зашифрованных сообщений по мере необходимости. Кроме того, вы получаете детальную информацию о другой стороне и обладаете уверенностью, что никто не изменил эти данные.

Проблема состоит в том, что информация из сертификата вызывает ровно столько доверия, сколько человек, подписавший этот сертификат. Любой человек может создать и подписать сертификат, в котором будет утверждаться все, что угодно. Для коммерческих транзакций полезно иметь третью доверенную сторону, которая будет проверять идентичность участников и информации, записанной в их сертификатах.

Такие третьи стороны называют *органами сертификации (Certifying Authority, CA).* Органы сертификации выдают цифровые сертификаты людям и компаниям, которые должны для этого пройти проверку на подлинность. Из всех органов сертификации наиболее известными являются Verisign (**http://www.verisign.com/**) и Thawte (**http://www.thawte.com/**). VeriSign и Thawte являются собственностью од-

ной компании, поэтому существенной разницы между их сертификатами нет. Сертификаты некоторых из менее известных органов сертификации, таких как Equifax Secure (**www.equifaxsecure.com**) стоят значительно дешевле.

Органы сертификации подписывают сертификаты, чтобы подтвердить, что они видели доказательства идентичности человека или компании. Важно заметить, что сертификат не является справкой или официальным подтверждением кредитоспособности. Сертификат также не гарантирует, что вы имеете дело с кем-то, обладающим хорошей репутацией. Сертификат гарантирует то, что если вас ограбят, то у вас будет шанс найти реальный физический адрес того, кого вы вызовете в суд.

Сертификат позволяет создать сеть доверия. Предположим, вы решили доверять органу сертификации, тогда вы можете решиться доверять людям и компаниям, которым доверяет выбранный орган сертификации. Далее можно решиться доверять всем органам сертификации, которым доверяет ваш орган сертификации, а также — людям, которым доверяют все эти органы сертификации.

На рис. 13.6 показан путь сертификата, который отображает Internet Explorer для каждого сертификата. Вы видите, что **www.equifaxsecure.com** имеет сертификат, выданный органом сертификации Equifax Secure E-Business. Этот орган, в свою очередь, имеет сертификат, выданный органом сертификации Thawte Server.

Цифровые сертификаты чаще всего используются с целью поддержки атмосферы респектабельности на вашем сайте электронной коммерции. Обладая сертификатом, выданным хорошо известным органом сертификации, Web-браузер может установить SSL-соединение с вашим сайтом и при этом не выводить никаких предупреждающих диалоговых окон. Web-серверы, которые поддерживают SSL-соединения, часто называют безопасными Web-серверами.

Безопасные Web-серверы

Для безопасной связи с Web-браузерами через протокол Secure Socket Layer можно использовать сервер Apache, Microsoft IIS или любой другой бесплатный или коммерческий Web-сервер. Использование сервера Apache позволяет вам отдать предпочтение операционной системе Unix, и такая конфигурация, вероятнее всего, окажется более надежной, хотя и более трудной в установке, чем IIS. Конечно, сервер Apache можно выполнять на платформе Windows NT.



РИСУНОК 13.6.
Путь сертификата www.equifaxsecure.com показывает сеть доверия этого сертификата, что позволяет доверять этому сайту.

Чтобы использовать SSL на сервере IIS, необходимо установить IIS, сгенерировать пару ключей и установить свой сертификат. Для активизации протокола SSL на сервере Apache потребуется установить три различных пакета: Apache, Mod_SSL и OpenSSL.

Ухватить свой кусок пирога можно, купив пакет Stronghold. Пакет Stronghold — это коммерческий продукт стоимостью примерно 1000 долларов США. Этот пакет функционирует на сервере Apache и поставляется как самоустанавливающийся бинарный файл, уже настроенный на использование SSL. Таким образом, вы получаете надежность Unix в композиции с простотой установки и технической поддержкой от производителя.

В приложении А приводятся инструкции по установке двух наиболее популярных Web-серверов — Apache и IIS. Можно сразу же начать использовать SSL, сгенерировав собственный сертификат, однако посетители вашего сайта будут предупреждаться, что вы собственноручно подписали свой сертификат. Для эффективного использования SSL потребуется получить сертификат у одного из органов сертификации.

Детали процесса получения сертификата зависят от конкретного органа сертификации, но в общем случае вам нужно будет доказать, что вы являетесь легально признанной организацией с физическим адресом, и эта организация владеет соответствующим именем домена.

Далее необходимо создать запрос на подпись сертификата (Certificate Signing Request, CSR). Этот процесс варьируется от сервера к серверу. Соответствующие инструкции размещаются на Web-сайтах органов сертификации. Пакеты Stronghold и IIS организуют этот процесс на базе диалоговых окон, а сервер Apache требует прямого ввода команд. Тем не менее, процесс по сути дела одинаков для всех серверов. Конечный результат этого процесса — получение зашифрованного запроса на подпись сертификата. CSR должен выглядеть приблизительно так:

```
----- BEGIN NEW CERTIFICATE REQUEST -----
MIIBuwIBAAKBgQCLn1XX8faMHhtzStp9wY6BVTPuEU9bpMmhrb6vgaNZy4dTe6VS
84p7wGepq5CQjfOL4Hjda+g12xzto8uxBkCDO98Xg9q86CY45HZk+q6GyGOLZSOD
8cQHwh1oUP65s5Tz018OFBzpI3bHxfO6aYelWYziDiFKp1BrUdua+pK4SQIVAPLH
SV9FSz8Z7IHOg1Zr5H82oQOlAoGAWSPWyfVXPAF8h2GDb+cf97k44VkHZ+Rxpe8G
ghlfBn9L3ESWUZNOJMfDLlny7dStYU98VTVNekidYuaBsvyEkFrny7NCUmiuaSnX
4UjtFDkNhX9j5YbCRGLmsc865AT54KRu31O2/dKHLo6NgFPirijHy99HJ4LRY9Z9
HkXVzswCgYBwBFH2QfK88C6JKW3ah+6cHQ4Deoiltxi627WN5HcQLwkPGn+WtYSZ
jG5tw4tqqogmJ+IP2F/5G6FI2DQP7QDvKNeAU8jXcuijuWo27S2sbhQtXgZRTZvO
jGn89BC0mIHgHQMkI7vz35mx1Skk3VNq3ehwhGCvJlvoeiv2J8X2IQIVAOTRp7zp
En7QlXnXw1s7xXbbuKP0
----- END NEW CERTIFICATE REQUEST -----
```

Вооружившись запросом CSR, соответствующей суммой, документацией, доказывающей, что вы существуете, а также проверив, что имя используемого домена совпадает с именем домена в деловых документах, можно зарегистрироваться в органе сертификации на получение сертификата.

Когда орган сертификации выдаст сертификат, его следует сохранить в своей системе и указать Web-серверу, где искать сертификат. Окончательный сертификат — это текстовый файл, который выглядит подобно показанному выше запросу CSR.

Аудит и регистрация

Используемая операционная система позволяет регистрировать события любого типа. С точки зрения безопасности могут интересовать следующие события: ошибки сети, доступ к определенным файлам, таким как файлы конфигурации или реестр NT, а также обращения к программам, таким как su (эта программа используется в Unix-системах, чтобы стать другим пользователем, как правило, root).

Файлы регистрации (журналы) помогают обнаружить ошибочные или злонамеренные действия по мере их возникновения. Если после обнаружения проблем проверить журналы, можно будет понять, как появилась проблема или каким образом предпринималась попытка проникновения. С файлами регистрации связаны две проблемы – размер и достоверность.

Если установить наиболее параноидальный критерий обнаружения и регистрации проблем, в результате получатся огромные журналы, которые проверять очень сложно. Чтобы решить проблему больших журналов необходимо воспользоваться существующими утилитами или, руководствуясь стратегией защиты, создать собственные сценарии поиска "интересных" событий в журнальных файлах. Процесс аудита можно выполнять в реальном масштабе времени или на периодической основе.

Журнальные файлы также могут подвергаться атакам. Если взломщик обладает привилегиями администратора в вашей системе, он может изменить файлы регистрации, дабы скрыть следы своей деятельности. Unix позволяет регистрировать события в журнале на другом компьютере. Это значит, что взломщику придется проникнуть, по меньшей мере, на два компьютера, чтобы замести следы. Подобное можно сделать и в Windows NT, однако не так легко.

Ваш системный администратор может производить регулярные проверки, но вам может понадобиться и внешний аудит, чтобы проверить поведение самого администратора.

Брандмауэры

Брандмауэры (firewall) в сети существуют для того, чтобы отделить локальную сеть от внешнего мира. Как и противопожарные стены, препятствующие распространению огня, сетевые брандмауэры не позволяют хаосу проникать внутрь сети.

Брандмауэр предназначен для защиты компьютеров в сети от атак извне. Брандмауэр фильтрует или запрещает передачу данных, если данные не удовлетворяют определенным правилам. Брандмауэр ограничивает действия компьютеров и людей, которые находятся вне сети.

Иногда брандмауэр используется для ограничения действий пользователей внутри сети. Брандмауэр может ограничить протоколы, доступные пользователям, запретить соединения с определенным узлом сети и заставить использовать прокси-сервер для снижения стоимости полосы пропускания.

Брандмауэром может быть либо специальное устройство, такое как маршрутизатор с правилами фильтрации, либо программа, выполняющаяся на компьютере. В любом случае, брандмауэру нужен доступ к двум сетям и набор правил. Брандмауэр контролирует весь поток данных, который проходит из одной сети в другую. Если поток данных удовлетворяет правилам, он передается в другую сеть, в противном случае поток останавливается и отвергается.

Пакеты можно фильтровать по типам, адресам отправителя, адресам получателя, а также информации о портах. Некоторые пакеты могут быть просто отброшены, в то время как другие могут регистрироваться в журналах и включать тревогу.

Резервное копирование данных

При любом плане восстановления после сбоя нельзя недооценивать важность резервного копирования. Оборудование и здания можно застраховать или поменять, можно разместить сайт где-то еще, но если потерять созданное программное обеспечение, то никакая страховая компания не сможет возместить убытки.

Регулярно создавайте резервные копии всех компонентов Web-сайта — статических страниц, сценариев и баз данных. Частота создания резервных копий зависит от того, насколь динамичен сайт. Если сайт полностью статический, можно обойтись резервным копированием при внесении изменений. Однако сайты, которые рассматриваются в этой книге, вероятно, будут изменяться часто, особенно, если планируется принимать заказы.

Большинство сайтов приличных размеров должны устанавливаться на сервере с дисковым массивом RAID (Redundant Array of Inexpensive Disks — избыточный массив недорогих дисков), который поддерживает зеркальное отображение. Такой подход учитывает и возможные сбои жестких дисков. Тем не менее, следует рассматривать ситуации, когда что-то может случиться со всем RAID-массивом, компьютером или вообще зданием.

Отдельные резервные копии следует создавать с частотой, которая соответствует объему производимых обновлений. Эти резервные копии необходимо хранить на отдельных носителях и, предпочтительно, в отдельном и безопасном месте на случай пожара, кражи или стихийных бедствий.

Существует масса приложений для резервного копирования и восстановления. Мы сконцентрируем внимание на том, как создавать резервные копии сайта, построенного на PHP и MySQL.

Резервное копирование общих файлов

Воспользовавшись приложениями для резервного копирования, можно достаточно легко создать резервные копии файлов HTML, PHP, изображений и других, не относящихся к базам данных.

Из бесплатных программ наиболее широко используется утилита резервного копирования AMANDA (Advanced Maryland Automated Network Disk Archiver). Эта утилита создана в университете Мериленда. Она поставляется в составе многих дистрибутивов Unix и может быть использована для резервного копирования Windows-компьютеров через пакет SAMBA. Дополнительную информацию об утилите AMABDA можно отыскать на сайте **http://www.amanda.org**

Резервное копирование и восстановление баз данных MySQL

Резервное копирование действующей базы данных значительно сложнее. Потребуется избежать копирования какой-либо таблицы, в которой в этот момент производятся изменения.

Инструкции по резервному копированию и восстановлению баз данных MySQL приводятся в главе 11.

Физическая безопасность

Рассмотренные до этого момента угрозы безопасности относились к нематериальным вещам, таким как программное обеспечение, но не следует забывать и о физической безопасности системы. Вам необходимо кондиционирование воздуха и защита от пожаров, людей (как просто неуклюжих, так и злоумышленников), перебоев электроснабжения и сбоев сети.

Система должна быть надежно закрыта. В зависимости от размеров системы, это может быть комната, клетка или шкаф. Сотрудники, которым не нужен физический доступ к системе, и не должны его иметь. Неавторизованные сотрудники могут намеренно или случайно выдернуть какой-то шнур или попытаться обойти систему безопасности с помощью загрузочной дискеты.

Противопожарные разбрызгиватели воды могут нанести такой же ущерб, как и сам пожар. В прошлом во избежание ущерба использовались системы тушения пожара на основе газа халон (halon). Теперь производство халона запрещено Монреальским протоколом о субстанциях, которые разрушают озоновый слой, поэтому должны использоваться новые, менее губительные системы на основе аргона или двуокиси углерода. Более подробную информацию об этом можно получить на сайте **http://epa.gov/ozone/title6/snap**.

Редкие и короткие перебои в электроснабжении происходят везде. В районах с неустойчивой погодой и наземными линиями электроснабжения перебои случаются регулярно. Если для вас важна постоянная работа системы, потребуется вложить деньги в бесперебойный источник питания (ИБП, UPS, Uninterruptible Power Supply). ИБП, который может поддерживать питание одного компьютера в течении 10 минут, стоит менее 300 долларов США. Более длительные перебои или больше оборудования могут увеличить ваши расходы. Длительные перебои в электроснабжении потребуют покупки генератора, который обеспечит работу как компьютеров, так и кондиционеров.

Как и перебои в электроснабжении, нет возможности контролировать как минутные, так и часовые перебои в Internet, хотя они случаются достаточно редко. Если сеть является критичной, имеет смысл использовать соединения с более чем одним Internet-провайдером. Конечно, два соединения с Internet будут стоить дороже одного, но в случае сбоя у вас все же останется ограниченное по скорости соединение, а это лучше чем ничего.

Проблемы подобного рода являются одной из причин, по которой возникнет желание разместить свои компьютеры в специально предназначенных местах. Хотя организация среднего размера не может позволить себе самостоятельно приобрести источники бесперебойного питания, которые поддержат систему дольше, чем несколько минут, иметь избыточные соединения с Internet и противопожарные системы, все это доступно в специально оборудованных офисных зданиях, предоставляющих место тысячам подобных организаций.

Что дальше

В главе 14 мы детально остановимся на аутентификации — обеспечении пользователей возможностью подтвердить свою личность. Будут рассмотрены различные методы аутентификации посетителей, включая и те, которые используют PHP и MySQL.

Глава 14

Аутентификация с помощью PHP и MySQL

В этой главе будет показано, как использовать различные возможности PHP и MySQL для аутентификации пользователей.

В главе будут рассматриваться следующие вопросы:

- Идентификация посетителей
- Реализация контроля доступа
- Базовая аутентификация
- Использование базовой аутентификации в PHP
- Использование базовой аутентификации при помощи файлов .htaccess сервера Apache
- Использование базовой аутентификации в IIS
- Использование аутентификации через модуль mod_auth_mysql
- Создание собственного метода аутентификации

Идентификация пользователей

Web — это достаточно анонимная среда, но часто полезно знать, кто посетил ваш сайт. К счастью для конфиденциальности посетителей, без их помощи можно получить только очень незначительную информацию.

Приложив немного усилий, серверы могут получить достаточно много информации о компьютерах и сетях, которые соединяются с серверами. Обычно Web-браузер идентифицирует себя, указывая название браузера, версию браузера и операционную систему. Имеется возможность определить разрешение и глубину цвета мониторов посетителей, а также размеры окна браузера.

Каждый подключенный к Internet компьютер имеет уникальный IP-адрес. Вы не многое узнаете из IP-адреса посетителя. Можно узнать, кто владеет этим адресом, а иногда с некоторой долей вероятности можно предположить географическое положение посетителя. Одни адреса

дают больше информации, другие — меньше. В основном, люди с постоянным подключением к Internet имеют постоянные IP-адреса. А клиенты, которые дозваниваются к Internet-провайдерам, в большинстве случаев получают во временное пользование один из IP-адресов провайдера. Когда в следующий раз вы увидите этот адрес, он уже может использоваться другим компьютером, а когда вы увидите предыдущего пользователя, у него, возможно, будет другой IP-адрес.

К счастью для пользователей Web, информация, которую выдает браузер, не позволяет идентифицировать пользователя. Если хотите знать имя посетителя и другие детали, следует спросить его об этом.

Многие Web-сайты заставляют пользователей предоставлять о себе информацию. Например, в электронном виде газета *New York Times* **(http://www.nytimes.com)** выдается бесплатно только тем, кто предоставляет о себе детальную информацию, такую как имя, пол и общий семейный доход. Сайт новостей и дискуссий для компьютерщиков Slashdot **(http://www.slashdot.org)** позволяет зарегистрированным пользователям настраивать для себя интерфейс сайта и принимать участие в дискуссиях, используя псевдонимы (nickname). Большинство сайтов электронной коммерции записывают детальную информацию о клиенте при оформлении первого заказа. Это означает, что покупателю не потребуется вводить информацию о себе при каждом заказе.

Запросив и получив в ответ информацию о посетителе, необходимо каким-то образом ассоциировать ее с посетителем при его следующих входах на сайт. Если предположить, что с определенного компьютера, используя определенное имя пользователя на этом компьютере, на сайт заходит только один пользователь, а каждый пользователь работает только на одном компьютере, то для идентификации пользователя можно создать cookie-набор на компьютере пользователя. Подобное предположение неверно для большинства пользователей. Часто многие люди поочередно пользуются одним и тем же компьютером, либо же кто-то использует несколько компьютеров. По крайней мере, иногда приходится повторно спрашивать пользователя о том, кто он. В дополнение к этому, придется попросить посетителя предоставить какие-то доказательства того, что он тот, за кого себя выдает.

Как упоминалось в главе 13, просьба к пользователю доказать свою личность называется *аутентификацией*. Обычный метод аутентификации в Web — это требование к посетителям предоставить уникальное имя пользователя и пароль. Аутентификация обычно используется для разрешения или запрещения доступа к определенным страницам или ресурсам. Аутентификация может быть необязательной либо использоваться для других целей, например, для персонализации.

Реализация контроля доступа

Простой контроль доступа реализовать несложно. Код, показанный на листинге 14.1, выводит одну из трех возможных страниц. Если этот файл загружен без параметров, будет отображаться HTML-форма с приглашением ввести имя пользователя и пароль (см. рис. 14.1).

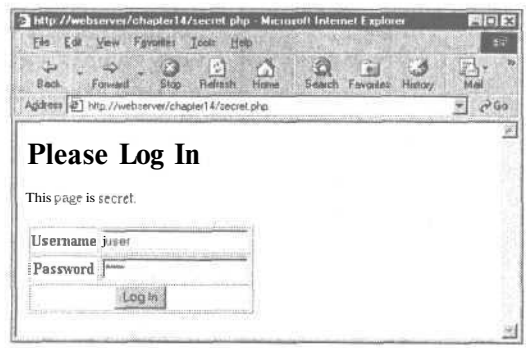


РИСУНОК 14.1. *Наша HTML-форма запрашивает имя и пароль пользователя для полунения доступа.*



РИСУНОК 14.2. *Когда пользователи вводят неправильные данные, им следует отобразить сообщение об ошибке. На реальном сайте это сообщение должно быть более дружественным.*



РИСУНОК 14.3. *Когда имя и пароль указаны правильно, сценарий выводит секретное содержимое.*

Если при загрузке параметры присутствуют, но они неправильные, отображается сообщение об ошибке (см. рис. 14.2).

Если при загрузке параметры присутствуют и они правильные, посетителю отображается секретное содержимое. Для нашего случая данные показаны на рис. 14.3.

Код для создания функциональности, продемонстрированной на рис. 14.1, 14.2 и 14.3, приведен в листинге 14.1

Листинг 14.1 secret.php — PHP- и HTML-код для реализации простого механизма аутентификации

```
<?
   if(!isset($name)&&!isset($password))
   {
      // Посетитель должен ввести имя и пароль
?>
      <hl>Flease Log In</hl>
      This page is secret.
      <form method = post action = "secret.php">
      <table border = 1>
      <tr>
         <th> Username </th>
         <td> <input type = text name = name> </td>
      </tr>
      <tr>
         <th> Password </th>
         <td> <input type = password name = password> </td>
      </tr>
      <tr>
         <td colspan =2 align = center>
            <input type = submit value = "Log In">
         </td>
      </tr>
      </table>
      </form>
<?
   }
   else if($name=="user"&&$password=="pass")
   {
      // Комбинация имени и пароля посетителя правильная
      echo "<h1>Here it is!</h1>";
      echo "I bet you are glad you can see this secret page.";
   }
```

```
    else
    {
       // Комбинация имени и пароля посетителя неправильная
      echo "<h1>Go Away!</h1>";
      echo "You are not authorized to view this resource.";
    }
?>
```

Код, показанный в листинге 14.1, реализует простой механизм, позволяющий санкционированным посетителям видеть защищенную страницу, однако код содержит несколько значительных проблем.

Этот сценарий:

- поддерживает только одно жестко закодированное имя пользователя и пароль
- хранит пароль в виде простого текста
- защищает только одну страницу
- передает пароль в виде простого текста

Упомянутые проблемы можно разрешить с различной степенью усилий и успеха.

Хранение паролей

Для хранения паролей существует много более подходящих, нежели код сценария, мест. Внутри сценария очень трудно изменять данные. Можно написать сценарий, который будет изменять себя, но это плохая идея. Это будет означать существование выполняющегося на сервере сценария, доступного для записи и изменений со стороны других пользователей. Хранение паролей в отдельном файле на сервере позволит без труда написать программу для добавления и удаления пользователей, а также для изменения паролей.

Внутри сценария или другого файла данных существует ограничение на количество пользователей, которых можно обслуживать, серьезно не навредив общей производительности сценария. Если планируется сохранять большое количество элементов в файле или производить поиск в рамках большого числа элементов, то, как обсуждалось ранее, следует рассмотреть возможность использования базы данных вместо двумерного файла. Практический метод выбора между файлом и базой данных гласит: если вы собираетесь хранить и производить поиск в более чем 100 элементах, следует отдать предпочтение базе данных.

Использование базы данных для хранения имен и паролей посетителей не сильно усложнит сценарий, но позволит быстро проводить аутентификацию множества пользователей. Это также упростит создание сценария для добавления и удаления пользователей, а также даст возможность пользователям изменять свои пароли.

Сценарий для аутентификации посетителей страницы с использованием базы данных приведен в листинге 14.2.

Листинг 14.2. secretdb.php — мы воспользовались MySQL для улучшения нашего простого механизма аутентификации.

```
<?
   if(!isset($name)&&!isset($password))
   {
      // Посетитель должен ввести имя и пароль
?>
      <h1>Please Log In</h1>
      This page is secret.
```

```
    <form method = post action = "secretdb.php">
  <table border = 1>
  <tr>
    <th> Username </th>
    <td> <input type = text name = name> </td>
  </tr>
  <tr>
    <th> Password </th>
    <td> <input type = password name = password> </td>
  </tr>
  <tr>
    <td colspan =2 align = center>
      <input type = submit value = "Log In">
    </td>
  </tr>
  </table>
  </form>
<?
  }
  else
  {
    // Подключиться к MySQL
    $mysql = mysql_connect ( 'localhost', 'webauth', 'webauth' );
    if(!$mysql)
    {
      echo 'Cannot connect to database. ' ;
      exit;
    }
     // Выбрать соответствующую базу данных
    $mysql = mysql_select_db( 'auth' );
    if(!$mysql)
    (
      echo 'Cannot select database.';
      exit;
    }
    // Запрос к базе данных, чтобы проверить,
    // существует ли соответствующая запись
    $query = "select count (*) from auth where
                name = ' $name' and
                pass = '$password'";

    $result = mysql_query ( $query ) ;
    if (!$result)
    {
      echo 'Cannot run query.';
      exit;
    }
    $count = mysql_result ( $result, 0, 0 );

    if ( $count > 0 )
    {
      // Комбинация имени и пароля посетителя правильная
      echo "<h1>Here it is!</h1>";
      echo "I bet you are glad you can see this secret page.";
    }
    else
    {
      // Комбинация имени и пароля посетителя не правильная
      echo "<h1>Go Away!</h1>";
      echo "You are not authorized to view this resource.";
    }
  }
?>
```

Используемую в примере базу данных можно создать, подключившись к MySQL как пользователь root и запустив показанный в листинге 14.3 сценарий.

Листинг 14.3. createauthdb.php — эти запросы создают базу данных auth, таблицу auth и двоих пользователей.

```
create database auth;

use auth;

create table auth (
          name              varchar(10)  not null,
          pass              varchar(30)  not null,
          primary key        (name)
);

insert into auth values
   ('user', 'pass');

insert into auth values
   ( 'testuser', password('test123') );

grant select, insert, update, delete
on auth.*
to webauth@localhost
identified by 'webauth';
```

Шифрование паролей

Независимо от того, где хранятся пароли — в базе данных или в файле — хранение паролей в виде простого текста сопряжено с неоправданным риском. Однонаправленный алгоритм хэширования обеспечит дополнительную защиту при незначительных дополнительных затратах.

PHP-функция **crypt()** представляет собой однонаправленную криптографическую хэш-функцию. Прототип этой функции таков:

```
string crypt (string str[, string salt])
```

Получив на входе строку **str**, эта функция возвращает псевдослучайную строку. Например, если передать в функцию строку **"pass"** и аргумент **salt** равный **"xx"**, то **crypt()** вернет строку **"xxkT1mYjlikoII"**. Эта строка не может быть дешифрована и превращена обратно в **"pass"** даже ее создателем, поэтому на первый взгляд строка может и не показаться столь уж полезной. Что делает функцию **crypt()** полезной, так это то, что результат этой функции детерминирован. При каждом вызове с одними и теми же параметрами **str** и **salt** эта функция будет возвращать один и тот же результат.

Вместо PHP-кода, такого как

```
if( $username == "user" && $password == "pass" )
{
   // Пароль совпадает
}
```

можно воспользоваться таким кодом

```
if( $username = 'user' && crypt($password,'xx') == 'xxkT1mYjlikoII' )
{
   // Пароль совпадает
}
```

Нам не требуется знать, как выглядела строка **"xxkT1mYjlikoII"** перед использованием функции **crypt()**. Необходимо только знать, совпадает ли введенный пароль с тем паролем, для которого применялась функция **crypt()**.

Как уже упоминалось, жесткое кодирование правильных имен и паролей посетителей — плохая идея. Для этого следует организовать отдельный файл или базу данных.

Если для хранения данных аутентификации используется база данных MySQL, можно воспользоваться PHP-функцией **crypt()** или MySQL-функцией **PASSWORD()**. Результат этих функций не совпадает, но они имеют одно предназначение. Обе функции — **crypt()** и **PASSWORD()** — получают строку как аргумент и применяют к полученной строке необращаемый алгоритм хэширования.

Чтобы задействовать функцию **PASSWORDO** в листинге 14.2 запрос SQL следует переписать так:

```
select count(*) from auth where
         name = '$name' and
         pass = password('$password')
```

Этот запрос посчитает количество строк в таблице **auth,** в которых значение поля name совпадает с содержимым переменной **Sname** и значение поля pass совпадает с результатом функции **PASSWORDO,** примененной к значению переменной **$password**. Учитывая, что мы заставляем посетителей выбирать уникальные имена, результатом запроса может быть 0 или 1.

Защита множества страниц

Защита более чем одной страницы с помощью подобных сценариев немного сложнее. Поскольку в HTTP-протоколе нет механизма состояний, то не существует автоматической связи или ассоциации между последовательными запросами от одного и того же посетителя. Это усложняет перенос между страницами введенных пользователем данных, таких как данные аутентификации.

Наиболее простой способ защиты множества страниц — это использование механизмов контроля доступа вашего Web-сервера. Вскоре мы это рассмотрим.

Чтобы самостоятельно создать такую функциональность, потребуется включить части листинга 14.1 в каждую страницу, которую необходимо защитить. При помощи директив **auto_prepend_file** и **auto_append_file** требуемый файл можно автоматически вставить в начало (prepend) или в конец (append) каждого файла в указанных каталогах. Использование упомянутых директив обсуждалось в главе 5.

Если воспользоваться таким подходом, то что произойдет, когда пользователь откроет несколько страниц на сайте? Недопустимо запрашивать пароль отдельно для каждой страницы, которую желает просмотреть пользователь.

Можно включить введенную пользователем информацию в каждую гиперссылку на странице. Так как пользователи могут применять пробелы или другие символы, запрещенные в URL, следует обратиться к функции **urlencode()**, чтобы безопасно упаковать подобные символы.

С этим подходом связаны еще некоторые проблемы. Поскольку данные аутентификации будут присутствовать в отправляемой посетителю Web-странице, защищенные страницы, которые посетил пользователь, могут быть просмотрены любым человеком, работающим за тем же компьютером. Для этого достаточно щелкнуть на кнопке Назад в окне браузера и просмотреть кэшированные копии страниц или заглянуть в историю посещения страниц. Пароль пересылается в браузер и обратно с каждой запрошенной или предоставленной страницей, что происходит чаще, чем это необходимо.

Решить проблему можно с помощью двух механизмов — базовой HTTP-аутентификации и поддержки сеансов. Базовая аутентификация позволяет решить проблему кэширования, но сервер все равно отправляет пароль Web-браузеру в каждом запросе. Управление сеансами позволяет решить обе проблемы. Сначала рассмотрим базовую HTTP-аутентификацию, а управление сеансами освещается в главах 20 и 24.

Базовая аутентификация

К счастью, аутентификация пользователей — это достаточно распространенная задача и существуют возможности аутентификации, встроенные в HTTP-протокол. Сценарии и Web-серверы могут запрашивать аутентификацию у Web-браузера. После этого Web-браузер должен вывести на экран диалоговое окно или что-то подобное и запросить у пользователя необходимую информацию.

Хотя Web-сервер запрашивает новые детали аутентификации в каждом запросе пользователя, Web-браузеру нет необходимости запрашивать эту информацию для каждой страницы. В общем случае браузер хранит детали аутентификации, пока открыто окно браузера, и автоматически отправляет их без вмешательства со стороны пользователя.

Описанная возможность HTTP-протокола называется *базовой аутентификацией*. Базовую аутентификацию можно включить средствами PHP или с помощью Web-сервера. Далее рассматриваются методы, предполагающие использование PHP, Apache и IIS.

Базовая аутентификация передает имя пользователя и пароль в виде простого текста и поэтому не особо безопасна. Протокол HTTP 1.1 обладает более безопасным методом, называемым *дайджест-аутентификацией (digest authentication)*. Этот метод использует алгоритм хэширования (как правило, MD5) для маскировки деталей транзакции. Дайджест-аутентификация поддерживается во многих Web-серверах, но не поддерживается в значительном числе браузеров. Дайджест-аутентификация поддерживается в браузере Microsoft Internet Explorer начиная с версии 5.0. Поддержка дайджест-аутентификации включена в Netscape Navigator версию 6.0.

В дополнение к очень слабой поддержке в наборе доступных браузеров, дайджест-аутентификация к тому же и не очень безопасна. И базовая, и дайджест-аутентификация предоставляют низкий уровень защищенности. Ни один из этих методов не дает пользователю гарантий, что он работает именно с тем компьютером, доступ к которому он планировал получить. Оба метода позволяют взломщику повторить тот же запрос серверу. Поскольку базовая аутентификация передает пароль пользователя в открытом виде, любой взломщик, способный перехватывать пакеты, может сымитировать любой запрос пользователя.

Базовая аутентификация предоставляет (низкий) уровень защиты, подобный тому, который обеспечивается при подключении по протоколу Telnet или FTP. Эти методы также передают пароли в виде простого текста. Дайджест-аутентификация несколько более безопасна и шифрует пароли перед передачей. Использование протокола SSL и цифровых сертификатов позволяет надежно защитить все части транзакций в Web.

Методы надежной защиты рассматриваются в главе 15. Однако во многих ситуациях наиболее подходящим будет быстрый и относительно незащищенный метод, такой как базовая аутентификация.

Базовая аутентификация позволяет защитить именованные области и требует от пользователей ввода правильного имени и пароля. Области именованные, поэтому на одном сервере может быть существовать множество областей. Различные файлы и каталоги на одном сервере могут принадлежать разным областям, каждая из которых за-

щищена своими наборами имен пользователей и паролей. Именованные области позволяют также сгруппировать в одну область несколько каталогов на одном физическом или виртуальном узле и защитить всю область одним паролем.

Использование базовой аутентификации в PHP

PHP-сценарии, в основном, можно назвать кросс-платформенными, но использование базовой аутентификации базируется на переменных среды, устанавливаемых сервером. Сценарий HTTP-аутентификации должен определять тип сервера и вести себя соответствующим образом в зависимости от того, выполняется ли он как модуль Apache на сервере Apache или как ISAPI-модуль на сервере IIS. Показанный в листинге 14.4 сценарий будет выполняться на обоих серверах.

Листинг 14.4 http.php — базовую HTTP-аутентификацию можно включить средствами PHP.

```
<?
// Если используется сервер IIS, потребуется установить переменные
// среды $PHP_AUTH_USER и $PHP_AUTH_PW
if (substr($SERVER_SOFTWARE, 0, 9) == "Microsoft" SS
    !isset($PHP_AUTH_USER) &&
    !isset($PHP_AUTH_PW) &&
    substr($HTTP_AUTHORIZATION, 0, 6) == "Basic "
   )
{
  list($PHP_AUTH_USER, $PHP_AUTH_PW) =
    explode(":",base64_decode(substr($HTTP_AUTHORIZATION,6)));
}
// Замените этот оператор if запросом к базе данных или чем-то подобным
if ($PHP_AUTH_USER != "user" I | $PHP_AUTH_PW != "pass")
{
  // Посетитель еще не передал деталей или его
  // имя и пароль неправильные
  header('WWW-Authenticate:Basic realm="Realm-Name"');
  if (substr($SERVER_SOFTWARE, 0, 9) == "Microsoft")
    header("Status: 401 Unauthorized");
  else
    header("HTTP/1.0 401 Unauthorized");

  echo "<h1>Go Away!</h1>";
  echo "You are not authorized to view this resource.";
}
else
{
  // посетитель предоставил правильную информацию
  echo "<h1>Here it is!</h1>";
  echo "<p>I bet you are glad you can see this secret page.";
}
?>
```

Код в листинге 14.4 работает так же, как и код из предыдущего листинга в этой главе. Если пользователь не передал данных аутентификации, ему выдается запрос на аутентификацию. Если пользователь предоставил неправильную информацию, для него отображается сообщение об отказе в доступе. Если же информация правильная, пользователь увидит содержимое страницы

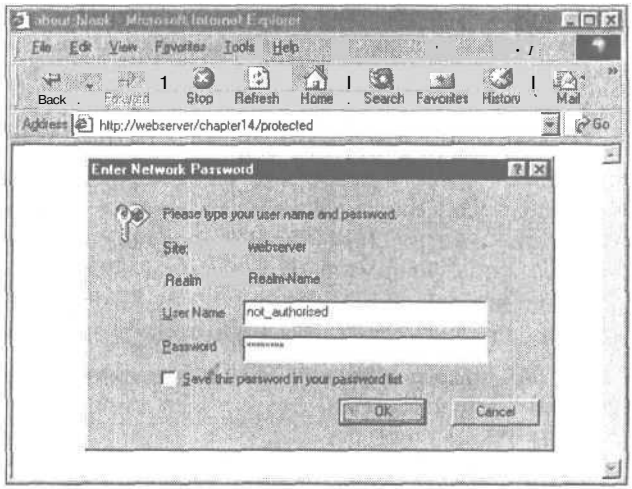


РИСУНОК 14.4.
При использовании HTTP-аутентификации за внешний вид диалогового окна регистрации отвечает браузер пользователя.

Интерфейс данного примера отличается от интерфейса предыдущих примеров. Мы не создаем HTML-форму для ввода имени и пароля. Диалоговое окно для аутентификации выведет браузер пользователя. Некоторые рассматривают это как улучшение, другие предпочитают иметь полный контроль над визуальными аспектами интерфейса. На рис. 14.4 показано диалоговое окно регистрации, которое выводится в браузере Internet Explorer.

Поскольку для аутентификации применяются встроенные возможности браузеров, последние демонстрируют некоторую осторожность в обработке неудачных попыток аутентификации. Internet Explorer дает пользователю три попытки аутентификации, и если все они проходят неудачно, выводится сообщение об отказе в доступе. Netscape Navigator предоставляет неограниченное число попыток, но между попытками выводит диалоговое окно с запросом о повторе "Authentication Failed. Retry?". Netscape отображает сообщение об отказе в доступе, только когда пользователь щелкает на кнопке Cancel.

Как и код из листингов 14.1 и 14.2, код данного примера можно вставить в начало каждого файла, который требуется защитить. Это можно сделать вручную или автоматически для каждого файла в каталоге.

Использование базовой аутентификации при помощи файлов .htaccess сервера Apache

Результат, подобный результату предыдущего сценария, может быть достигнут и без написания PHP-сценария. Сервер Apache обладает множеством различных методов аутентификации, которые можно использовать для определения правильности введенных пользователем данных. Наиболее простой из этих методов — это задействование функциональности модуля **mod_auth,** которая сравнивает пары имя-пароль со строками текстового файла на сервере.

Чтобы получить на экране пользователя результат предыдущего сценария, потребуется создать два HTML-файла — один для содержимого страницы и один для сообщения об отказе в доступе. В предыдущих примерах были опущены некоторые HTML-дескрипторы, но на самом деле в HTML-файлах присутствуют дескрипторы **<html>** и **<body>**.

В листинге 14.5 показан код HTML-файла, который будут видеть пользователи, прошедшие аутентификацию. Файл называется **content.html.** Листинг 14.6 содержит HTML-код страницы с сообщением об отказе в доступе. Этот файл называется **rejection.html.** Создавать страницу, которая будет отображаться в случае ошибки, вовсе не обязательно, но наличие такой страницы с полезной информацией свидетельствует о хорошем стиле и должном профессионализме. Учитывая, что такая страница будет отображаться пользователю при неудачной попытке войти в закрытую зону, полезной может быть информация о том, как зарегистрироваться и получить пароль, или как пароль может быть сброшен в первоначальное состояние и отправлен по электронной почте, если вдруг он забыт.

Листинг 14.5. content.html — пример содержимого.

```
<html><body>
<hl>Here it is!</h1>
<p>I bet you are glad you can see this secret page.
</body></html>
```

Листинг 14.6. rejection.html — простое сообщение об ошибке 401.

```
<html><body>
<h1>Go Away!</h1>
<p>You are not authorized to view this resource.
</body></html>
```

В этих файлах нет ничего нового. Интересным для данного примера является файл из листинга 14.7. Этот файл следует назвать **.htaccess. Он** управляет доступом к файлам и подкаталогам каталога, в котором он расположен.

Листинг 14.7 .htaccess — с помощью файла .htaccess можно установить различные параметры сервера Apache, включая активацию аутентификации.

```
ErrorDocument 401 /chapter14/rejection.html
AuthUserFile /home/book/.htpass
AuthGroupFile /dev/null
AuthName "Realm-Name"
AuthType Basic
require valid-user
```

В листинге 14.7 показан файл **.htaccess** для включения аутентификации в каталоге. С помощью этого файла можно установить различные параметры, однако в этом примере все шесть строк относятся к аутентификации.

Первая строка

```
ErrorDocument 401 /chapter14/rejection.html
```

указывает серверу Apache, какой документ следует отобразить посетителям, не прошедшим аутентификацию. Директиву **ErrorDocument** можно использовать несколько раз в одном файле, чтобы указать собственные страницы для сообщений о других ошибках протокола HTTP, например, об ошибке 404. Синтаксис этой директивы таков:

```
ErrorDocument error_number URL
```

Важно, чтобы страница с сообщением об ошибке 401 была доступна для всех посетителей. Не имеет смысла поддерживать собственную страницу с сообщением о не-

удачной аутентификации, если эта страница располагается в каталоге, в который можно попасть только после успешного прохождения аутентификации.

Строка

```
AuthUserFile /home/book/.htpass
```

указывает серверу где искать файл, содержащий пользовательские пароли. Этот файл часто называют **.htpass**, но можно выбрать произвольное имя файла. Не важно как этот файл называется, важно где он расположен. Этот файл не следует располагать в рамках дерева Web-каталогов т.е. в одном из каталогов, из которого его можно загрузить из Web-сервера. Файл **.htpass** для данного примера показан в листинге 14.8.

В дополнение к возможности указывать санкционированных пользователей индивидуально, можно указать, что доступ к ресурсу разрешен санкционированным пользователям только определенной группы. В данном примере это не предпринимается и строка

```
AuthGroupFile /dev/null
```

устанавливает параметр **AuthGroupFile** указывающим на **/dev/null** — специальный файл в UNIX-системах, который является гарантированно пустым.

Как и в примере с PHP, для использования HTTP-аутентификации защищаемой области следует присвоить имя. Это выполняет следующая строка

```
AuthName "Realm-Name"
```

Имя области может быть произвольным, но следует помнить, что это имя будет видно посетителям. Дабы еще раз напомнить, что имя области в этом примере следует изменить, было выбрано имя **"Realm-Name"** ("Имя-Области").

Поскольку сервер поддерживает различные методы аутентификации, следует указать, какой именно метод следует использовать.

```
AuthType Basic
```

Также следует указать, кому разрешен доступ. Можно указать определенные группы, определенных пользователей, или, как сделано в примере, всех санкционированных пользователей.

Строка

```
require valid-user
```

указывает, что доступ разрешен всем пользователям, успешно прошедшим аутентификацию.

Листинг 14.8. .htpass — файл паролей хранит имена и зашифрованные пароли для каждого пользователя.

```
user1:0nRp9M80GS7zM
user2:nC13sOTOhp.ow
user3:yjQMCPWjXFTzU
user4:LOmlMEi/hAme2
```

Каждая строка в файле **.htpass** содержит имя пользователя и зашифрованный пароль, разделенные двоеточием.

Точное содержимое этого файла может варьироваться. Чтобы создать такой файл, воспользуйтесь небольшой утилитой **htpasswd**, которая распространяется вместе с сервером Apache.

Эту программу можно использовать одним из двух следующих способов.

```
htpasswd [-cmdps] passwordfile username
```

ИЛИ

```
htpasswd -b[cmdps] passwordfile username password
```

Единственный применяемый аргумент — это -c. Этот аргумент указывает **htpasswd,** что требуется создать новый файл. Используйте этот аргумент для создания первого пользователя. Будьте внимательны и не используйте этот аргумент для следующих пользователей, поскольку если файл уже существует, то **htpasswd** удалит его и создаст новый файл с тем же именем.

Необязательные аргументы m, d, p, и s следует применять для выбора алгоритма шифрования (включая отказ от шифрования).

Аргумент b заставляет утилиту ожидать пароль в качестве аргумента, а не запрашивать его отдельно. Этот аргумент полезен для запуска **htpasswd** в неинтерактивном режиме как часть командного файла, но этот аргумент не следует использовать при вызове htpasswd из командной строки.

Приведенные ниже команды использовались для создания файла, показанного в листинге 14.8.

```
htpasswd -bc /home/book/.htpass user1 pass1
htpasswd -b /home/book/.htpass user2 pass2
htpasswd -b /home/book/.htpass user4 pass3
htpasswd -b /home/book/.htpass user4 pass4
```

Аутентификацию такого типа легко настроить, но с ней связано несколько проблем.

Имена и пароли пользователей хранятся в текстовом файле. Каждый раз, когда браузер запрашивает файл, защищенный через **.htaccess,** сервер должен проанализировать этот файл и после этого проанализировать еще и файл **.htpass,** чтобы попытаться найти соответствующее имя пользователя и пароль. Вместо использования файла **.htaccess** то же самое можно указать в файле **httpd.conf** — базовом файле конфигурации Web-сервера. В то время как файл htaccess анализируется при каждом запросе, файл **httpd.conf** анализируется только в момент запуска сервера. Подобный подход ускорит обработку запросов, но при внесении изменений потребует останова и перезапуска сервера.

Независимо от места хранения директив сервера, файл паролей анализируется при каждом запросе. Это означает, что подобно другим рассмотренным методам, которые используют двумерный файл, данный метод не годится в случае обработки сотен тысяч пользователей.

Использование базовой аутентификации в IIS

Как и сервер Apache, сервер IIS также поддерживает HTTP-аутентификацию. В Apache используется подход, свойственный UNIX-системам — управление сервером осуществляется путем редактирования текстовых файлов. Как вы уже догадались, управлять сервером IIS можно через диалоговые окна.

В Windows 2000 изменить конфигурацию Internet Information Server 5 (IIS5) можно с помощью утилиты Internet Services Manager, которая расположена в группе Administrative Tools (Инструменты администрирования) в панели управления.

Окно Internet Services Manager должно выглядеть подобно окну, показанному на рис. 14.5. Древовидный список в левой части окна показывает, что на компьютере с именем windows-server запущено несколько служб. Нас интересует служба Default Web Site — Web-сайт, установленный по умолчанию. На этом сайте имеется каталог **protected,** который содержит файл **content.html.**

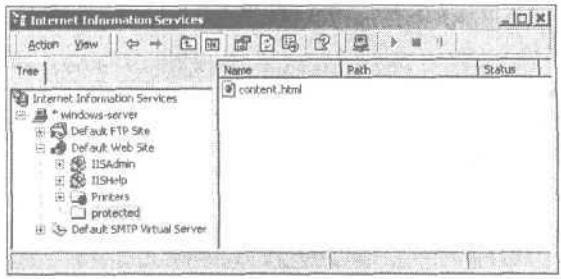


РИСУНОК 14.5.
Консоль Microsoft Management Console позволяет настраивать сервер Internet Information Server 5.

Чтобы активировать базовую аутентификацию для каталога **protected,** щелкните правой кнопкой мыши на этом каталоге и выберите пункт Properties (Свойства) в контекстном меню.

Диалоговое окно Properties позволяет изменить многие параметры для этого каталога. Нам интересны вкладки Directory Security (Защита каталогов) и Custom Errors (Настройка ошибок). Обратите внимание на раздел Anonymous Access (Анонимный доступ) and Authentication Control (Управление аутентификацией) во вкладке Directory Security. Щелчок на кнопке Edit приводит к отображению диалогового окна, показанного на рис. 14.6.

В этом диалоговом окне можно отключить анонимный доступ и включить базовую аутентификацию. Если установить параметры в соответствие с рис. 14.6, то файлы в выбранном каталоге смогут просматривать только те пользователи, которые предоставили соответствующее имя и пароль.

Чтобы повторить поведение предыдущего примера, нужно еще создать страницу, сообщающую пользователю о вводе некорректных данных аутентификации. Перейти в закладку Custom Errors можно после закрытия диалогового окна Authentication Methods.

Закладка Custom Errors, показанная на рис. 14.7, позволяет ассоциировать ошибки с сообщениями об ошибках. В этом примере мы воспользовались тем же файлом сообщения об отказе в доступе, который применялся раньше. Исходный код файла **rejection.html** можно найти в листинге 14.6. Сервер IIS выводит более точное, чем сервер Apache, сообщение об ошибке благодаря тому, что IIS в дополнение к коду ошибки выдает еще и причину ее появления. Для ошибки 401, представляющей неудачную аутентификацию, сервер IIS выдает пять различных причин. Можно установить различные сообщения для всех причин, но для данного примера решено заменить сообщением об отказе только два сообщения, которые могут появиться в результате выполнения примера.

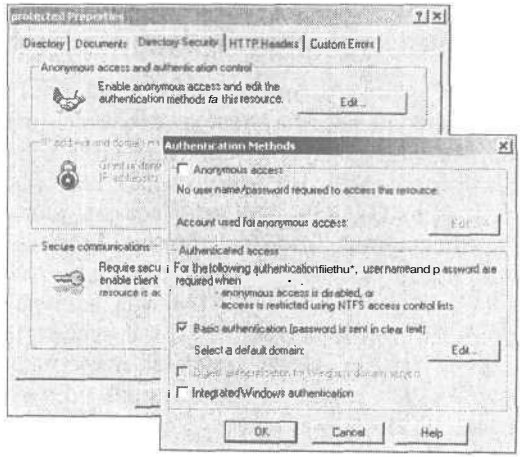


РИСУНОК 14.6.
Сервер IIS5 по умолчанию разрешает анонимный доступ, но позволяет также включить аутентификацию.

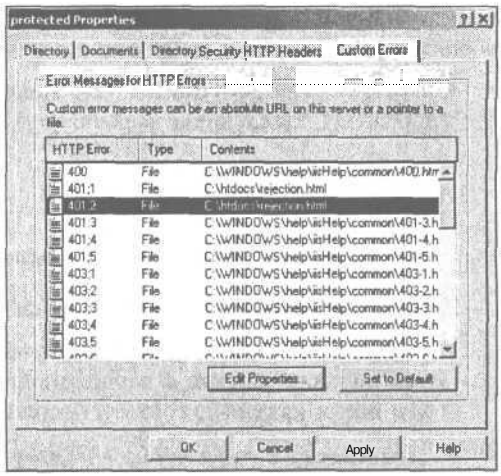


РИСУНОК 14.7.
Закладка Custom Errors позволяет ассоциировать собственные сообщения об ошибках с собственно ошибками.

Это все, что нужно проделать в плане включения аутентификации для выбранного каталога в сервере IIS5. Как и другие приложения Windows, сервер IIS5 настроить проще, чем аналогичную службу в UNIX, но сложнее копировать с компьютера на компьютер или из каталога в каталог. Кроме того, сервер IIS5 можно случайно настроить так, что ваш компьютер окажется вообще незащищенным.

Большая проблема с сервером IIS состоит в том, что он аутентифицирует пользователей, сравнивая переданную информацию с информацией о пользователях на том же компьютере. Если нужно разрешить доступ пользователю **"john"** с паролем **"password",** потребуется создать учетную запись для этого пользователя с тем же именем и паролем на этом компьютере или в домене этого компьютера. При создании учетных записей для аутентификации в Web следует быть очень осторожным. Удостоверьтесь, что для таких учетных записей установлены только те права, которые необходимы для просмотра Web-страниц, и что учетная запись не обладает правами доступа к другим службам, подобным Telnet.

Использование аутентификации через модуль mod_auth_mysql

Как уже упоминалось ранее, можно легко и эффективно использовать модуль **mod_auth** в сервере Apache. Но этот модуль не очень практичен для загруженных сайтов с большим количеством пользователей, поскольку **mod_auth** хранит пользовательские пароли в текстовом файле.

К счастью, при использовании модуля **mod_auth_mysql** можно получить большую часть простоты применения **mod_auth** в сочетании со скоростью баз данных. Этот модуль работает подобно **mod_auth,** но благодаря использованию базы данных, способен быстрее производить поиск в больших списках пользователей.

Чтобы использовать этот модуль, его следует скомпилировать и установить, или попросить сделать это системного администратора.

Установка модуля mod_auth_mysql

Для задействования модуля **mod_auth_mysql** сначала следует установить сервер Apache и MySQL, руководствуясь инструкциями из приложения А. После этого необходимо

выполнить еще несколько шагов. В файлах **README и USAGE** находятся достаточно хорошие инструкции, а ниже рассматривается суть этих инструкций.

1. Отыщите дистрибутив данного модуля. Он находится на сопровождающем книгу CD-ROM. Последнюю версию модуля можно выгрузить с сайтов

   ```
   http://www.zend.com
   ```

   или

   ```
   http://www.mysql.com.downloads/contrib.html
   ```

2. Воспользуйтесь утилитами zip и tar для распаковки исходных кодов.

3. Перейдите в каталог **mod_auth_mysql** и запустите сценарий **configure.** Этот сценарий запросит данные о расположении MySQL и исходных кодов сервера Apache. Мы ввели каталоги, соответствующие конфигурации нашей системы:

   ```
   ./configure --with-mysql=/var/mysql  --with-apache=/src/apache_1.3.12
   ```

   Расположение ваших каталогов может отличаться.

4. Запустите **make,** а затем **make install.** При запуске сценария **configure** для сервера Apache следует добавить такую строку

   ```
   --activate-module=src/modules/auth_mysql/libauth_mysql.a
   ```

   Для конфигурирования системы на нашем компьютере использовалась следующая строка:

   ```
   ./configure --enable-module=ssl \
   --activate-module=src/modules/php4/libphp4.a \
   --enable-module=php4 --prefix=/usr/local/apache --enable-shared=ssl  \
   --activate-module=src/modules/auth_mysql/libauth_mysql.a
   ```

5. После выполнения оставшихся инструкций из приложения A, потребуется создать базу данных и таблицу, в которой будут храниться данные аутентификации. Не следует создавать отдельную базу данных и таблицу, можно воспользоваться существующими таблицами, такими как база данных **auth** из более ранних примеров в этой главе.

6. В файл **htpd.conf** следует добавить строку параметров для модуля **mod_auth_mysql.** Эти параметры будут использоваться для подключения к базе данных, а строка в файле конфигурации выглядит так:

   ```
   Auth_MySQL_Info hostname user password
   ```

Заработало?

Простейший способ проверки, работают ли откомпилированные модули — это проверить, запускается ли сервер Apache. Для запуска сервера Apache воспользуйтесь следующей строкой:

```
/usr/local/apache/bin/apachectl startssl
```

Если сервер запустится с директивой **Auth_MySQL_Info** в файле конфигурации **httpd.conf,** значит, модуль **mod_auth_mysql** успешно установлен.

Использование *модуля mod_auth_mysql*

После успешной установки использовать модуль **mod_auth_mysql** не намного сложнее, чем модуль **mod_auth**. В листинге 14.9 показан пример файла **.htaccess,** который позволит аутентифицировать пользователей с шифрованием паролей, которые будут храниться в созданной в начале главы базе данных.

Листинг 14.9. .htaccess — этот файл включает аутентификацию пользователей с применением базы данных MySQL

```
ErrorDocument 401 /chapter14/rejection.html

AuthName "Realm Name"
AuthType Basic

Auth_MySQL_DB auth
Auth_MySQL_Encryption_Types MySQL
Auth_MySQL_Password_Table auth
Auth_MySQL_Username_Field name
Auth_MySQL_Password_Field pass

require valid-user
```

Можно заметить, что большая часть этого файла совпадает с файлом, показанным в листинге 14.7. Все еще указывается файл с сообщением об ошибке 401 (неудачная аутентификация). Снова указана базовая аутентификация и имя области аутентификации. И как в листинге 14.7, доступ разрешен любому пользователю, который успешно прошел аутентификацию. Поскольку применяется модуль **mod_auth_mysql** и не хотелось бы использовать установки по умолчанию, в листинге 14.9 указываются дополнительные директивы для настройки. Директивы **Auth_MySQL_DB, Auth_MySQL_Password_Table, Auth_MySQL_Username_Field, и Auth_MySQL_Password_Field** используются для указания, соответственно, имени базы данных, таблицы и полей имени и пароля пользователя.

В листинге присутствует директива **Auth_MySQL_Encryption_Types,** которая обеспечивает выбор типа шифрования. В примере выбрано шифрование паролей MySQL. Для этой директивы приемлемыми являются значения **Plaintext, Crypt_DES** или **MySQL.** Значение **Crypt_DES** активизирует использование стандартного для UNIX алгоритма шифрования DES.

С точки зрения пользователя, пример с модулем **mod_auth_mysql** работает в точности так, как пример с модулем **mod_auth.** Пользователь видит диалоговое окно в своем браузере. Если пользователь проходит аутентификацию, он увидит защищенное содержимое страницы, а если нет — сообщение об ошибке.

Для большинства Web-сайтов аутентификация с помощью модуля **mod_auth_mysql** подходит практически идеально. Этот метод позволяет использовать любой удобный механизм для создания новых учетных записей в базе данных. Для большей гибкости и контроля над частями Web-страниц придется создавать собственный метод аутентификации с применением PHP и MySQL.

Создание собственного метода аутентификации

Мы рассмотрели способы создания собственных методов аутентификации, включая некоторые недостатки и соглашения, а также встроенные методы аутентификации, не такие гибкие как ваш собственный код. После изучения методов управления сеансами появится возможность создавать собственные методы аутентификации с меньшим количеством компромиссов, чем было сделано в этой главе.

В главе 20 рассматривается создание системы аутентификации пользователей, которая благодаря использованию сеансов для передачи переменных между страницами, избегает некоторых проблем, встретившихся в этой главе.

В главе 24 будет показан процесс применения созданной системы на реальном проекте. В ней вы узнаете, как создать качественную систему аутентификации.

Дополнительная информация

Детали HTTP-аутентификации описаны в документе RFC 2617, который доступен по адресу

```
http://www.rfc-editor.org/rfc/rfc2617.txt
```

Документация по модулю **mod_auth**, управляющему базовой аутентификацией в сервере Apache, находится на сайте

```
http://www.apache.org/docs/mod/mod_auth.html
```

Документацию по модулю **mod_auth_mysql** можно найти по адресу

```
http://www.zend.com
```

ИЛИ

```
http://www.express.ru/docs/mod_auth_mysql_base.html
```

Что дальше

Следующая глава описывает методику защиты данных на всех стадиях обработки — на стадии ввода, передачи и хранения. В ней рассматривается использование SSL, цифровых сертификатов и шифрования.

Глава 15

Реализация безопасных транзакций в PHP и MySQL

В этой главе объясняется, как можно безопасно обрабатывать пользовательские данные при вводе, пересылке и сохранении. Это позволяет производить транзакцию между пользователем и сервером, защищенную с обеих сторон. Обсуждаемые в этой главе темы включают:

- Обеспечение безопасности транзакций
- Использование слоя безопасных сокетов
- Обеспечение безопасного хранения данных
- Зачем хранить номера кредитных карточек?
- Использование шифрования в PHP

Обеспечение безопасности транзакций

Обеспечение безопасности транзакций в Internet сводится к проверке потока информации в системе и удостоверение того, что в каждой точке она является защищенной. В контексте сетевой безопасности не существует абсолюта. Нет систем, которые можно сделать полностью недоступными для проникновения. Под безопасностью здесь понимается то, что уровень усилий, необходимых для взлома системы или пересылаемой информации, сравним со значимостью этой информации.

Для эффективного применения усилий по защите необходимо проверять поток информации во всех частях системы. Поток пользовательской информации в типовом приложении, написанном с применением PHP и MySQL, показан на рис. 15.1.

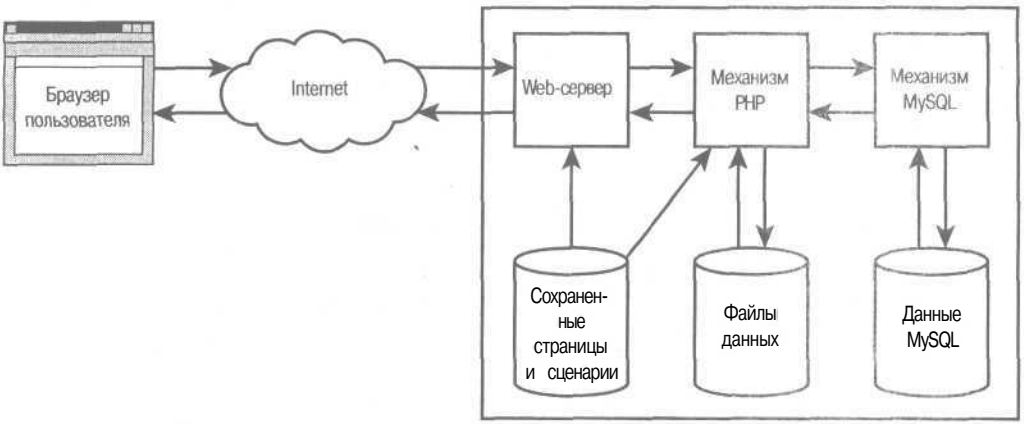


РИСУНОК 15.1 *Схема элементов сохранения и обработки пользовательской информации в среде типового Web-приложения.*

Подробности каждой транзакции, происходящей в системе, могут быть различными в зависимости и от внутренней структуры системы, и от пользовательских данных и действий, послуживших источником транзакции. Их можно исследовать следующим образом. Каждая транзакция между Web-приложением и пользователем начинается с того, что пользовательский браузер отправляет через Internet запросы к Web-серверу. Если страница содержит PHP-сценарий, то Web-сервер передает полномочия обработки страницы в механизм PHP.

PHP-сценарий может читать или записывать данные на диск. Он также может включать PHP- или HTML-файлы с использованием **include()** или **require()** и отправлять SQL-запросы демону MySQL, получая от него ответы. Механизм MySQL отвечает за чтение и запись собственных данных на диск.

Такая система состоит из трех основных частей:

- Пользовательская машина
- Internet
- Разработанная система

Мы рассмотрим вопросы безопасности для каждой из частей, но. разумеется, пользовательская машина и Internet находятся за пределами вашего контроля.

Пользовательская машина

На машине пользователя выполняется Web-браузер — и это все, что мы знаем. Мы не можем управлять такими факторами, как настройка безопасности этой машины. Более того, необходимо помнить, что машина может быть абсолютно незащищенной или даже служить в качестве общедоступного терминала в библиотеке, школе или Internet-кафе.

Существует несколько различных браузеров, обладающих разными возможностями. Если принять во внимание только последние версии двух популярных браузеров, то большинство различий в них влияет лишь на то, как форматируется и выводится HTML-код, однако нам требуется исследовать вопросы безопасности.

Следует обратить внимание, что некоторые могут отключать свойства, рассматриваемые как опасные касаемо защиты системы, например, Java, cookie-наборы или JavaScript. Если вы используете эти свойства, то либо убедитесь, что ваше приложение работает при условии их отключения, либо обеспечьте упрощенный интерфейс, который позволит упомянутым пользователям успешно работать с вашим сайтом.

Пользователи за пределами США и Канады могут иметь Web-браузеры, использующие лишь 40-битное шифрование. Хотя правительство США в январе 2000 г. изменило законодательство, позволив экспортировать усложненное шифрование (в страны, не облагаемые эмбарго) и 128-битное шифрование стало доступным большинству пользователей, некоторые из них могли еще не обновить версии своих браузеров. Если только вы не гарантируете безопасность своим пользователям в тексте своего сайта, вам, как Web-разработчику, нет надобности беспокоиться об этих вопросах. SSL автоматически позволяет серверу и браузеру пользователя взаимодействовать на максимально возможном уровне безопасности.

К сожалению, нельзя быть уверенным в том, что именно Web-браузер соединяется с сайтом, используя предназначенный для этого интерфейс. Запросы сайта могут приходить с другого сайта, "ворующего" изображения или данные, или же от кого-либо, использующего программное обеспечение наподобие библиотеки cURL с целью обхода защиты.

О библиотеке cURL, позволяющей эмулировать запросы браузера, будет рассказано в главе 17. С одной стороны, это полезно для разработчиков, однако может быть задействовано и в злонамеренных целях.

Хотя нет возможности изменить или управлять настройкой пользовательских машин, беспокоиться об этом не следует. Изменчивость машин пользователей играет роль только в том, какая функциональность может обеспечиваться в сценариях серверной (PHP), а какая — клиентской (JavaScript) стороны.

Функциональность, обеспечиваемая PHP, может быть совместимой с любым браузером, поскольку ее результатом является лишь HTML-страница. Использование чего-либо более сложного, нежели самый примитивный вариант JavaScript, приводит к необходимости учета свойств индивидуальных версий браузеров.

С точки зрения защиты, для проверки данных стоит использовать серверные сценарии, так как в таком случае исходный код не будет виден пользователю. При проверке данных средствами JavaScript пользователи могут видеть исходный код и, возможно, перехитрить его.

Все требуемые данные можно сохранять на сервере (файлы и записи базы данных) или на пользовательской машине (cookie-наборы). Использование cookie-наборов для сохранения ограниченных данных (ключ сеанса) рассматривается в главе 20.

Большая часть данных находится на Web-сервере или в базе данных. Существует немало причин, по которым следует хранить на пользовательской машине настолько мало информации, насколько это возможно. Информация, сохраненная за пределами системы, не придает уверенности в том, что она надежно сохранена, что пользователь не удалит ее, или же что он не изменит ее в целях взлома системы.

Internet

Как и в случае пользовательской машины, характеристики Internet также не поддаются контролю, однако при разработке системы игнорировать их нельзя.

Internet обладает множеством замечательных свойств, однако по определению не является безопасной сетью. При пересылке информации из одной точки в другую сле-

дует помнить, что ее могут просматривать и даже изменять другие пользователи. Об этом упоминалось в главе 13. Памятуя об этом, можно решить, что следует предпринять.

Итак, доступно несколько возможностей:

- Пересылать информацию в любом случае, не забывая, что она может уже не являться приватной.
- Зашифровать информацию перед пересылкой, чтобы сохранить приватность и предотвратить преступное использование.
- Решить, что информация является слишком чувствительной к риску перехвата, и подыскать другие способы ее распространения.

Internet — это еще и исключительно анонимная среда. Очень трудно удостовериться, что лицо является именно тем, за кого себя выдает. Даже если в этом можно убедиться для собственного удовлетворения, это будет непросто доказать, преодолевая достаточно высокий уровень сомнения в таких организациях, как, например, суд. А это вызывает проблемы с отказом от обязательств, о чем упоминалось в главе 13.

Короче говоря, приватность и отказ от обязательств — это большие проблемы при проведении транзакций через Internet.

Есть, как минимум, два способа для защиты информации, поступающей на или с Web-сервера через Internet:

- SSL (Secure Sockets Layer, Слои безопасных сокетов)
- S-HTTP (Secure Hypertext Transfer Protocol, Безопасный протокол передачи гипертекстов)

Обе технологии обеспечивают приватный, защищенный от взлома обмен сообщениями и аутентификацию, однако если SSL широко распространен, то S-HTTP применяется еще нечасто. SSL подробно рассматривается далее в главе.

Ваша система

Той частью Вселенной, которой вы можете полностью управлять, является ваша система. Ее компоненты обведены на рис. 15.1 сплошной линией. Эти компоненты могут быть физически разделены и соединены сетью, либо же существовать на одной машине.

Можно практически не беспокоиться о безопасности информации, пересылкой которой через Web занимаются продукты независимых разработчиков. Авторы этих программ уделили им достаточное внимание. Если используется последняя версия широко распространенного продукта, то все известные на текущий момент проблемы можно узнать, воспользовавшись любым поисковым сайтом. Очень важно постоянно следить за обновлением такого рода информации.

Если инсталляция и конфигурирование входят в ваши обязанности, следует побеспокоиться о том, как проинсталлировано и сконфигурировано программное обеспечение. Немало ошибок в защите системы являются результатом неследования предупреждениям в документации или же общими вопросами системного администрирования, которые представляют собой тему отдельной книги. Поэтому вам стоит или приобрести хорошую книгу по администрированию операционной системы, или нанять эксперта по этому вопросу.

Следует отметить, что в общем случае более безопасным (и эффективным) является установка PHP как модуля SAPI Web-сервера, а не запускать его через CGI.

Первое, о чем следует позаботиться — что должны, а чего не должны делать ваши сценарии.

Какие потенциально чувствительные к перехвату данные ваше приложение будет пересылать пользователю через Internet? Какие данные оно будет запрашивать у пользователя? Если пересылается информация, представляющая собой приватную транзакцию между вами и пользователем, необходимо прибегнуть к SSL.

Здесь уже обсуждался вопрос применения SSL между пользовательским компьютером и сервером. Следует также рассмотреть ситуацию, когда данные пересылаются по сети от одного компонента системы другому. Так происходит, например, тогда, когда база данных MySQL находится не на одной и той же машине с Web-сервером. PHP соединяется с сервером MySQL по TCP/IP, а такое соединение не шифруется. Если обе машины находятся в локальной сети, следует убедиться, что сеть надежно защищена. Если они взаимодействуют через Internet, то система, возможно, работает несколько медленнее, а к соединению необходимо относиться так же, как и к любому другому соединению через Internet.

PHP не содержит естественного способа произвести такое соединение по SSL. Команда **fopen()** поддерживает HTTP, но не HTTPS. Поддержка SSL содержится в библиотеке cURL. О ней рассказано в главе 17.

Очень важно, чтобы когда пользователи думают, что они имеют дело с вами, они действительно имели дело с вами. Регистрация с помощью цифрового сертификата поможет защитить посетителей от обманных действий (когда кто-то другой пытается выдать свой сайт за ваш), позволит использовать SSL, не выдавая пользователям предупреждающего сообщения, и обеспечит электронному предприятию определенную долю уважения.

Проверяют ли сценарии данные, вводимые пользователями?

Действительно ли защищена сохраняемая информация?

Ответы на эти и другие вопросы сформулированы в следующих разделах этой главы.

Использование слоя безопасных сокетов

Слой безопасных сокетов (SSL) представляет собой набор протоколов, изначально разработанный компанией Netscape с целью обеспечения безопасного соединения Web-серверов и Web-браузеров. С тех пор он стал неофициальным стандартным методом для обмена чувствительной информацией между браузерами и серверами.

Достаточно хорошо поддерживаются SSL как версии 2, так и версии 3. Большинство Web-серверов либо включают функциональность SSL, либо допускают ее подключение в виде модуля. Internet Explorer и Netscape Navigator поддерживают SSL, начиная с версии 3.

Сетевые протоколы и программное обеспечение, использующее их, рассматриваются зачастую как набор слоев. Каждый слой передает данные или запрашивает службы из слоя ниже или выше. Такой стек протоколов показан на рис. 15.2.

Когда для пересылки информации используется HTTP, этот протокол вызывает *Transmission Control Protocol (TCP, Протокол управления передачей),* который, в свою очередь, зависит от *Internet Protocol (IP, Межсетевого протокола).* Последний протокол требует протокола, управляющего сетевыми аппаратными средствами и преобразующего пакеты данных в электрические сигналы, отправляемые в точку назначения.



РИСУНОК 15.2 *Стек протоколов, используемый протоколом слоя приложения, например, HTTP.*

HTTP называется протоколом слоя приложения. Протоколами такого типа являются FTP, SMTP и telnet (как показано на рис. 15.2), а также POP, ШАР и ряд других. TCP — это один из двух протоколов транспортного слоя, используемого в сетях TCP/IP. IP — протокол сетевого слоя. Хост сетевого слоя отвечает за соединение хоста (компьютера) с сетью. Протоколы этого слоя в данном стеке TCP/IP не указаны, поскольку для различных типов сетей на этом уровне требуются различные протоколы.

При отправке данные пересылаются по этому стеку от приложения к физической сетевой среде. При получении данные проходят от физической сети через стек к приложению.

Использование SSL добавляет к описанной модели еще один прозрачный слой. Слой SSL находится между транспортным слоем и слоем приложения. Модифицированная схема показана на рис. 15.3. Слой SSL изменяет данные, поступившие от приложения HTTP, до их передачи транспортному слою.

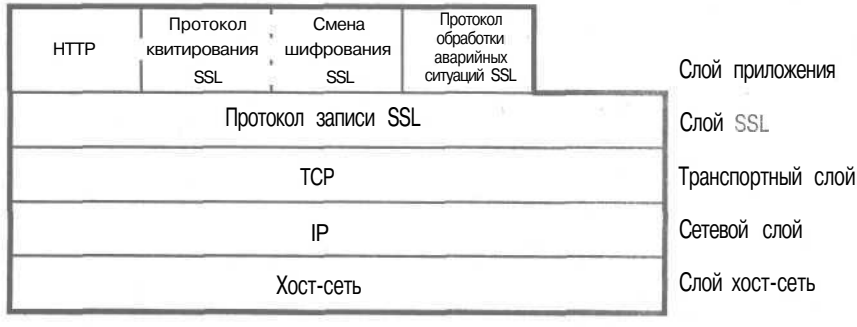


РИСУНОК 15.3 *SSL добавляет дополнительный слой в стек протоколов, а также в слой приложения — для управления своими действиями.*

Теоретически SSL может обеспечить среду безопасной пересылки и другим протоколам кроме HTTP, но обычно его используют только для HTTP. Возможность использования других протоколов связана с тем, что слой SSL является исключительно прозрачным, т.е. он обеспечивает такой же интерфейс протоколам выше него, как и транспортный слой. Кроме того, он прозрачно обеспечивает установку соединения, шифрование и дешифрование.

Когда Web-браузер соединяется с безопасным Web-сервером через HTTP, обеим сторонам необходимо воспользоваться протоколом установки соединения (квитирования), позволяющим договориться об аутентификации и кодировании.

Последовательность установки соединения включает в себя следующие шаги:

1. Браузер соединяется с сервером, поддерживающим SSL, и запрашивает у него аутентификацию.
2. Сервер отправляет свой цифровой сертификат.

3. Сервер может дополнительно (крайне редко) запросить аутентификацию у браузера.
4. Браузер посылает список поддерживаемых им алгоритмов шифрования и хэширования. Сервер выбирает наиболее сложное шифрование из поддерживаемых им.
5. Браузер и сервер генерируют ключи сеанса:
   5.1. Браузер получает открытый ключ сервера из его цифрового сертификата и использует его для шифрования случайного числа.
   5.2. Сервер отвечает отправкой случайных данных в текстовом формате (если только браузер не выслал цифровой сертификат в ответ на запрос сервера – в этом случае сервер использует открытый ключ браузера).
   5.3. Ключи шифрования для этого сеанса генерируются по случайным данным с использованием хэш-функции.

Генерирование качественных случайных данных, дешифрация цифровых сертификатов, генерирование ключей и использование криптографии по открытому ключу требует времени, поэтому процедура установки соединения выполняется не сразу. К счастью, результаты кэшируются, поэтому, если тот же самый браузер соединяется с тем же сервером, процесс установки соединения требует времени только один раз.

Пересылка данных по SSL-соединению проходит следующие стадии:

1. Данные разбиваются на пакеты.
2. Каждый пакет (необязательно) сжимается.
3. Каждый пакет имеет коды аутентификации сообщения (message authentication code, MAC), вычисленные с использованием хэш-алгоритма.
4. MAC и сжатые данные комбинируются и шифруются.
5. Зашифрованные пакеты вместе с заголовочной информацией пересылаются по сети.

Весь процесс показан на рис. 15.4.

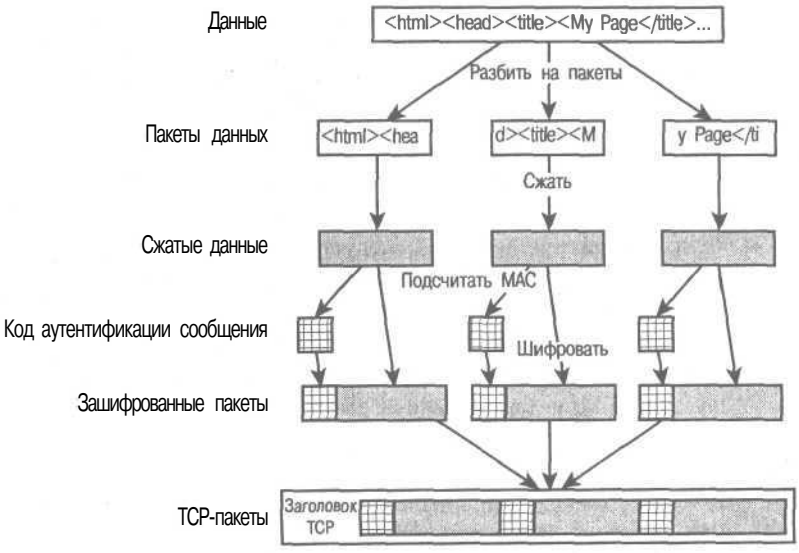


РИСУНОК 15.4 *SSL разбивает данные на пакеты, сжимает их, хэширует и шифрует до пересылки.*

Из диаграммы можно заметить, что заголовок TCP добавляется после кодирования данных. Это значит, что маршрутная информация может быть перехвачена. И хотя шпионы не смогут получить информацию, они смогут определить, кто ею обменивается.

Причина, по которой сжатие в SSL осуществляется до шифрования, состоит в том, что хотя большая часть трафика в сети сжимается до пересылки, зашифрованные данные сжимаются плохо.

Алгоритмы сжатия основаны на поиске повторяющихся последовательностей данных. Попытка их применения после того, как данные в результате шифрования были превращены в случайный набор битов, в большинстве случаев оказывается бесполезной. Было бы нежелательным, чтобы SSL, разработанный для повышения безопасности сети, приводил к существенному увеличению трафика.

Хотя SSL достаточно сложен, пользователи и разработчики могут не беспокоиться об этом, поскольку его внешние интерфейсы полностью повторяют существующие протоколы.

В относительно недалеком будущем SSL 3.0 будет, скорее всего, заменен на TLS 1.0 (Transport Layer Security, защита транспортного слоя), но на момент написания книги TLS являлся лишь проектом стандарта, не поддерживаемым пока еще ни одним сервером или браузером. TLS предназначен быть действительно открытым стандартом, а не стандартом, который был разработан одной организацией и открыт для остальных. Он основывается на SSL 3.0, однако содержит ряд улучшений.

Проверка данных, вводимых пользователем

Один из принципов создания надежного Web-приложения заключается в том, чтобы никогда не доверять пользовательскому вводу. До того как помещать их в файл или в базу данных либо передавать на выполнение системной команде, данные, поступившие от пользователя, следует обязательно проверить.

В нескольких местах в этой книги обсуждались методы проверки пользовательского ввода. Ниже приводится их краткий список.

- Функцию **addslashes()** следует использовать для фильтрации данных до их пересылки в базу данных. Она позволяет избежать символов, которые могут вызывать проблемы при сохранении в базе данных. Чтобы вернуть данные к исходному виду, можно воспользоваться функцией **stripslashes()**.
- Магические кавычки. Включить директивы **magic_quotes_gpc** и **magic_quotes_runtime** можно в файле **php.ini.** Они автоматически добавляют или убирают управляющие символы обратной косой черты, причем **magic_quotes_gpc** выполняет это для входных переменных методов **GET, POST** и cookie-наборов, а **magic_quote_runtime** — для данных, входящих или исходящих из базы данных.
- Функцию **escapeshellcmd()** следует использовать до передачи данных таким командам, как **system()** или **exec().** Функция помещает символы обратной косой черты перед каждым метасимволом, который может быть использован злонамеренным пользователем с целью запуска определенных системных команд.
- Для удаления из строки HTML- и PHP-дескрипторов можно воспользоваться функцией **strip_tags()**. Это не позволит создавать сценарии внутри данных, которые сервер передает для отображения браузеру.
- Функция **htmlspecialchars()** предназначена для преобразования некоторых символов в их HTML-эквиваленты. Например, символ < преобразуется в **<**. Таким образом любые дескрипторы сценария можно превратить в безопасные символы.

Обеспечение безопасного хранения данных

Три различных типа сохраняемых данных (HTML- или PHP-файлы, данные сценариев и данные MySQL) часто размещаются в разных областях одного и того же диска, однако на рис. 15.1 они показаны отдельно. Каждый тип требует различных предосторожностей, поэтому и обсуждаться они будут отдельно.

Наиболее опасным является выполняемое содержимое. На Web-сайте таковым обычно являются сценарии. Необходимо проявлять особую осторожность при установке прав доступа внутри Web-иерархии. Под последней здесь подразумевается дерево каталогов, начинающееся с **htdocs** на сервере Apache или **inetpub** на сервере IIS. Посторонние должны иметь возможность только читать сценарии, но ни в коем случае не записывать в них.

Такое же условие касается и каталогов внутри Web-иерархии. Только их владельцы должны обладать правами записи в них. Другие пользователи, включая и того, под чей учетной записью запускается Web-сервер, не должны иметь прав доступа по записи или создания новых файлов в каталогах, которые могут быть загружены с Web-сервера. Если разрешить запись файлов, станет возможным создание злонамеренных сценариев и запуск их из Web-сервера.

Если сценариям требуется записывать в файлы, для этой цели необходимо создать каталог за пределами Web-дерева. В особенности это касается сценариев по выгрузке файлов. Сценарии и записываемые ими данные не должны перемешиваться.

При записи важные данные можно зашифровать. Однако подобный подход редко дает хорошие результаты.

Предположим, что на Web-сервере имеется файл **creditcardnumbers.txt**, и взломщик получил доступ к серверу, что еще он может прочесть? Чтобы зашифровать и дешифровать данные, требуются соответствующие программы и один или несколько файлов ключей. Если взломщик может прочесть данные, скорее всего, ничто не помешает ему прочесть файлы ключей, равно как и другие файлы.

Шифрование данных на Web-сервере имеет смысл только в том случае, если программное обеспечение и ключи для дешифрации хранятся на другой машине. Один из способов обработки важных данных состоит в их шифровании на сервере с последующей пересылкой на другую машину, возможно, по электронной почте.

Содержимое базы данных похоже на данные, хранящиеся в обычных файлах данных. Если MySQL настроен корректно, то только эта система может записывать информацию в свои файлы. А это значит, что беспокоиться следует только о доступе пользователей в рамках MySQL. В книге уже обсуждались вопросы, связанные с системой управления правами доступа MySQL, которая присваивает определенные права определенным пользователям на указанных хостах.

Особого внимания требует следующий факт: часто в PHP-сценариях необходимо указать пароль для доступа в MySQL. PHP-сценарии зачастую являются общедоступно загружаемыми. Однако это не представляет такой проблемы, как может показаться сначала. Если конфигурация Web-сервера не нарушена, исходный код PHP никогда не будет виден снаружи.

Если Web-сервер сконфигурирован для обработки файлов с расширением **.php** при помощи интерпретатора PHP, внешние пользователи не имеют возможности увидеть исходный код. Нем не менее, при использовании расширений следует быть осторожным. Если файлы **.inc** разместить в Web-каталогах, по внешнему запросу можно будет получить исходный код. В этом случае необходимо поместить файлы за пределами Web-дерева и либо сконфигурировать сервер так, чтобы он не пересылал файлы с таким расширением, либо использовать для них только расширение **.php**.

Если Web-сервер вместе с вами используют и другие, то ваши пароли MySQL могут быть видны таким пользователям. В зависимости от того, как сконфигурирована система, упомянутая ситуация может быть неизбежной. Выходом может послужить или настройка Web-сервера таким образом, чтобы сценарии выполнялись отдельными пользователями, или запуск копии Web-сервера для каждого пользователя. Если вы не являетесь администратором Web-сервера (скорее всего, это так, раз вы делите его с другими), имеет смысл обсудить проблемы с администратором и изучить возможные способы защиты.

Зачем хранить номера кредитных карточек?

При обсуждении безопасного хранения важных данных есть один их тип, который заслуживает особого внимания. Пользователей Internet охватывает паранойя при мысли о номерах кредитных карточек. Поэтому, если вы собираетесь хранить их на своем сайте, следует проявить предельную осторожность. Кроме того, следует задаться вопросом, действительно ли это необходимо.

Что происходит с номером кредитной карточки? Если выполняется одноразовая транзакция и обработка карточки в реальном времени, лучше просто получить номер и направить его сразу в шлюз транзакций, не сохраняя на своем сервере.

Если необходимо производить периодические отчисления, например, взымать ежемесячные взносы с одной и той же кредитной карточки, такой подход не подойдет. В данной ситуации номера карточек следует хранить за пределами Web-сервера.

Если вы все же собираетесь хранить данные о кредитных карточках клиентов, вам потребуется профессиональный системный администратор, возможно, немного параноик в вопросах безопасности, который тратил бы достаточное время на изучение последней информации о защите операционной системы и используемых продуктов.

Использование шифрования в PHP

Простая, но полезная задача, демонстрирующая применение шифррования, заключается в отправке зашифрованной почты. Стандартом *де-факто* многие годы был PGP — Pretty Good Privacy (Хорошо обеспечиваемая приватность). Филипп Р. Циммерман (Philip R. Zimmermann) создал PGP специально для того, чтобы придать почте необходимую конфиденциальность.

Существуют бесплатные версии PGP, однако это программное обеспечение не является свободно распространяемым. Бесплатную версию можно применять только в некоммерческих целях.

Если вы являетесь гражданином США и находитесь в пределах США или гражданином Канады в пределах Канады, можете получить бесплатную версию с сайта

```
http://web.mit.edu/network/pgp.html
```

Для использования PGP в коммерческих целях в рамках США или Канады можно получить лицензию от Network Associates. Подробности указаны на сайте

```
http://www.pgp.com
```

Для использования PGP за пределами США и Канады следует обратиться к интернациональной странице PGP:

```
http://www.pgpi.org
```

Недавно появилась альтернатива PGP с открытым исходным кодом. GPG — Gnu Privacy Guard (Хранитель приватности GNU) — представляет собой распространяемую свободно замену PGP. Он не содержит запатентованных алгоритмов и поэтому может использоваться в коммерческих целях без каких-либо ограничений.

Оба этих продукта выполняют свою задачу похожими способами. На уровне командной строки разница практически незаметна, но PGP имеет и другие полезные интерфейсы, например, модули для популярных почтовых программ, которые автоматически дешифруют сообщения при получении.

GPG можно загрузить с сайта

```
http://www.gnupg.org
```

Оба продукта можно использовать вместе, создав, например, зашифрованное сообщение при помощи GPG и переслав его лицу, использующему PGP для дешифрации (если это последняя версия). Ниже приведен пример с использованием GPG для создания зашифрованных сообщений на Web-сервере. Применение в данном случае PGP практически не потребовало бы изменений.

Данный пример требует наличия работающей системы GPG. Возможно, GPG уже установлен в системе. Если это не так, не стоит беспокоиться: процедура инсталляции проста, хотя настройка и требует некоторых ухищрений.

Инсталляция GPG

Чтобы установить GPG на Linux машине, необходимо загрузить требуемый архивный файл из сайта **www.gnupg.org,** а затем воспользоваться утилитами **gunzip и tar.**

Для компиляции и установки программы используются те же команды, что и для большинства программ Linux:

```
configure (или ./configure в зависимости от настроек систем)
make
make install
```

Если инсталляция производится не пользователем root, следует запустить конфигурационный сценарий с опцией **--prefix**:

```
./configure --prefix=/путь/к/вашему/каталогу
```

Это требуется по той причине, что только пользователь root имеет права доступа к каталогу по умолчанию для GPG.

Если все прошло хорошо, то GPG скомпилирован и выполняемый файл скопирован в **/usr/local/bin/gpg** или указанный каталог. Многие опции можно изменить — подробности приведены в документации по GPG.

Для Windows-сервера процесс выглядит еще проще. Требуется загрузить zip-файл, разархивировать его и поместить файл **gpg.exe** в каталог, входящий в переменную PATH. (Вполне подойдет C:\Windows\ или что-то похожее.) Затем необходимо создать каталог C:\gnupg, запустить сеанс командной строки и ввести команду **gpg.**

Кроме того, следует установить GPG или PGP и сгенерировать пару ключей в системе, с которой будет приходить почта.

На Web-сервере практически нет различий между версиями командной строки GPG и PGP, поэтому здесь будет использоваться пакет GPG, поскольку он распространяется свободно. Однако на машине, с которой будет поступать почта, лучше отдать предпочтение коммерческой версии PGP, чтобы воспользоваться удобным графическим интерфейсом модуля, встраиваемого в систему чтения почты.

Если на машине чтения почты еще нет пары ключей, ее необходимо создать. Вспомните, что пара ключей состоит из Открытого ключа (Public Key), который другие пользователи (и PHP-сценарии) применяют для шифрования почты до отправки вам, и Закрытого ключа (Private Key), который используется с вашей стороны для дешифрации полученных сообщений или шифрования исходящих.

Важно, чтобы генерация ключа производилась на машине для чтения почты, а не на Web-сервере, так как закрытый ключ хранить на ней не следует.

Если для генерации ключей применяется версия командной строки GPG, следует ввести такую команду:

```
gpg --gen-key
```

Программа задаст несколько вопросов, причем на большинство из них можно дать ответ, предлагаемый по умолчанию. Программа запросит имя и почтовый адрес, которые будут использоваться в имени ключа. Ключ автора называется **'Luke Welling <luke@tangledweb.com.au>'**. Шаблон создания ключей очевиден.

Для экспорта общедоступного ключа из вновь созданной пары используется команда:

```
gpg --export > filename
```

В результате будет создан двоичный файл, подходящий для импорта GPG- или PGP-ключей на другую машину. Если ключ требуется переслать по почте, его можно сохранить в ASCII-формате командой:

```
gpg --export -a > filename
```

После извлечения открытого ключа файл можно загрузить на Web-сервер с использованием FTP.

Далее подразумевается, что на сервере установлен UNIX. Для Windows шаги будут такими же, однако имена каталогов и системных команд будут другими.

Необходимо зарегистрироваться под своей учетной записью на Web-сервере и изменить права доступа к файлу, чтобы его могли читать другие пользователи:

```
chmod 644 filename
```

Необходимо создать набор ключей, чтобы пользователь, под именем которого запускаются PHP-сценарии, мог задействовать GPG. Каким является этот пользователь, зависит от того, как настроен сервер. Зачастую это **'nobody'**, но здесь возможны и другие варианты.

Итак, необходимо зарегистрироваться в качестве пользователя Web-сервера. Для этого к серверу необходим доступ как root. На многих системах Web-сервер запускается пользователем **nobody.** Это предполагается и в последующих примерах. (Можно изменить имя так, как того требует ваша система.) Введите команды

```
su root
su nobody
```

Создайте каталог с владельцем **nobody** для сохранения в нем набора ключей и другой конфигурационной информации GPG. Каталог должен находиться в каталоге пользователя **nobody.**

Каталог каждого пользователя определяется в файле **/etc/passwd**. На многих Linux-системах каталогом **nobody** является /, причем **nobody** не обладает правами записи в него. На многих BSD-системах каталог **nobody** указан как **/nonexistent**. Поскольку это-

го каталога не существует, в него нельзя ничего записать. В нашем случае пользователю **nobody** присвоен каталог **/tmp**. Здесь потребуется убедиться, что пользователь Web-сервера имеет такой начальный каталог, куда он может записывать файлы.

Введите

```
cd ~
mkdir .gnupg
```

Пользователю **nobody** требуется собственный ключ для шифрования. Для его создания следует запустить команду:

```
gpg --gen-key
```

Поскольку пользователь **nobody** практически не получает почты, для него можно создать только ключ для исходящей почты. Его единственная цель состоит в том, чтобы сделать извлеченный ранее открытый ключ вполне достойным доверия.

Для импорта этого открытого ключа применяется команда:

```
gpg --import filename
```

Чтобы сообщить GPG, что этому ключу можно доверять, необходимо изменить свойства ключа

```
gpg --edit-key 'Luke Welling <luke@tangledweb.com.au>'
```

Текст в кавычках представляет собой имя ключа. Очевидно, что именем вашего ключа будет не **'Luke Welling <luke@tangledweb.com.au>'**, а комбинация имени, комментария и почтового адреса, заданных при его генерации.

Среди опций этой программы есть и **help**, которая выводит описание доступных команд — **trust (доверять), sign (кодировать) и save (сохранить).**

Команда **trust** указывает GPG, что ключ достоин полного доверия, **sign** применяется для кодирования открытого ключа с использованием закрытого ключа пользователя **nobody,** save — для выхода из программы с сохранением всех изменений.

Тестирование GPG

После всех этих операций пакет GPG настроен и готов к использованию.

Для его тестирования потребуется создать файл **test.txt,** содержащий некоторый текст.

Запуск команды

```
дрд -а --recipient 'Luke Welling <luke@tangledweb.com.au>' --encrypt test.txt
```

(где задано имя вашего ключа) должен выдать предупреждение

```
gpg: Warning: using insecure memory!
```

и создать файл с именем **test.txt.asc.** Он содержит зашифрованное сообщение, которое выглядит примерно так:

```
----- BEGIN PGP MESSAGE -----
Version: GnuPG v1.0.3 (GNU/Linux)
Comment: For info see http://www.gnupg.org

hQEOA0DU7hVGgdtnEAQAhr4HgR7xpIBsK9CiELQw85+k1QdQ+p/FzqL8tICrQ+B3
0GJTEehPUDErwqUw/uQLTds0r1oPSrIAZ7c6GVkh0YEVBj2MskT81IIBvdo95OyH
K9PUCvg/rLxJ1kxe4Vp8QFET5E3FdII/1y8VP5gSTE7gAgm0SbFf3S91PqwMyTkD
/2oJEvL6e3cP384s0i81rBbDbOUAAhCjjXt2DX/uX9q6P18QW56UICUOn4DPaW1G
```

```
/gnNZCkcVDgLcKfBjbkB/TCWWhpA7o7kX4CIcIh7KlIMHY4RKdnCWQf271oE+8i9
cJRSCMsFIoI6MMNRCQHY6p9bfxL2uE39IRJrQbe6xoEe0nkB0uTYxiL0TG+FrNrE
tvBVMS0nsHu7HJey+oY4Z833pk5+MeVwYumJwlvHjdZxZmV6wz46GO2XGT17b28V
wSBnWOoBHSZsPvkQXHTOq65EixP8y+YJvBN3z4pzdH0Xa+NpqbH7q3+xXmd30hDR
+u7t6MxTLDbgC+NR
=gfQu
-----END PGP MESSAGE-----
```

При переносе этого файла на систему, где был создан ключ, и запуске команды:

```
gpg -d test.txt.asc
```

можно просмотреть исходный текст сообщения.

Чтобы поместить текст в файл, а не выводить на экран, следует воспользоваться флажком -o:

```
gpg -do test.out test.txt.asc
```

Если GPG настроен так, что пользователь, от имени которого запускаются PHP-сценарии, может запустить его из командной строки, то практически все завершено. Если что-то не работает, проконсультируйтесь с системным администратором либо обратитесь к документации по GPG.

В листингах 15.1 и 15.2 показано, как можно пересылать зашифрованные почтовые сообщения, используя PHP для вызова GPG.

Листинг 15.1 private_mail.php — **HTML-форма для отправки зашифрованных почтовых сообщений**

```
<html>
<body>
<h1>Send Me Private Mail</h1>

<?
 // Эту строку необходимо изменить, если не используются порты
 // по умолчанию (порт 80 для обычного трафика и порт 443 для SSL)
 if($HTTP_SERVER_VARS["SERVER_PORT"]!=443)
   echo "<p><font color = red>
              WARNING: you have not connected to this page using SSL.
              Your message could be read by others.</font></p>";
?>

<form method = post action = send_private_mail.php><br>
Your email address:<br>
<input type = text name = from size = 38Xbr>
Subject:<br>
<input type = text name = title size = 38xbr>
Your message:<br>
<textarea name = body cols = 30 rows = 10>
</textareaXbr>
<input type = submit value = "Send!">
</form>
</body>
</html>
```

Листинг 15.2 send_private_mail.php — PHP-сценарий для вызова GPG и отправки зашифрованной почты

```
<?
   $to_email = "luke@localhost";

   // Указать gpg, где находится набор ключей
   // В данной системе он содержится в каталоге /tmp/ пользователя nobody
   putenv("GNUPGHOME=/tmp/.gnupg");

   // Создать уникальное имя файла
   $infile = tempnam("", "pgp");
   $outfile = $infile.".asc";

   // Записать в файл текст, введенный пользователем
   $fp = fopen($infile, "w");
   fwrite($fp, $body);
   fclose($fp);

   // Настроить параметры команды
   $command =  "/usr/local/bin/gpg -a \\
               --recipient 'Luke Welling <luke@tangledweb.com.au>' \\
               --encrypt -o $outfile $infile";

   // Запустить команду gpg
   system($command, $result);

   // Удалить незашифрованный временный файл
   unlink($infile);

if ($result==0)
   {
     $fp = fopen($outfile, "r");
     if (!$fp||filesize ($outfile)==0)
     {
       $result = -1;
     }
     else
     {
       // Прочесть зашифрованный файл
       $contents = fread ($fp, filesize ($outfile));
       // Удалить временный зашифрованный файл
       unlink($outfile);

       mail($to_email, $title, $contents, "From: $from\n");
       echo "<h1>Message Sent</hl>
             <p>Your message was encrypted and sent.
             <p>Thankyou.";
     }
   }

   if ($result!=0)
   {
     echo "<h1>Error:</h1>
           <p>Your message could not be encrypted, so has not been
   sent.
           <p>Sorry.";
   }
?>
```

Чтобы этот код работал в вашей ситуации, необходимо внести некоторые изменения. Сообщение электронной почты отправляется по адресу **$to_email**.

Строку

```
putenv("GNUPGHOME=/tmp/.gnupg");
```

следует изменить так, чтобы она отражала местонахождение набора ключей GPG. В примере это каталог пользователя **nobody** **/tmp/**.

Функция **tempnam()** используется для создания уникального имени временного файла. Можно указать и каталог, и префикс имени файла. Поскольку подобные файлы создаются и удаляются в течение секунды, как они называются, не имеет большого значения. Здесь указан префикс 'pgp', а каталогом является временный системный каталог, используемый PHP по умолчанию.

Оператор

```
$command = "/usr/local/bin/gpg -a ".
           "--recipient 'Luke Welling <luke@tangledweb.com.au>' ".
           "--encrypt -o $outfile $infile";
```

настраивает команду и ее параметры для вызова gpg. Его можно изменить в своих целях. Как и при использовании командной строки, GPG необходимо указать, какой ключ применять для шифрования сообщений.

Оператор

```
system($command, $result);
```

запускает на выполнение строку **$command** и присваивает возвращаемое значение переменной **$result**.

Возвращаемое значение, в принципе, можно опустить, но оно позволяет проверить и сообщить пользователю, если операция не завершилась успешно.

Когда временные файлы становятся ненужными, они удаляются при помощи функции **unlink()**. Это значит, что пользовательская почта в незашифрованном виде сохраняется на сервере на короткий промежуток времени. Более того, если Web-сервер выйдет из строя во время исполнения, файл останется на сервере.

При обсуждении вопросов безопасности сценариев важно принять во внимание все информационные потоки внутри системы. GPG позволяет отправителю шифровать почту, а получателю — дешифровать ее, но как именно информация поступает от отправителя к получателю? Если для отправки зашифрованной при помощи GPG почты используется Web-интерфейс, поток информации будет выглядеть примерно так, как показано на рис. 15.5.

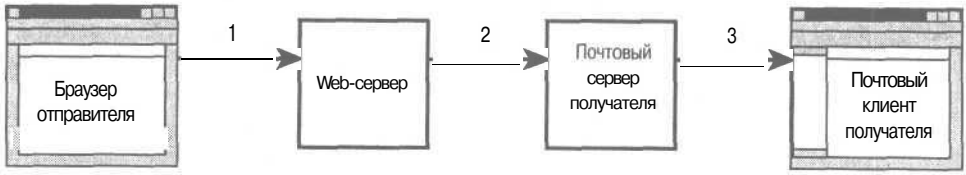


РИСУНОК 15.5 *В приложении, шифрующем почту, сообщение пересылается через Internet трижды.*

На этом рисунке каждая стрелка представляет пересылку сообщения с одной машины на другую. При каждой пересылке сообщение путешествует через Internet, проходя по пути промежуточные сети и машины.

Рассматриваемый здесь сценарий хранится на машине, обозначенной на диаграмме как Web-сервер. На ней сообщение шифруется с использованием открытого ключа получателя. После этого через SMTP-протокол сообщение пересылается на почтовый сервер получателя. Получатель соединяется со своим почтовым сервером по протоколу POP или ШАР и загружает сообщение с помощью почтового клиента. Затем он дешифрует сообщение, используя свой закрытый ключ.

Пересылки данных на рис. 15.5 обозначены цифрами 1, 2 и 3. На стадиях 2 и 3 информация пересылается в форме зашифрованного при помощи GPG сообщения и недоступна тем, кто не имеет закрытого ключа. Однако при пересылке 1 сообщение имеет обычный текстовый вид, в котором отправитель вводил его в HTML-форму.

Если информация настоль важна, что шифруется на второй и третьей стадии, нелогично пересылать ее в обычном формате на первой стадии. Именно поэтому сценарий размещается на сервере, поддерживающем SSL.

Если соединение со сценарием происходит не через порт 443, выдается предупреждение. Этот порт используется для SSL по умолчанию. Если ваш сервер использует другой порт для SSL, потребуется изменить код сценария.

Вместо того чтобы выдавать сообщение об ошибке, можно обработать эту ситуацию по-другому. Можно просто перенаправить пользователя на тот же URL, но через соединение SSL. Кроме того, ошибку можно просто игнорировать, поскольку зачастую не важно, была ли форма получена по безопасному соединению. Важно только, чтобы информация, введенная пользователем в форму, пересылалась по безопасному соединению. Поэтому можно просто задать полный URL как параметр **action** для формы.

Открывающий дескриптор формы выглядит так:

```
<form method = post action = send_private_mail.php>
```

Его можно изменить так, чтобы данные пересылались серверу через SSL-соединение, даже если пользователь подключился без SSL:

```
<form method = post action = "https://webserver/send_private_mail.php">
```

Если в коде задан такой URL, можно быть уверенным, что пользовательские данные будут пересылаться через SSL, однако в этом случае код требует модификации при использовании на другом сервере или даже просто в другом каталоге.

Хотя и не столь важно, чтобы пустая форма пересылалась пользователям через SSL, обычно лучше сделать именно так. Если пользователь будет видеть символ замка в углу окна своего браузера, он будет уверен, что передавать информацию безопасно. Далеко не все пользователи будут просматривать исходный HTML-код, чтобы узнать какие именно действия определены в атрибутах формы.

Дополнительная информация

Спецификации SSL версии 3.0 находятся на сайте Netscape:

```
http://home.netscape.com/eng/ssl3/
```

Дополнительные сведения о сетях и сетевых протоколах можно найти в специальной литературе.

Что дальше

На этом обсуждение вопросов электронной коммерции и безопасности завершается. В следующей части рассматриваются дополнительные возможности PHP, включающие в себя взаимодействие с другими машинами в Internet, создание изображений на лету и управление сеансами.

Часть 4

Усложненные технологии применения PHP

В этой части:

16 Взаимолействие с файловой системой и сервером
17 Использование функций работы с сетью и протоколами
18 Управление датой и временем
19 Создание изображений
20 Управление сеансами в PHP
21 Аругие полезные свойства PHP

Глава 16

Взаимодействие с файловой системой и сервером

В главе 2 мы рассмотрели запись данных в файлы Web-сервера и считывание их оттуда. Теперь самое время заняться другими функциями PHP, предназначенными для взаимодействия с файловой системой Web-сервера.

Мы рассмотрим:

- Загрузку файлов на сервер
- Функции работы с каталогами
- Выполнение операций с файлами на сервере
- Запуск программ на сервере
- Использование переменных среды сервера

Прежде чем приступить к изучению этих функций, стоит обратиться к примеру.

Предположим, что необходимо предоставить клиенту возможность вносить изменения в какую-то часть содержимого Web-сайта (например, в раздел новостей компании) либо нам самим понадобился более удобный FTP-интерфейс. Одно из возможных решений в подобных ситуациях — предоставление возможности пользователю загружать на сервер простые текстовые файлы. Далее эти файлы включаются в содержимое сайта при помощи шаблона, сконструированного в PHP — именно так мы поступали в главе 6.

Прежде чем уйти с головой в изучение функций файловой системы, рассмотрим кратко процесс загрузки файлов.

Основы загрузки файлов на сервер

Поддержка загрузки файлов на сервер по протоколу HTTP — одна из важнейших функциональных возможностей PHP. В этом процессе пересылка файлов происходит не

в обычном для HTTP-протокола направлении — с сервера на браузер, а в противоположном — с браузера на сервер. Обычно для передачи данных в этом направлении применяются HTML-формы. Форма, применяемая в примере, показана на рис. 16.1.



РИСУНОК 16.1
Типы полей HTML-формы, используемой для загрузки файлов на сервер, отличаются от типов полей обычных форм

Несложно заметить, что форма содержит поле для ввода имени файла. Предусмотрена также кнопка Browse (Просмотр) для поиска требуемых файлов в локальной файловой системе. Вполне вероятно, что читателю не доводилось ранее работать с формами для загрузки файлов на сервер. Мы рассмотрим реализацию такой формы ниже.

После ввода имени файла пользователь пересылает его на сервер по щелчку на кнопке Send File (Переслать файл). Файл будет перенаправлен на сервер, где его уже ожидает PHP-сценарий.

HTML-код загрузки файлов на сервер

Для реализации пересылки файлов на сервер применяются специально для этого предназначенные HTML-конструкции. HTML-код формы, показанной на рис. 16.1, приводится в листинге 16.1.

Листинг 16.1 upload.html — HTML-форма для загрузки файлов на сервер

```
<html>
<head>
   <title>Administration — upload new files</title>
</head>
<body>
<h1>Upload new news files</h1>
<form enctype="multipart/form-data" action="upload.php" method=post>
   <input type="hidden" name="MAX_FILE_SIZE" value="1000">
   Upload this file: <input name="userfile" type="file">
   <input type="submit" value="Send File">
</form>
</body>
</html>
```

Обратите внимание на то, что указан метод загрузки POST. Применим также метод PUT, поддержка которого реализована в NetScape Composer и Amaya. Поддержка метода GET в упомянутых программах отсутствует.

Ниже перечислены особенности этой формы.

- В дескрипторе <form> необходимо установить атрибут **enctype="multipart/form-data",** по которому сервер сможет определить, что файл передается с обычными данными формы.

- Форма должна содержать поле с указанием максимально допустимого объема загружаемого файла. Это скрытое поле представлено в данной форме в виде:

```
<input type="hidden" name="MAX_FILE_SIZE" value="1000">
```

  Указанному полю необходимо присвоить имя **MAX_FILE_SIZE** и значение, равное максимально допустимому объему пересылаемого файла (в байт).

- Еще одно скрытое поле применяется для указания файла в качестве типа передаваемых данных:

```
<input name="userfile" type="file">
```

  Значение атрибута **name** (имя файла) может быть любым, но при этом необходимо иметь в виду, что это же имя придется указывать при обращении к файлу из принимающего его PHP-сценария.

Реализация *PHP-кода для работы с* файлом

Написание PHP-процедуры приема файла не составляет особого труда.

При загрузке на сервер файл временно помещается в некоторый каталог, определенный на Web-сервере для этой цели. Если файл не переместить или не переименовать, прежде чем сценарий завершит работу, он будет уничтожен.

Поскольку HTML-форма содержит поле с именем **userfile**, PHP передаются четыре переменные:

- Переменная **$userfile** содержит временное местоположение файла на Web-сервере.
- Переменная **$userfile_name** содержит имя файла в системе пользователя.
- Переменная **$userfile_size** содержит размер файла в байт.
- Переменная **$userfile_type** содержит MIME-тип файла — например, **text/plain** (текст/простой) или **image/gif** (изображение/gif).

Обращение к этим переменным возможно также через массив **$HTTP_POST_FILES**:

- $HTTP_POST_FILES['userfile']['tmp_name']
- $HTTP_POST_FILES['userfile']['name']
- $HTTP_POST_FILES['userfile']['size']
- $HTTP_POST_FILES['userfile']['type']

Если известно расположение и имя файла, его можно скопировать в какой-то другой каталог. Это необходимо сделать до завершения сценария загрузки, которое приведет к удалению файла из временного каталога — в противном случае загрузка окажется бессмысленной.

В нашем примере загружаемый файл должен содержать статьи новостей, поэтому следует удалить все дескрипторы, которые в нем могут быть, и сохранить их в некотором каталоге. Сценарий этой процедуры приведен в листинге 16.2.

Листинг 16.2 upload.php — PHP-сценарий приема файла, пересылаемого при помощи HTML-формы

```
<head>
   <title>Uploading...</title>
</head>
<body>
<h1>Uploading file...</h1>
```

```
<?
   if  ($userfile=="none")
   {
     echo  "Problem:  no  file  uploaded";
     exit;
   }
   if   ($userfile_size==0)
   {
     echo  "Problem:  uploaded  file  is  zero  length";
     exit;
   }
   if  ($userfile_type  !=  "text/plain")
   {
     echo "Problem:  file is not plain text" ;
     exit;
   }
   if(!is_uploaded_file($userfile))
   {
     echo  "Problem:  possible  file upload attack";
     exit;
   }
   $upfile=" /home/book/uploads/" . $userfile_name;
   if ( ! copy ($userfile, $upfile) )
   <
     echo  "Problem:  Could not move  file  into  directory";
     exit;
   }

   echo  "File uploaded successfully<br><br>" ;
   $fp = f open ($upfile, "r") ;
   $contents = fread ($fp, filesize ($upfile)) ;
   f close ($fp) ;
   $contents = strip_tags ($contents) ;
   $fp = f open ($upf ile, "w") ;
   fwrite($fp,  $contents) ;
   fclose($fp);
   echo  "Preview of uploaded file contents: <br><hr>" ;
   echo  $contents;
   echo  "<br><hr>";
?>
</body>
</html>
<?
   // Эта функция взята из руководства по PHP.
   // is_uploaded_file встроена в PHP4.0.3.
   // Для предыдущих версий можно воспользоваться следующим кодом.
   function is_uploaded_file($filename)  {
     if (!$tmp_file= get_cfg_var ( 'upload_tmp_dir' ) ) {
          $tmp_file=dirname (tempnam(' ' ,' ' ) ) ;
     }
     $tmp_file .= '/' • basename ($filename) ;
     /* Пользователь может иметь конечную косую черту в php.ini... */
     return  (ereg_replace('/+', '/', $tmp_file)  == $filename) ;
   }
?>
```

Обратите внимание, что основную часть объема этого сценария составляют операторы проверки ошибок. Зафузка файлов на сервер чревата нарушениями безопасности, которые следует, по возможности, предотвращать. Важно также тщательно проверять загружаемый файл на предмет допустимости открытой публикации содержащейся в нем информации.

Рассмотрим основные операции сценария.

Первым делом, выполняется проверка значения переменной **$userfile**. Если оно оказывается равным **"none",** загрузка файла не происходит. Далее проверяется наличие у файла содержимого (файл предполагается непустым, если значение переменной **$userfile_size** больше нуля) и соответствие типа этого содержимого требуемому (по значению переменной **$userfile_type**).

Следующая проверка — пытаемся ли мы открыть загруженный файл, или же это локальный файл наподобие **/etc/passwd**. К этому мы еще вернемся.

Если все проверки дают положительный результат, файл копируется в каталог включаемых файлов. В данном примере это каталог **/home/book/uploads/**. Он находится вне дерева Web-документов и потому представляет собой прекрасное место для размещения файлов, включаемых в какой-либо проект.

После этого файл открывается, очищается при помощи функции **strip_tags()** от ненужных HTML- или PHP-дескрипторов и записывается на прежнее место.

Наконец, содержимое файла выводится на экран, чтобы пользователь мог убедиться в успешном завершении загрузки.

Результаты одного (успешного) запуска этого сценария показаны на рис. 16.2.

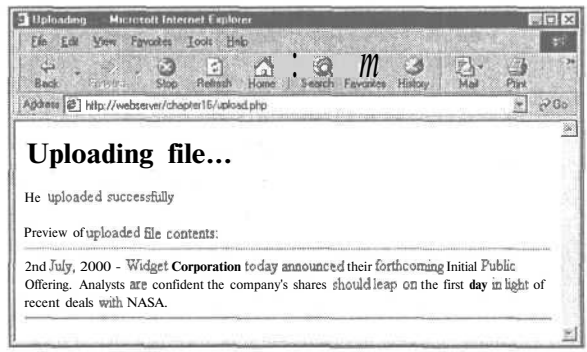


РИСУНОК 16.2
После копирования и переформатирования файла, его содержимое выводится на экран для подтверждения успешного завершения загрузки

В сентябре 2000 г. появилось сообщение о разработке, при помощи которой взломщики могут переключать сценарий загрузки на обработку локального файла вместо загруженного. Разработка задокументирована в списке рассылки BUGTRAQ. Официальные рекомендации по обеспечению безопасности опубликованы во множестве архивов BUGTRAQ, в частности, на странице

```
http://lists.insecure.org/bugtraq/2000/Sep/0237.html
```

С целью проверки успешного завершения загрузки файла и гарантии того, что мы имеем дело не с локальным файлом наподобие **/etc/passwd,** была применена функция **is_uploaded_file()**. Эта функция появилась в PHP 4.0.3. Во время написания книги текущей была версия 4.0.2, поэтому примеры применения этой функции взяты из руководства по PHP.

Небрежно написанный сценарий загрузки может позволить посетителю с недобрыми намерениями создать временный файл с определенным именем и заставить сценарий обрабатывать его как загруженный. А поскольку многие сценарии загрузки возвращают пользователю копию загруженного файла либо размещают в месте, из которого ее можно загрузить, у пользователей появляется возможность доступа к любому файлу, доступному для считывания Web-сервером. Среди этих файлов могут оказаться файлы

с конфиденциальной информацией — например, **/etc/passwd** или файл исходного PHP-кода, содержащий пароли доступа в базу данных.

Часто возникающие затруднения

При загрузке файлов на сервер следует принимать во внимание несколько важных моментов.

- В предыдущем примере предполагается наличие проверки прав доступа пользователей. Нельзя разрешать загрузку файлов на сервер кому угодно.
- Если разрешить загрузку файлов на сервер без проверки прав доступа либо пользователям, не заслуживающим доверия, возникают очень веские основания для беспокойства за содержимое этих файлов. Загрузка и запуск сценария, способного нанести вред — события крайне нежелательные. Поэтому следует проверять не только тип и содержимое файла, как в приведенном примере, но также и его имя. Очень неплохая мысль — присваивать загруженным файлам "безопасные" имена.
- В системе NT или прочих версиях Windows не забывайте указывать в строках пути обратную косую черту вместо обычной.
- Если возникают затруднения с запуском этой схемы, проверьте файл **php.ini**. Он должен содержать директиву **upload_tmp_dir**, задающую каталог, к которому необходимо получить доступ. Для загрузки файлов больших объемов может понадобиться также настройка директивы **memory_limit**, которая определяет максимально допустимые размеры загружаемых файлов.
- Если PHP работает в безопасном режиме, выводится сообщение о невозможности доступа к временному файлу. Это можно исправить только запуском в обычном режиме либо написанием сценария для копирования файла в доступный каталог, не используя синтаксис PHP. Запускать его можно из PHP-сценария. Ближе к концу главы мы рассмотрим способы запуска программ на сервере из PHP.

Использование функций работы с каталогами

После загрузки нескольких файлов пользователям может потребоваться просмотреть их и выполнить некоторые манипуляции с содержимым.

Для этой цели в PHP предусмотрен набор функций для работы с файлами и каталогами.

Считывание из каталогов

Прежде всего, составим сценарий просмотра каталогов для проверки загруженных файлов. Программирование просмотра каталогов в PHP совершенно несложно. В листинге 16.3 приведен пример соответствующего сценария.

Листинг 16.3 browsedir.php - просмотр каталогов для проверки результатов загрузки файлов

```
<html>
<head>
  <title>Browse  Directories</title>
</head>
<body>
<h1>Browsing</h1>
```

```
<?
   $current_dir = "/home/book/uploads/";
   $dir = opendir($current_dir);
   echo "Upload directory is $current_dir<br>";
   echo "Directory Listing:<br><hr><br>";
   while ($file = readdir($dir))
   {
        echo "$file<br>";
   }
   echo "<hr><br>";
   closedir($dir);
?>
</body>
</html>
```

В сценарии задействованы функции **opendir()**, closedir() и **readdir()**.

Функция **opendir()** открывает каталог для чтения. Она действует подобно обычной файловой функции **fopen(),** однако с аргументом, содержащим имя каталога, а не файла:

```
$dir = opendir($current_dir);
```

Функция возвращает дескриптор каталога — опять же, подобно функции **fopen(),** возвращающей дескриптор файла.

После того как каталог открыт, из него можно читать имена файлов при помощи функции **readdir($dir)**, как показано в примере. После считывания всех файлов функция возвращает значение false. Следует иметь в виду, что она возвращает **false** также в случае считывания файла с именем "0", и если этот вариант возможен, следует предусмотреть соответствующую проверку. Файлы не сортируются каким-либо образом, поэтому для сортировки их потребуется сначала сохранить в массиве, а затем этот массив отсортировать.

По завершении чтения каталог следует закрыть, воспользовавшись функцией **closedir($dir)** — опять же, аналогичной файловой функции **fclose().**

Пример вывода результатов просмотра каталога показан на рис. 16.3.

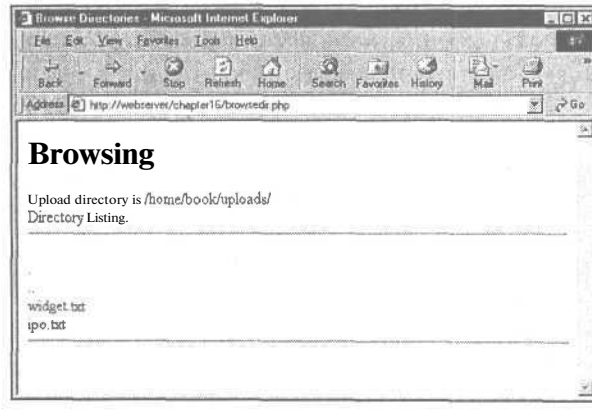


РИСУНОК 16.3
В листинге каталога представлено все его содержимое, в том числе . (текущий каталог) и .. (каталог уровнем выше). Можно определить вывод без этих элементов

Организуя просмотр таким способом, разумно ограничить количество просматриваемых (доступных) каталогов.

Еще одна полезная функция из этого же ряда — **rewinddir($dir)**. Она возвращает указатель считываемого файла на начало каталога.

Альтернативой перечисленным функциям может служить класс **dir**, определенный в PHP. Его свойства **handle и path,** а также методы **read(), close() и rewind()**, аналогичны соответствующим "неклассовым" переменным и функциям.

Получение сведений о текущем каталоге

PHP располагает средствами получения дополнительной информации относительно пути к файлу.

Функции **dirname($path) и basename($path)** возвращают составляющие пути, содержащие, соответственно, имена каталога и файла. Они могут оказаться полезными для нашего сценария просмотра каталогов — в особенности, если предстоит создать сложную систему с осмысленными именами каталогов и файлов.

В листинг содержимого каталога можно также включить строку с указанием свободного дискового объема, отведенного под загружаемые файлы. Для этого следует воспользоваться функцией **diskfreespace($path).** Если передать в эту функцию в качестве аргумента путь к каталогу, она вернет объем свободного пространства на диске в байтах (Windows) либо в файловой системе (UNIX), в которой расположен каталог.

Создание и удаление каталогов

PHP предоставляет средства не только для пассивного считывания информации. Функции **mkdir() и rmdir()** предназначены, соответственно, для создания и удаления каталогов. Конечно, эти операции выполнимы только в каталогах, к которым имеется доступ.

Применение функции **mkdir()** не так просто, как может показаться на первый взгляд. Функции необходимо указать два аргумента — путь к создаваемому каталогу (включая его имя) и разрешения доступа к нему, например:

```
mkdir("/tmp/testing", 0777);
```

Однако указанные разрешения могут не совпасть с заданными в действительности. Последние получаются в результате выполнения логической операции AND (И) - подобие вычитания — над разрешениями, указанными в качестве аргумента, и маской **umask.** Например, если **umask = 022,** получим разрешения, равные 0755.

Иногда бывает полезно до создания каталога устранить этот эффект, установив нулевое значение umask:

```
$oldumask = umask(0);
mkdir("/tmp/testing", 0777);
umask($oldumask);
```

В приведенном примере использована функция **umask()**, выполняющая проверку и модификацию значения **umask.** Она изменяет значение **umask** на заданное, возвращая прежнее значение; та же функция, вызванная без аргументов просто возвращает текущее значение **umask.**

Функция **rmdir()** удаляет каталог:

```
rmdir("/tmp/testing");
```

или

```
rmdir("c:\\tmp\\testing");
```

Удаляемый каталог должен быть пустым.

Взаимодействие с файловой системой

Естественно, PHP обеспечивает возможность получения информации не только о каталогах, но и о файлах Web-сервера, а также взаимодействие с ними. Мы уже рассматривали запись данных в файлы и считывание их оттуда, но это далеко не все файловые функции.

Считывание информации из файла

Изменим наш сценарий просмотра каталогов, добавив в него чтение файлов:

```
while ($file = $dir->read())
{
 echo "<a href=\ "filedetails.php?file=".$file."\ ">".$file."</a><br>";
}
```

Теперь составим сценарий **filedetails.php** (листинг 16.4) для считывания дополнительной информации о файле.

Этот сценарий страдает одним существенным недостатком: некоторые из используемых в нем функций, в том числе **fileowner() и filegroup()**, в Windows отсутствуют или выполняются ненадежно.

Листинг 16.4 filedetails.php — функции получения состояния файлов и результаты их выполнения

```
<html>
<head>
   <title>File Details</title>
</head>
<body>
<?
   $current_dir = "/home/book/uploads/";
   $file = basename($file);   // удалить информацию о каталоге в целях
   безопасности
   echo "<h1>Details of file: ".$file."</h1>";
   $file = $current_dir.$file;
   echo "<h2>File data</h2>";
   echo "File last accessed: ".date("j F Y H:i", fileatime($file))."<br>";
   echo "File last modified: ".date("j F Y H:i", filemtime($file))."<br>";
   $user = posix_getpwuid(fileowner($file));
   echo "File owner: ".$user["name"]."<br>";
   $group = posix_getgrgid(filegroup($file));
   echo "File group: ".$group["name"]."<br>";
   echo "File permissions: ".decoct(fileperms($file))."<br>";
   echo "File type: ".filetype($file)."<br>";
   echo "File size: " .filesize($file)." bytes<br>" ;
   echo "<h2>File tests</h2>";
   echo "is_dir: ".(is_dir($file)? "true" : "false")."<br>";
   echo "is_executable: ".(is_executable($file)? "true" : "false")."<br>";
   echo "is_file: ".(is_file($file)? "true" : "false")."<br>";
   echo "is_link: " . (is_link($file)? "true" : "false")."<br>";
   echo "is_readable: ".(is_readable($file)? "true" : "false")."<br>";
   echo "is_writable: ". (is_writable($file)? "true" : "false")."<br>";
?>
</body>
</html>
```

Результаты тестового запуска сценария из листинга 16.4 показаны на рис. 16.4.

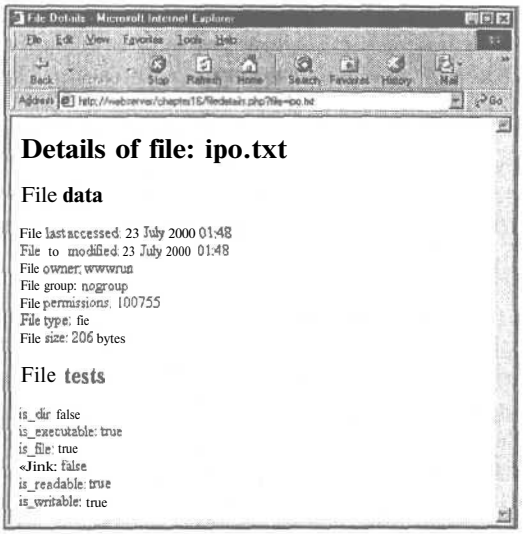


РИСУНОК 16.4
Подробное описание файла. Обратите внимание, что разрешения даны в шестнадцатеричном формате.

Теперь поговорим о том, как действует каждая из функций, используемых в сценарии листинга 16.4.

Как уже упоминалось, функция **basename()** возвращает имя файла без указания каталога, в котором он расположен. (Чтобы получить имя каталога без имени файла, воспользуйтесь функцией **dirname()**.)

Функции **fileatime()** и **filemtime()** возвращают метки времени, соответственно, последнего обращения и последней модификации файла. Мы переформатировали эти метки, воспользовавшись функцией **date()**, чтобы представить их в более наглядном виде. В некоторых операционных системах эти функции возвращают одинаковую информацию (что имеет место и в приведенном примере) — это зависит от того, как и какая именно информация хранится в самой системе.

Функции **fileowner() и filegroup()** возвращают идентификаторы, соответственно, пользователя **(uid)** и группы **(gid)**, которым предоставлен доступ к файлу. Для большей наглядности идентификаторы можно преобразовать в имена, воспользовавшись функциями, соответственно, **posix_getpwuid() и posix_getgrgid().** В аргументах этих функций указываются идентификаторы **uid** или gid, а возвращаемая ими информация представляет собой массив данных о пользователе или группе, включая имя; все эти особенности использованы в сценарии.

Функция **fileperms()** возвращает разрешения доступа к файлу. У нас они переформатированы при помощи функции **decoct()** в шестнадцатеричное представление, более привычное для пользователей UNIX.

Функция **filetype()** возвращает некоторые сведения о типе файла. Возвращаемые значения выглядят так: fifo, char, dir, block, link, file и unknown.

Функция **filesizeQ** возвращает размер файла в байтах.

Следующий набор функций **is_dir(), is_executable(), is_file(), is_link(), is_readable() и is_writable()**. Каждая из них проверяет определенный атрибут файла, возвращая в результате **true** или **false.**

Вместо большого числа перечисленных функций можно использовать единственную **stat(),** возвращающую массив информации, который содержит значения, возвращаемые отдельными функциями. Аналогичная ей функция **lstat()** применяется для символических ссылок.

Все функции определения состояния файлов довольно-таки затратны в смысле времени выполнения, поэтому возвращаемые ими результаты кешируются. Если понадобится сверить некоторые сведения до и после модификации файла, обратитесь к функции

```
clearstatcache();
```

чтобы уничтожить предыдущие результаты. Если бы наш сценарий был предназначен для подобной проверки, он должен был бы начинаться с вызова именно этой функции, чтобы гарантировать получение неустаревших результатов.

Изменение свойств файла

Имеется возможность не только просматривать свойства файла, но и вносить в них изменения.

Функции **chgrp(file, group), chmod(file, permissions) и chown(file, user)** аналогичны функциям с теми же именами из UNIX. Ни одна из ни не работает в Windows-системах. Правда, **chown()** выполняется, но в любом случае возвращает **true.**

Функция **chgrp()** применяется для смены группы, которой предоставлен доступ к файлу. Фактически, функция меняет группу (или группы), к которым принадлежит указанный пользователь, причем не обязательно root.

Функция **chmod()** применяется для изменения разрешений доступа к файлу. Разрешения указываются в обычном формате **chmod** системы UNIX — их необходимо указывать с префиксом "О", обозначающим восьмеричный формат, например:

```
chmod("somefile.txt", 0777);
```

Функция **chown()** применяется для смены владельца файла. Она работает только в сценарии, запущенном пользователем root, что не предполагается.

Создание, перемещение и удаление файлов

Файловая система содержит также функции создания, перемещения и удаления файлов.

Первая и самая простая функция — **touch()**. С ее помощью можно создать файл и поменять его имя или время последней модификации. Ее действие аналогично действию команды **touch** из UNIX. Прототип функции таков:

```
int touch (string file, [int time])
```

Если файл с указанным именем уже существует, время его модификации будет изменено на текущее время либо на время, указанное во втором аргументе (если он не опущен). Обратите внимание, что время должно быть указано в формате метки времени файла. Если файл не существует, он создается.

Функция **unlink()** предназначена для удаления файлов (обратите внимание, что ее имя — не **delete).** Синтаксис вызова функции выглядит следующим образом:

```
unlink($filename);
```

Это одна из функций, не работающих под управлением Win32. В Windows для удаления файлов следует применять функцию

```
system("del filename.ext");
```

Копирование и перемещение файлов выполняется функциями **copy() и rename():**

```
copy($source_path, $destination_path);
```

```
rename($oldfile, $newfile);
```

Читатель, возможно, обратил внимание на использование функции **copy()** в листинге 16.2.

Функция **rename()** имеет двойное назначение — кроме переименования файлов, она также выполняет их перемещение. Отдельной функции перемещения файлов в PHP нет. Возможность перемещения файлов из одной файловой системы в другую, а также записи переименуемого файла поверх существующего зависит от операционной системы, поэтому на сервере следует заранее провести все необходимые проверки. Кроме того, следует соблюдать осторожность при указании пути к файлу — относительный путь соотносится с местоположением сценария, а не исходного файла.

Функции запуска программ

Теперь оставим файловую систему и обратимся к функциям запуска команд на сервере.

Эти функции полезны, например, при создании Web-интерфейса к существующей системе запуска программ с командной строки. В частности, мы использовали их для диспетчера рассылки электронной почты **ezmlm** Нам еще предстоит обратиться к этим функциям, когда мы будем рассматривать социологические исследования.

Существует четыре способа запуска программ на Web-сервере. Различия между ними незначительны.

1. **exec()**

   Прототип функции **exec():**

   ```
   string exec (string command [, array result [, int return_value]])
   ```

   В качестве аргумента указывается команда, которую требуется выполнить, например:

   ```
   exec("ls -la");
   ```

   Функция **exec()** не осуществляет непосредственный вывод.

   Она возвращает последнюю строку результата выполнения команды.

   Если указать аргумент **result,** будет возвращаться массив строк, содержащий все строки результата выполнения команды, а если **return_value** — получается код возврата.

2. **passthru()**

   Прототип функции **passthru()**:

   ```
   void passthru (string command [, int return_value])
   ```

   Результат выполнения функции **passthru()** выводится в браузер. Это удобно, если он представлен в двоичном формате — например, графическом.

   Функция значений не возвращает.

   Аргументы те же, что и у функции **exec().**

3. **system()**

   Прототип функции **system()**:

   ```
   string system (string command [, int return_value])
   ```

Функция выводит результат выполнения команды в браузер. Она пытается очистить буферы вывода после каждой строки (при условии запуска PHP в виде модуля сервера), что ее и отличает от функции **passthru()**.

Функция возвращает последнюю строку результата выполнения команды (после успешного выполнения) или значение **false** (в случае ошибки).

Аргументы — те же, что и в других функциях.

4. Обратные кавычки О

Вкратце мы его рассматривали в главе 1. По существу, это знак операции запуска программы.

Обратная кавычка не обеспечивает непосредственного вывода данных. Результатом выполнения команды будет строка. Можно организовать вывод этой строки либо обработать ее как-то иначе.

В листинге 16.5 приведен сценарий с использованием перечисленных выше функций для выполнения одной и той же команды.

Листинг 16.5 **progex.php** — функции получения состояния файлов и возвращаемые результаты

```
<?
    echo "<pre>";
    // версия exec
    exec("ls -la", $result);
    foreach ($result as $line)
       echo "$line\ n";
    echo "<br><hr><br>";
    // версия passthru
    passthru("ls -la");
    echo "<br><hr><br>";
    // версия system
    $result = system("ls -la");
    echo "<br><hr><br>";
     //версия с обратными кавычками
    $result = `ls -al`;
    echo $result;
    echo "</pre>";
?>
```

Одним из этих подходов можно было бы воспользоваться для просмотра каталогов в сценарии, рассмотренном в начале главы.

Если в выполняемую команду включаются данные, представляемые пользователем, ее потребуется прежде пропустить через функцию **escapeshellcmd(),** чтобы не допустить запуска каких-либо команд, которые могут нанести вред системе. Вот пример применения этой функции:

```
system(escapeshellcmd($command_with_user_data));
```

Взаимодействие со средой: getenv() и putenv()

Прежде чем завершить этот раздел, рассмотрим использование переменных среды. Для этого в PHP определены две функции: **getenv(),** возвращающая переменные среды, и **putenv(),** устанавливающая эти переменные.

Заметим, что под средой подразумевается среда на сервере, в которой выполняется PHP.

Для получения списка всех переменных среды PHP используется функция **phpinfo()**. К одним из них приходится обращаться чаще, к другим — реже. Например, следующий оператор:

```
getenv("HTTP_REFERER");
```

возвращает URL-адрес страницы, с которой пользователь перешел на текущую страницу.

Пример установки переменных среды при помощи функции **putenv():**

```
$home = "/home/nobody";
putenv (" HOME=$home ");
```

Более полные сведения о переменных среды можно найти в спецификации CGI по адресу:

```
http://hoohoo.ncsa.uiuc.edu/cgi/env.html
```

Дополнительная информация

Большинство функций файловой системы PHP имеют двойников в операционной системе, поэтому советуем искать дополнительную информацию на страницах руководства по UNIX.

Что дальше

В главе 17 мы рассмотрим использование сетевых функций и функций работы с протоколами для взаимодействия с системами, отличными от сервера, на котором установлен PHP. Это открывает новые возможности применения сценариев.

Глава 17

Использование функций работы с сетью и протоколами

В этой главе исследуются сетевые функции PHP, благодаря которым сценарии могут взаимодействовать с Internet. В нем существует немало ресурсов и большое количество доступных для использования протоколов. В главе обсуждаются следующие вопросы:

- Обзор доступных протоколов
- Отправка и получение почты
- Использование других Web-служб через HTTP
- Применение функций сетевого поиска
- Использование FTP
- Использование общих сетевых соединений с помощью библиотеки cURL

Обзор доступных протоколов

Протоколы представляют собой правила взаимодействия для определенных ситуаций. Например, всем известен протокол встречи с кем-либо: поздороваться, пожать руку, поговорить и затем попрощаться. Подобным же образом устроены и компьютерные сетевые протоколы.

Как и протоколы общения между людьми, различные компьютерные протоколы используются в различных ситуациях и приложениях. HTTP, или протокол передачи гипертекстов (Hypertext Transfer Protocol), используется для передачи Web-страниц. FTP, или протокол передачи файлов (File Transfer Protocol), предназначен для пересылки файлов между машинами по сети. Существует множество других протоколов.

Протоколы и другие стандарты Internet описаны в документах, называемых *RFC (Requestsfor Comments, Запросы на комментарии)*. Эти документы составляются организацией Internet Engineering Task Force (IETF). Документы RFC широко распространены в Internet. Основным их источником является Web-сайт RFC Editor по адресу

```
http://www.rfc-editor.org/
```

Если при работе с определенным протоколом возникают трудности, документы RFC можно использовать как надежный источник информации, полезный при отладке кода. Однако такие документы, как правило, очень подробны и занимают сотни страниц.

Наиболее известными документами являются, например, RFC2616, в котором описывается протокол HTTP/1.1, и RFC822, где описан формат почтовых сообщений Internet.

В этой главе рассматриваются аспекты PHP, использующие некоторые из этих протоколов. Здесь рассказано об отправке сообщений с применением протокола SMTP, получение почты с помощью POP и IMAP, соединении с Web-серверами по HTTP и HTTPS, а также пересылке файлов по FTP.

Отправка и получение почты

Основной способ отправки почтовых сообщений в PHP заключается в простом вызове функции **mail().** Ее применение обсуждалось в главе 4, поэтому здесь мы не будем к ней возвращаться. Эта функция использует для отправки почты протокол SMTP (Simple Mail Transfer Protocol, Простой протокол пересылки почты).

Чтобы добавить **в mail()** дополнительную функциональность, можно воспользоваться одним из множества свободно распространяемых классов. В главе 27 используется почтовый класс HTML MIME, созданный Ричардом Хейесом (Richard Heyes) для отправки HTML-файлов, прикрепленных к почтовому сообщению. SMTP предназначен только для отправки почты. Протоколы IMAP (Internet Message Access Protocol, протокол для доступа к сообщениям Internet, описанный в RFC2060) и POP (Post Office Protocol, почтовый протокол, описанный в RFC1939 и STD0053) используются для получения почты с почтового сервера. Эти протоколы не предназначены для отправки сообщений.

ШАР используется для получения и управления почтовыми сообщениями, сохраненными на сервере, и является более сложным, нежели POP, основное применение которого заключается в простой загрузке сообщений на клиентскую машину и удалении их с сервера.

PHP содержит библиотеку IMAP. Ею можно воспользоваться для установления не только IMAP-соединений, но и соединений POP и NNTP (Network News Transfer Protocol, Протокол передачи сетевых новостей).

Использование библиотеки IMAP будет подробно рассматриваться в главе 26.

Использование других Web-служб через NTTP

Одно из лучших применений Web заключается в возможности использовать, изменять или встраивать существующие службы и информацию в собственные страницы. PHP существенно упрощает упомянутые задачи. Далее приведен соответствующий пример.

Предположим, что компании, в которой вы работаете, требуется отображать на своей начальной странице котировки акций. Эта информация доступна на сайте фондовой биржи, но как ее получить?

Прежде всего, необходимо определить исходный URL, по которому следует искать информацию. После этого всякий раз, когда кто-либо посещает начальную страницу компании, можно открывать соединение с этим URL, получать страницу и выводить требуемую информацию.

В качестве примера рассмотрим сценарий, который получает и форматирует биржевую информацию NASDAQ (Система автоматической котировки Национальной ассоциации биржевых дилеров). В целях примера биржевые котировки запрашиваются из сайта Amazon.com. (Требуемая информация может отличаться, но принципы останутся такими же.) Сценарий приведен в листинге 17.1.

Листинг 17.1 lookup.php — сценарий запрашивает информацию о котировках NASDAQ, символ тикера (биржевого аппарата, передающего котировки ценных бумаг) задается переменной $symbol

```
<html>
<head>
   <title>Stock Quote from NASDAQ</title>
</head>
<body>
<?
   // выбор сайта с биржевой информацией
   $symbol="AMZN";
   echo "<h1>Stock Quote for $symbol</h1>";

   // соединиться с указанным URL и прочесть информацию
   $theurl = "http://quotes.nasdaq-amex.com/Quote.dll?"
             ."page=multi&mode=Stock&symbol=".$symbol;
   if (!($fp = fopen($theurl, "r")))
   {
      echo "Could not open URL";
      exit;
   }
   $contents = fread($fp, 1000000);
   fclose($fp);

   // найти часть страницы, которую следует отобразить
   $pattern = "(\\$[0-9 ]+\\. [0-9]+)";
   if (eregi($pattern, $contents, $quote))
   {
      echo "$symbol was last sold at: ";
      echo $quote[1];
   } else
   {
      echo "No quote available";
   };

   // указать источник информации
   echo "<br>"
         ."This information retrieved from <br>"
         ."<a href=\"$theurl\">$theurl</a><br>"
         ."on ".(date("l js F Y g:i a T"));
?>
</body>
</html>
```

Пример вывода после запуска сценария показан на рис. 17.1.

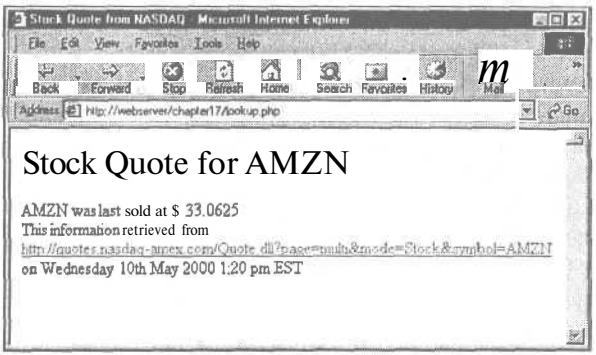


РИСУНОК 17.1
В сценарии для получения биржевых котировок NASDAQ используются регулярные выражения.

Сценарий достаточно прост — фактически в нем не используются функции, не рассмотренные ранее.

Здесь стоит вспомнить, что когда обсуждалось чтение из файлов в главе 2, было оговорено, что эти функции можно использовать для чтения из URL. Именно так и происходит в данном примере. Вызов функции **fopen()**

```
$£p = fopen($theurl, r)
```

возвращает указатель на начало страницы из заданного URL. Все сводится просто к чтению страницы из URL и закрытию соединения:

```
$contents = fread($fp, 1000000);
fclose($fp);
```

Обратите внимание, что при чтении из файла используется большое число. Когда файл находится на локальном сервере, можно использовать число **filesize(*$file*)**, но для URL это не работает.

После этого полный текст Web-страницы уже содержится в переменной **$contents**. Для поиска требуемой части страницы используются регулярные выражения и функция **eregi()**:

```
$pattern = (\\\ $[0-9 ]+\\.[0-9]+);
if (eregi($pattern, $contents, $quote))
{
     echo $symbol was last sold at: ;
     echo $quote[1];
}
```

И все!

Этим подходом можно воспользоваться для многих целей. Другим примером может послужить помещение на страницу информации о погоде.

Наилучший подход представляет собой комбинирование информации из различных источников, что придает странице некоторый вес и изящество. Примером может послужить известный сценарий Филиппа Гринспана (Philip Greenspun) под названием Bill Gates Wealth Clock (Счетчик состояния Билла Гейтса):

```
http://www.webho.com/WealthClock
```

Эта страница имеет два источника информации. Текущее население США запрашивается из сайта U.S. Census Bureau (Бюро переписи населения США). Затем запрашивается текущий курс акций Microsoft, эти данные объединяются, добавляется изряд-

ная часть, необходимая по мнению автора, и вычисляется новая информация — приблизительная оценка состояния Билла Гейтса.

Одно стороннее замечание: если используется внешняя информация, причем в коммерческих целях, как в этом примере, стоит внимательно изучить источник. В некоторых случаях могут возникнуть проблемы, связанные с охраной прав на интеллектуальную собственность.

Если создается сценарий наподобие приведенного, может возникнуть необходимость передачи данных, например, параметров, введенных пользователем, при соединении с внешним URL. В этом случае стоит воспользоваться функцией **url_encode()**. Она преобразует заданную строку в формат, более подходящий для URL, заменяя, например, пробелы знаками плюс. Вызов функции выглядит так:

```
$encodedparameter = url_encode($parameter);
```

Применение функций сетевого поиска

PHP содержит набор "поисковых" функций, предназначенных для проверки информации об именах хостов, IP-адресах и почтовых обменах. Например, если создается сайт наподобие Yahoo!, куда могут добавляться новые URL, можно автоматически проверить правильность информации о хосте и контактной информации. Таким образом можно сэкономить немало усилий по отслеживанию несуществующих сайтов и некорректных почтовых адресов.

Пример HTML-формы для ввода информации в поисковый сайт приведен в листинге 17.2.

Листинг 17.2 directory_submit.html — **HTML-форма для ввода информации**

```
<head>
  <title>Submit your site</title>
</head>
<body>
<h1>Submit site</h1>
<form method=post action="directory_submit.php">
URL: <input type=text name="url" size=30 value="http://"><br>
Email contact: <input type=text name="email" size=23Xbr>
<input type="submit" name="Submit site">
</form>
</body>
</html>
```

Это очень простая форма — вместе с введенными данными она показана на рис. 17.2.



РИСУНОК 17.2
Формы ввода информации на поисковых сайтах, как правило, требуют URL сайта и контактную информацию, используя которую администраторы могут известить о добавлении сайта в поисковую систему.

При нажатии кнопки Submit (Отправить) в первую очередь необходимо убедиться, что URL размещен на реальной машине, а затем, что хостовая часть почтового адреса также отвечает реальной машине в сети. Далее приводится сценарий, выполняющий упомянутые действия, а его вывод показан на рис. 17.3.

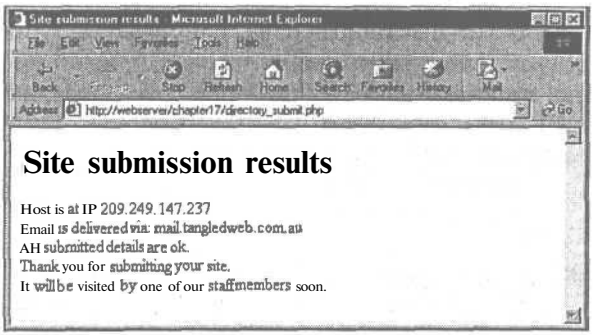


РИСУНОК 17.3
Эта версия сценария выводит результаты проверки имени хоста URL и почтового адреса — в законченной версии эта информация может и не отображаться, но здесь нас интересуют результаты проверки.

Сценарий, производящий проверки, использует две функции из набора сетевых функций PHP — **gethostbyname()** и **getmxrr()**. Полный сценарий находится в листинге 17.3.

Листинг 17.3 directory_submit.php — **сценарий для проверки URL и почтового адреса**

```
<html>
<head>
  <title>Site submission results</title>
</head>
<body>
<h1>Site submission results</h1>
<?
  // Проверить URL
  $url = parse_url($url);
  $host = $url[host];
  if(!($ip = gethostbyname($host)))
  {
    echo "Host for URL does not have valid IP";
    exit;
  }
  echo "Host is at IP $ip <br>";
  // Проверить почтовый адрес
  $email = explode("@", $email);
  $emailhost = $email[1];
  if (!getmxrr($emailhost, $mxhostsarr))
  {
    echo "Email address is not at valid host";
    exit;
  }
  echo "Email is delivered via: ";
  foreach ($mxhostsarr as $mx)
    echo "$mx ";
  // Если сценарий дошел до этой точки, значит все в порядке
  echo "<br>All submitted details are ok.<br>";
  echo "Thank you for submitting your site.<br>"
       ."It will be visited by one of our staff members soon."
```

```
    // В реальном случав добавить в базу данных ожидающих сайтов...
?>
</body>
</html>
```

Рассмотрим наиболее интересные части сценария.

Вначале к заданному URL применяется функция **parse_url().** Эта функция возвращает ассоциативный массив различных частей имени URL. Доступные блоки информации включают в себя протокол **(scheme),** пользователя **(user),** пароль **(pass),** хост **(host),** порт **(port),** путь **(path),** запрос **(query)** и фрагмент **(fragment).** Обычно все эти элементы не требуются, но для примера показано, как они формируют URL.

Для заданного URL

```
http://nobody:secret@bigcompany.com:80/script.php?variable=value#anchor
```

элементы массива имеют следующие значения

- scheme: http://
- user: nobody
- pass: secret
- host: bigcompany.com
- port: 80
- path: script.php
- query: variable=value
- fragment: anchor

В сценарии требуется только информация о хосте, поэтому массив обрабатывается следующим образом:

```
$url = parse_url($url);
$host = $url[host];
```

После этого можно получить IP-адрес хоста, если он содержится в DNS-сервере, с использованием функции **gethostbyname()**. Она возвращает IP-адрес, если он существует или значение false в противном случае:

```
$ip = gethostbyname($host)
```

Здесь можно идти и другим путем, воспользовавшись функцией **gethostbyaddr(),** которая по IP-адресу хоста возвращает его символическое имя. Если применить две эти функции последовательно, то в конце может получиться имя хоста, отличное от того, с которого начинался поиск. Это может означать, что сайт использует службу виртуальных хостов.

Если искомый URL существует, далее выполняется проверка почтового адреса. Вначале он разбивается на имя пользователя и имя хоста при помощи функции **explode()**:

```
$email = explode(@, $email);
$emailhost = $email[1];
```

Имея имя хоста, несложно проверить, можно ли на него отправить почту. Это делается с помощью функции **getmxrr()**:

```
getmxrr($emailhost, $mxhostsarr)
```

Функция возвращает набор записей MX (Mail Exchange, почтовый обмен) для этого адреса в массиве, заданном переменной **$mxhostarr**.

MX-запись содержится в DNS и ее поиск происходит так же, как поиск имени хоста. Машина, указанная в записи MX необязательно является той, куда придет почта. Однако она знает, куда необходимо перенаправить почту. (Их может быть несколько, поэтому функция возвращает массив, а не строку с именем хоста.) Если в DNS не содержится MX-записи для данного хоста, значит, почте просто некуда идти.

Если все проверки завершаются успешно, данные можно разместить в базе данных для последующей проверки кем-либо из сотрудников поискового сайта.

Кроме упомянутых функций можно воспользоваться более общей функцией **checkdnsrr()** Она по имени хоста возвращает значение true, если для него существует какая-либо запись в DNS.

Использование FTP

File Transfer Protocol (протокол передачи файлов), или FTP, используется для передачи файлов между хостами в сети. Используя PHP, можно применять **fopen()** и другие файловые функции для FTP-соединений так же, как и для HTTP-соединений, для установки соединения и передачи файлов на или с FTP-сервера. Кроме того, в стандартной инсталляции PHP имеется набор функций, специфичных для FTP.

Однако эти функции по умолчанию не включаются. Для того чтобы воспользоваться ими под UNIX, необходимо запустить конфигурационную PHP-программу **configure** с опцией **--enable-ftp**, а затем перезапустить **make.** Для использования FTP-функций с двоичным модулем Win32, следует добавить строку

```
extension=php_ftp.dll
```

в разделе "Windows Extensions" ("Расширения Windows") файла **php.ini**. (Более подробно о конфигурировании PHP рассказывается в приложении А.)

Использование FTP для создания резервной или зеркальной копии файла

FTP-функции полезны при перемещении или копировании файлов на и с других хостов. Один из распространенных примеров их использования — это создание резервной копии Web-сайта или зеркальной копии файлов на другом сервере. В листинге 17.4 приведен простой пример — сценарий, использующий FTP-функции для создания зеркальной копии файла.

Листинг 17.4 ftpmirror.php — сценарий для загрузки новых версий файла с FTP-сервера

```
<html>
<head>
   <title>Mirror update</title>
</head>
<body>
<h1>Mirror update</h1>
<?
// установка переменных — можно изменить для своих целей
$host = "ftp.cs.rmit.edu.au";
$user = "anonymous";
$password = "laura@tangledweb.com.au";
$remotefile = "/pub/tsg/ttssh14.zip";
$localfile = "$DOCUMENT_ROOT/../writable/ttssh14.zip";
```

```
// установка соединения с хостом
$conn = ftp_connect("$host");
if (!$conn)
{
   echo "Error: Could not connect to ftp server<br>";
   exit;
}
echo "Connected to $host.<br>";

// регистрация на хосте
@ $result = ftp_login($conn, $user, $pass);
if (!$result)
{
   echo "Error: Could not log on as $user<br>";
   ftp_quit($conn);
   exit;
}
echo "Logged in as $user<br>";

// проверка времени файла — следует ли его обновлять
echo "Checking file time...<br>";
if (file_exists($localfile))
{
   $localtime = filemtime($localfile);
   echo "Local file last updated ";
   echo date("G:i j-M-Y", $localtime);
   echo "<br>";
}
else
   $localtime=0;
$remotetime = ftp_mdtm($conn, $remotefile);
if (!($remotetime >= 0))
{
    // Это не значит, что файла там нет, сервер может просто не
    // поддерживать "время модификации"
    echo "Can't access remote file time.<br>";
    $remotetime=$localtime+1;  // проверка обновления
}
else
{
   echo "Remote file last updated ";
   echo date("G:i j-M-Y", $remotetime);
   echo "<br>";
}
if (!($remotetime > $localtime))
{
    echo "Local copy is up to date.<br>";
    exit;
}

// загрузка файла
echo "Getting file from server...<br>";
$fp = fopen ($localfile, "w");
if (!$success = ftp_fget($conn, $fp, $remotefile, FTP_BINARY))
{
   echo "Error: Could not download file";
   ftp_quit($conn);
   exit;
}
fclose($fp);
echo "File downloaded successfully";
```

```
// отсоединение от хоста
ftp_quit($conn);

?>
</body>
</html>
```

Вывод после запуска сценария показан на рис. 17.4.

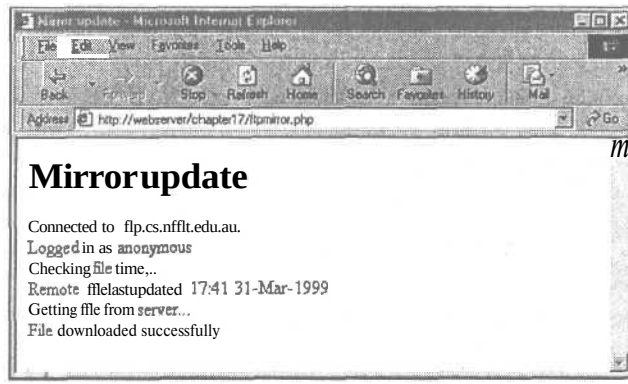


РИСУНОК 17.4
Сценарий по созданию зеркальной копии файла по FTP проверяет, является ли локальная версия файла новой и если нет, загружает новую версию.

Этот сценарий является достаточно общим. Он начинается с установки переменных:

```
$host = ftp.cs.rmit.edu.au;
$user = anonymous;
$password = laura@tangledweb.com.au;
$remotefile = /pub/tsg/ttssh14.zip;
$localfile = $DOCUMENT_ROOT/../writable/ttssh14.zip;
```

Переменная **$host** содержит имя FTP-сервера, с которым будет происходить соединение, а переменные **$user** и **$password** — соответственно, имя пользователя и пароль, необходимые для регистрации на сервере.

Многие FTP-сайты поддерживают так называемый *анонимный вход (anonymous login),* т.е. свободно доступное имя, которым для регистрации может воспользоваться любой пользователь. Пароль не требуется, но распространенный этикет состоит в вводе своего почтового адреса в качестве пароля, чтобы системные администраторы могли увидеть, откуда приходят их пользователи. Эти правила соблюдаются и здесь.

Переменная **$remotefile** содержит путь к файлу, который требуется загрузить. В данном случае происходит загрузка и создание зеркальной локальной копии программы Tera Term SSH — клиента SSH под Windows. (SSH означает secure shell (защищенный командный интерпретатор). Это аналог Telnet, в котором при передаче информации применяется кодирование.)

Переменная **$localfile** содержит путь, по которому следует разместить загруженный файл на локальной машине.

Можно изменить значения переменных, чтобы воспользоваться сценарием в собственных целях.

Основные шаги в этом сценарии совпадают с действиями, предпринимаемыми при загрузке файла через FTP вручную с использованием интерфейса командной строки:

1. Соединиться с удаленным FTP-сервером.
2. Зарегистрироваться (как пользователь или же под анонимной учетной записью).

3. Проверить, обновлялся ли файл на удаленной машине.
4. Если да, загрузить его.
5. Закрыть FTP-соединение.

Рассмотрим эти шаги последовательно.

Соединение с удаленным FTP-сервером

Этот шаг равносилен вводу

```
ftp hostname
```

в командной строке Windows или UNIX. В PHP за выполнение этого шага отвечает следующий фрагмент кода:

```
$conn = ftp_connect($host);
if (!$conn)
{
   echo Error: Could not connect to ftp server<br>;
   exit;
}
echo Connected to $host.<br>;
```

Здесь происходит вызов функции **ftp_connect().** Аргументом функции является имя хоста, а возвращаемым значением — либо дескриптор соединения, либо значение false, если соединиться не удалось. Вторым, необязательным параметром функции является номер порта. (Здесь он не используется.) Если номер порта не указан, по умолчанию FTP-соединение осуществляется через порт 21, стандартный FTP-порт.

Регистрация на FTP-сервере

Следующий шаг состоит в регистрации на сервере под именем определенного пользователя с определенным паролем. Он достигается вызовом функции **ftp_login():**

```
@ $result = ftp_login($conn, $user, $pass);
if (!$result)
{
   echo "Error: Could not log on as $user<br>";
   ftp_quit($conn);
   exit;
}
echo "Logged in as $user<br>";
```

Эта функция принимает три аргумента: дескриптор FTP-соединения (возвращаемый функцией **ftp_connect()),** имя пользователя и пароль. Если пользователь смог зарегистрироваться, функция возвращает значение true, в противном случае — значение false. Обратите внимание, что в начале первой строки находится символ @, который подавляет вывод ошибок. Так делается потому, что иначе, если пользователь не сможет зарегистрироваться, в окне браузера будет выводиться предупреждение PHP. Однако можно просто перехватить ошибку, как это и делается здесь путем проверки значения переменной **$result**, и выдать свое, более дружественное и информативное сообщение.

Отметьте, что если регистрации не происходит, то фактически FTP-соединение закрывается функцией **ftp_quit();** об этом — далее.

Проверка времени обновления файла

Поскольку требуется обновить локальную копию файла, важно сперва проверить, следует ли обновлять файл, поскольку вовсе нет необходимости заново загружать файл,

особенно большого размера, если копия совпадает с оригиналом. Это позволяет снизить трафик сети. Рассмотрим фрагмент кода, который решает эту задачу.

Вначале проверяется наличие локальной копии файла при помощи функции **file_exists()**. Если копии нет, то, очевидно, файл необходимо загрузить. Если же она существует, то при помощи функции **filemtime()** запрашивается время последней модификации файла, которое затем присваивается переменной **$localtime** Если локальной копии файла не существует, переменной **$localtime** присваивается значение 0, т.е. дата локального файла становится заведомо "старше" любой даты модификации файла на удаленной машине:

```
echo "Checking file time...<br>";
if (file_exists($localfile))
{
   $localtime = filemtime($localfile);
   echo "Local file last updated ";
   echo date("G:i j-M-Y", $localtime);
   echo "<br>";
}
else
   $localtime=0;
```

(О функциях **ffle_exists()** и **filemtime()** можно узнать подробнее в главах 2 и 16.)

После выяснения вопроса с локальным временем необходимо узнать время модификации файла на удаленной машине. Для этого существует функция **ftp_mdtm()**:

```
$remotetime = ftp_mdtm($conn, $remotefile);
```

Эта функция принимает два аргумента — дескриптор FTP-соединения и путь к удаленному файлу. Функция возвращает либо время последней модификации файла в формате UNIX, либо -1, если произошла какая-либо ошибка. Не все FTP-серверы поддерживают упомянутое свойство, поэтому вызов этой функции не всегда дает требуемый результат. В этом случае переменная **$remotetime** становится искусственно "новее", чем **$localtime**, так как ей присваивается значение **$localtime+1**. А это значит, что будет предпринята попытка загрузить файл:

```
if (!($remotetime >= 0))
{
    // Это не значит, что файла там нет, сервер может просто не
    // поддерживать "время модификации"
    echo "Can't access remote file time.<br>";
    $remotetime=$localtime+1;  // убедиться, что обновление произошло
}
else
{
   echo "Remote file last updated ";
   echo date("G:i j-M-Y", $remotetime);
   echo "<br>";
}
```

Теперь, когда имеются даты двух файлов, их можно сравнить и сделать вывод о необходимости загрузки файла:

```
if {!($remotetime > $localtime))
{
    echo "Local copy is up to date.<br>";
    exit;
}
```

Загрузка файла

На этой стадии происходит попытка загрузить файл с сервера:

```
echo "Getting file from server...<br>";
$fp = fopen ($localfile, "w");
if (!$success = ftp_fget($conn, $fp, $remotefile, FTP_BINARY))
{
   echo "Error: Could not download file";
   fclose($fp);
   ftp_quit($conn);
   exit;
}
fclose($fp);
echo "File downloaded successfully";
```

Как было показано ранее, локальный файл открывается при помощи функции **fopen()**. После этого вызывается функция **ftp_fget(),** которая загружает файл и сохраняет его локально. Эта функция принимает четыре аргумента. Назначение первых трех из них очевидно — это дескриптор FTP-соединения, дескриптор локального файла и путь к удаленному файлу. Четвертый параметр определяет режим FTP-доступа.

Пересылка через FTP может осуществляться в двух режимах: ASCII и двоичном. Режим ASCII используется для пересылки текстовых файлов (т.е. файлов, состоящих только из ASCII-символов), а двоичный режим — для пересылки всех остальных файлов. FTP-библиотека PHP содержит две предопределенных константы — **FTP_ASCII** и **FTP_BINARY**, которые соответствуют двум этим режимам. Необходимо выяснить, какой режим соответствует типу файла и передать требуемую константу в четвертом аргументе функции **ftp_fget().** Вданном случае происходит пересылка zip-файла, поэтому применяется режим **FTP_BINARY**.

Функция **ftp_fget()** возвращает значение true, если все в порядке, или false, если произошла ошибка. Результат присваивается переменной **$success**; он позволяет сообщить пользователю, как прошла пересылка файла.

После загрузки локальный файл закрывается при помощи функции **fclose()**.

Вместо **ftp_fget()** можно было бы воспользоваться альтернативной функцией — **ftp_get(),** которая имеет следующий прототип:

```
int ftp_get (int ftp_connection, string localfile_path,
   string remotefile_path, int mode)
```

Эта функция работает так же, как **ftp_fget(),** но не требует открытия локального файла. Ей необходимо указать имя локального файла, в который будет происходить запись, а не его дескриптор.

Обратите внимание, что в PHP не существует эквивалента FTP-команды mget, которая используется для загрузки нескольких файлов. Вместо этого следует применять несколько вызовов **ftp_fget()** или **ftp_get().**

Отсоединение

По окончании следует закрыть FTP-соединение при помощи функции **ftp_quit()**:

```
ftp_quit($conn);
```

Аргументом функции является дескриптор FTP-соединения.

Загрузка файлов на сервер

Если требуется копировать файлы с локального сервера на удаленную машину, можно воспользоваться двумя функциями, противоположными **ftp_fget() и ftp_get().** Они называются **ftp_fput() и ftp_put()** и имеют следующие прототипы:

```
int ftp_fput (int ftp_connection, string remotefile_path,
   int fp, int mode)
int ftp_put (int ftp_connection, string remotefile_path,
   string localfile_path, int mode)
```

Их аргументы совпадают с аргументами их _get-эквивалентов.

Как избежать тайм-аутов

Одна из проблем, с которыми можно столкнуться при загрузке файлов через FTP, связана с превышением максимального времени выполнения. О такой ситуации всегда можно узнать из сообщения об ошибке, которое выдает PHP. Так происходит, если локальный сервер использует медленную или переполненную сеть, или если случается загрузка большого файла, например, видеоролика.

Максимальное время выполнения по умолчанию для всех сценариев PHP определено в файле **php.ini.** По умолчанию его значение составляет 30 секунд. Оно предназначено для отладки сценариев, которые по каким-либо причинам становятся неуправляемыми. Однако при загрузке файлов через FTP, если связь с остальным миром является медленной или размер файла достаточно велик, пересылка может занять больше времени.

К счастью, максимальное время выполнения определенного сценария можно изменить, используя функцию **set_time_limit()**. Вызов этой функции устанавливает максимальное число секунд для выполнения сценария, начиная с момента вызова функции. Например,

```
set_time_limit(90);
```

позволит сценарию выполняться 90 секунд с момента вызова функции.

Другие функции работы с FTP

В PHP имеется несколько других полезных функций работы с FTP.

Функция **ftp_size()** возвращает размер файла на удаленном сервере и имеет следующий прототип:

```
int ftp_size(int ftp_connection, string remotefile_path)
```

Она возвращает размер удаленного файла в байтах или значение -1 в случае ошибки. Некоторые FTP-серверы эту функцию не поддерживают.

Функция **ftp_size()** позволяет оценить максимальное время выполнения для определенной пересылки. Зная размер файла и скорость соединения, можно приблизительно оценить время пересылки и соответствующим образом воспользоваться функцией **set_time_limit()**.

Список файлов в каталоге на удаленном FTP-сервере можно получить при помощи следующего фрагмента кода:

```
$listing = ftp_nlist($conn, "$directory_path");
foreach ($listing as $filename)
  echo "$filename <br>";
```

Для получения списка имен файлов в определенном каталоге используется функция **ftp_nlist().**

Другие FTP-функции в PHP позволяют выполнить практически все, что можно сделать из командной строки FTP-клиента. FTP-функции, соответствующие командам FTP, можно найти в электронном руководстве по PHP по адресу

```
http://php.net/manual/ref.ftp.php
```

Исключение составляет команда mget (multiple get), однако можно воспользоваться функцией **ftp_nlist()** для получения списка файлов, а затем поочередно загрузить их.

Использование общих сетевых соединений с помощью библиотеки cURL

В PHP (начиная с версии 4.0.2) определен набор функций, который действует как интерфейс с cURL, библиотекой функций Client URL из файла libcurl, написанного Дениэлом Стенбергом (Daniel Stenberg).

Ранее в этой главе рассматривалась функция **fopen()**, а также функция для чтения удаленного файла через HTTP. Этим и ограничиваются возможности **fopen()**. Кроме того, было показано, как можно установить FTP-соединение, используя FTP-функции.

Функции cURL позволяют устанавливать соединения через FTP, HTTP, HTTPS, Gopher, Telnet, DICT, FILE и LDAP. Кроме того, можно использовать сертифицированное соединение по HTTPS, пересылать параметры HTTP POST и HTTP GET, загружать файлы на сервер по FTP или HTTP, работать через прокси-серверы, устанавливать cookie-наборы и осуществлять простую аутентификацию пользователя через HTTP.

Другими словами, при помощи cURL можно реализовать практически любое сетевое соединение.

Для использования cURL с PHP необходимо загрузить библиотеку libcurl, откомпилировать ее и запустить конфигурационный сценарий PHP **configure** с опцией **--with-curl=[path]**. Путь **path** должен содержать каталог библиотеки и каталоги с заголовочными файлами. Библиотеку можно отыскать на сайте

```
http://curl.haxx.se/
```

Обратите внимание, что для работы с PHP требуется версия cURL не ниже 7.0.2-beta.

Для использования cURL достаточно освоить лишь несколько простых функций. Обычно процедура применения этой библиотеки выглядит следующим образом

1. Начать сеанс cURL по вызову функции **curl_init()**.
2. Установить параметры пересылки с помощью функции **curl_setopt()**. Именно здесь указываются такие аргументы, как URL, параметры, передаваемые этому URL, или точка назначения для вывода, получаемого с URL.
3. Когда все готово, необходимо вызвать функцию **curl_exec(),** выполняющую реальное соединение.
4. Закрыть сеанс cURL с использованием функции **curl_close().**

Единственное, что может измениться — это URL и параметры, передаваемые **curl_opt().** Количество настраиваемых аргументов этой функции достаточно велико.

Типовые способы применения библиотеки cURL включают в себя

- Загрузку страниц с сервера, использующего HTTPS (функцию **fopen()** нельзя применять для таких целей)
- Соединение со сценарием, который ожидает получения данных от HTML-формы методом **POST**
- Написание сценария, отправляющего наборы тестовых данных сценариям с целью проверки их вывода

Здесь рассматривается первый пример — простое применение, которое, однако, не может быть реализовано другими способами.

Пример, приведенный в листинге 17.5, соединяется с сервером Equifax Secure Server по HTTPS и загружает найденный на нем файл на локальный Web-сервер.

Листинг 17.5 **https-curl.php** — сценарий для установки **HTTPS-соединений**

```
<?
echo "<h1>HTTPS transfer with cURL</h1>";
$outputfile = "$DOCUMENT_ROOT/../writable/equifax.html";
$fp = fopen($outputfile, "w");
echo "Initializing cURL session...<br>";
$ch = curl_init();
echo "Setting cURL options...<br>";
curl_setopt ($ch, CURLOPT_URL, "https://equifaxsecure.com");
curl_setopt ($ch, CURLOPT_FILE, $fp);
echo "Executing cURL session...<br>";
curl_exec ($ch);
echo "Ending cURL session...<br>";
curl_close ($ch);
fclose($fp);
?>
```

Рассмотрим этот сценарий. Он начинается с открытия локального файла при помощи функции **fopen().** В нем будет сохраняться страница, пересланная через защищенное соединение.

После этого создается сеанс cURL с помощью функции **curl_init()**:

```
$ch = curl_init();
```

Функция возвращает дескриптор сеанса cURL. Ее можно вызвать так же, как здесь, без параметров, либо же указать строку с URL. Кроме того, URL можно задать с помощью функции **curl_setopt(),** как и сделано в данном случае:

```
curl_setopt ($ch, CURLOPT_URL, "https://equifaxsecure.com");
curl_setopt ($ch, CURLOPT_FILE, $fp);
```

Функция **curl_setopt()** получает три аргумента. Первый из них является дескриптором сеанса, второй — именем устанавливаемого параметра, а третий — значением указанного параметра.

В данном случае устанавливаются две опции. Первая — это URL, с которым необходимо установить соединение. За это отвечает параметр **CURLOPT_URL.** Вторая опция — файл, в который будут направляться данные, поступающие через соединение. Если не указать файл, то данные будут направляться в стандартный выходной поток — как правило, браузер. В данном случае указан открытый ранее дескриптор файла.

После установки всех опций cURL выполняет реальное соединение:

```
curl_exec ($ch);
```

Вызов этой функции устанавливает соединение с указанным URL, загружает страницу и сохраняет ее в файле, на который указывает $fp.

После этого следует закрыть сеанс cURL и файл, в который производилась запись:

```
curl_close ($ch);
fclose($fp);
```

Вот и все по этому простому примеру.

Стоит ознакомиться с классом Snoopy, который можно найти по адресу

```
http://snoopy.sourceforge.net/
```

Этот класс с помощью cURL обеспечивает функциональность Web-клиента.

Дополнительная информация

В этой главе уделялось много внимания основам, но, как несложно догадаться, на эти темы существует еще немало материалов.

Информацию об отдельных протоколах и принципах их работы можно найти в документах RFC по адресу

```
http://www.rfc-editor.org/
```

Интересную информацию о протоколах можно отыскать на сайте Консорциума World Wide Web (World Wide Web Consortium):

```
http://www.w3.org/Protocols/
```

Протоколу TCP/IP посвящено великое множество книг. Выберите любую доступную.

На Web-сайте cURL содержатся советы по применению версии командной строки функций cURL. Их легко перевести в PHP-версии:

```
http://curl.haxx.se/docs/httpscripting.shtml
```

Что дальше

В главе 18 рассматриваются библиотеки PHP-функций для работы с датой и календарем. Будет показано, как преобразовать форматы, введенные пользователем, в форматы PHP и MySQL и обратно.

Глава 18

Управление датой и временем

В этой главе обсуждается проверка и форматирование даты и времени, а также их преобразование в различные форматы. Последнее особенно важно при преобразовании между форматами MySQL и PHP, UNIX и PHP, а также для дат, введенных пользователем в HTML-формах.

В этой главе рассматриваются следующие вопросы:

- Получение даты и времени средствами PHP
- Преобразования даты в форматах PHP и MySQL
- Операции над датами
- Использование календарных функций

Получение даты и времени средствами PHP

В главе 1 упоминалось об использовании функции date() для получения и форматирования даты и времени в PHP. Здесь об этой и других PHP-функциях рассказывается более подробно.

Использование функции date()

Функция date() принимает два аргумента, один из которых является необязательным. Первый аргумент представляет собой строку формата, а второй, необязательный, – метку времени UNIX. Если метка времени не указана, то функция date() обрабатывает текущую дату и время. Она возвращает отформатированную строку, содержащую дату.

Типовый вызов функции выглядит так:

```
echo date("jS F Y");
```

Вывод этого выражения имеет вид "31th July 2001".

Коды форматирования, используемые функцией date(), перечислены в табл. 18.1.

Таблица 18.1 Коды форматирования PHP-функции date()

Кол	Описание
a	Утро или время после полудня, представленное двумя строчными символами, "am" или "pm".
A	Утро или время после полудня, представленное двумя прописными символами, "AM" или "PM".
B	Internet-время Swatch — универсальная временная схема. Более подробно о ней можно узнать на сайте **http://www.swatch.com.**
d	День месяца в виде двузначного числа с ведущим нулем. Диапазон значений — от "01" до "31".
D	День недели в виде трехбуквенной аббревиатуры. Диапазон значений — от "Mon" (понедельник) до "Sun" (воскресенье).
F	Месяц в полнотекстовом формате. Диапазон значений — от "January" (январь) до "December" (декабрь).
g	Часы в 12-часовом формате без ведущих нулей. Диапазон значений — от "1" до "12".
G	Часы в 24-часовом формате без ведущих нулей. Диапазон значений — от "О" до "23".
h	Часы в 12-часовом формате с ведущими нулями. Диапазон значений — от "01" до "12".
H	Часы в 24-часовом формате с ведущими нулями. Диапазон значений — от "00" до "23".
i	Минуты с ведущими нулями. Диапазон значений — от "00" до "59".
I	Переход *на* летнее время, представленный значением логического типа. Если переход на летнее время установлен, функция возвращает значение "1", иначе — "О".
j	День месяца в виде числа без ведущих нулей. Диапазон значений — от "1" до "31".
l	День недели в полнотекстовом формате. Диапазон значений — от "Monday" (понедельник) до "Sunday" (воскресенье).
L	Високосный год, представленный значением логического типа. Функция возвращает значение "1", если дата принадлежит високосному году, и "О" — в противном случае.
m	Месяц в двузначном числовом формате с ведущими нулями. Диапазон значений — от "01" до "12".
M	Месяц в виде трехбуквенной аббревиатуры. Диапазон значений — от "Jan" (январь) до "Dec" (декабрь).
п	Месяц в виде числа без ведущих нулей. Диапазон значений — от "1" до "12".
s	Секунды с ведущими нулями. Диапазон значений от "00" до "59".
S	Порядковый суффикс для дат в двухбуквенном формате. Он может принимать значение "st", "nd", "rd" или "th" в зависимости от числа, за которым он следует.
t	Полное количество дней в месяце. Диапазон значений — от "28" до "31".
T	Временная зона сервера, заданная в трехбуквенном формате, например, "EST".
U	Число секунд с 1 января 1970 г. до текущего момента; его также называют меткой времени UNIX для текущей даты.
w	День недели в виде числа. Диапазон значений — от "О" (воскресенье) до "6" (суббота).
y	Год в двузначном формате, например, "00".
Y	Год в четырехзначном формате, например, "2000".
z	День года в виде числа. Диапазон значений — от "О" до "365".
Z	Смещение текущей временной зоны в секундах. Диапазон значений — от "-43200" до "43200".

Работа с метками времени UNIX

Второй аргумент функции date() является меткой времени UNIX.

Большинство UNIX-подобных систем хранят текущее время в виде 32-разрядного целого числа секунд, начиная с полуночи 1 января 1970 г. по Гринвичу. Эту дату называют началом эпохи UNIX. Это может выглядеть как некая тайна для посвященных, тем не менее, это стандарт.

Метки времени UNIX — компактный способ хранения даты и времени и стоит отметить, что на него совершенно не повлияла проблема 2000-го года (Y2K), от которой "пострадали" другие сокращенные форматы хранения даты. Тем не менее, для программного обеспечения в 2038 г. подобная проблема может возникнуть. Хотя метки времени не имеют фиксированного размера, они "привязаны" к размеру длинного целого в C (32 бита). Наиболее вероятным решением к 2038 г. будет использование в компиляторах более емкого типа.

Даже если PHP запускается на Windows-сервере, все равно функция date() и другие PHP-функции используют именно такой формат хранения даты.

Если требуется преобразовать время и дату в формат метки времени UNIX, можно воспользоваться функцией **mktime()**, которая имеет следующий прототип:

```
int mktime (int hour, int minute, int second, int month,
   int day, int year [, int is_dst])
```

Назначение аргументов вполне очевидно кроме последнего, is_dst, который указывает, действует ли переход на летнее время. Его можно установить равным 1, если переход на летнее время действует, и 0 — если нет, либо -1 (значение по умолчанию), если это неизвестно. В любом случае, этот аргумент является необязательным и поэтому используется редко.

Основная ошибка, которой следует избегать при использовании функции **mktime()** — неинтуитивный порядок аргументов. Порядок не позволяет пропустить время. Если время не является важным, можно установить часы, минуты и секунды равными 0. Однако, можно опустить значения с правой стороны списка аргументов. Незаданные величины примут значения текущего времени. Следовательно, вызов функции

```
$timestamp = mktime();
```

вернет метку времени UNIX для текущей даты и времени. То же самое можно, конечно же, получить и с помощью такого вызова функции

```
$timestamp = date("U");
```

В функцию **mktime()** можно передать год как в дву-, так и в четырехзначном формате. Двузначные значения от 0 до 69 интерпретируются как годы от 2000 до 2069, а от 70 до 99 — как годы от 1970 до 1999.

Использование функции getdate()

Другая функция определения даты, которая может оказаться полезной, называется getdate(). Она имеет следующий прототип:

```
array getdate (int timestamp)
```

Ее аргументом является метка времени, а возвращаемым значением — ассоциативный массив, содержащий компоненты даты и времени, как показано в табл. 18.2.

Табл.18.2 Ассоциативный массив пар ключ-значение, возвращаемый функцией getdate()

Ключ	Значение
seconds	Секунды, числовое значение
minutes	Минуты, числовое значение
hours	Часы, числовое значение
mday	День месяца, числовое значение
wday	День недели, числовое значение
топ	Месяц, числовое значение
year	Год, числовое значение
yday	День года, числовое значение
weekday	День недели, полнотекстовый формат
month	Месяц, полнотекстовый формат

Проверка правильности дат

Для проверки правильности дат можно воспользоваться функцией **checkdate().** Это особенно полезно при проверке дат, вводимых пользователем. Функция checkdate() имеет следующий прототип:

```
int checkdate (int month, int day, int year)
```

Она проверяет, является ли год целым числом от 0 до 32767, месяц — целым от 1 до 12, и что указанное число существует в указанном месяце. Эта функция учитывает и високосные года.

Например,

```
checkdate(9, 18, 1972);
```

вернет значение true, а

```
checkdate(9, 31, 2000)
```

нет.

Преобразования даты в форматах PHP и MySQL

Дата и время в MySQL имеют несколько иной формат, чем можно было бы ожидать. Формат времени практически не требует изменений, но в дате в MySQL на первом месте стоит год. Например, дата 31th of July 2001 может вводиться как 2001-07-31 или 01-07-31. Даты, получаемые из MySQL, имеют этот формат по умолчанию.

Следовательно, взаимодействие PHP и MySQL обычно требует некоторого преобразования даты. Его можно выполнить с любой стороны.

При пересылке дат из PHP в MySQL их можно легко преобразовать в требуемый формат при помощи функции **date(),** как было показано ранее. Небольшое предостережение заключается в том, что следует использовать версию числа и месяца с ведущими нулями во избежание путаницы в MySQL.

Если же преобразование необходимо выполнить в MySQL, для этого существуют две полезных функции **DATE_FORMAT() и UNIX_TIMESTAMP().**

Функция **DATE_FORMAT()** работает аналогично подобной функции в PHP, но использует другие коды формата. Чаще всего она применяется для вывода даты в формате MM-DD-YYYY (месяц/число/год) вместо естественного для MySQL формата

YYYY-MM-DD (год/месяц/число). Для этого необходимо сформировать следующий запрос:

```
SELECT DATE_FORMAT(date_column, '%m %d %Y')
FROM tablename;
```

Код формата %m отвечает месяцу в двузначном формате, %d — дню в двузначном формате, %Y — году в четырехзначном формате. Наиболее полезные коды формата MySQL для преобразования даты перечислены в табл. 18.3.

Таблица 18.3 Коды формата MySQL-функции DATE_FORMAT()

Кол	Описание
%M	Месяц, полнотекстовый формат
%W	День недели, полнотекстовый формат
%D	День месяца, численный формат с текстовым суффиксом (например, 1st)
%Y	Год, 4-значное число
%y	Год, 2-значное число
%a	День недели, трехсимвольный формат
%d	День месяца, число с ведущими нулями
%e	День месяца, число без ведущих нулей
%m	Месяц, число с ведущими нулями
%c	Месяц, число без ведущих нулей
%b	Месяц, 3-символьное текстовое значение
%j	День года, числовое значение
%H	Часы в 24-часовом формате с ведущими нулями
%k	Часы в 24-часовом формате без ведущих нулей
%h или %I	Часы в 12-часовом формате с ведущими нулями
%l	Часы в 12-часовом формате без ведущих нулей
%i	Минуты, число с ведущими нулями
%r	Время в 12-часовом формате (hh:mm:ss [AM|PM])
%T	Время в 24-часовом формате (hh:mm:ss)
%S или %s	Секунды, число с ведущими нулями
%p	AM или PM
%w	День недели, число от 0 (воскресенье) до 6 (суббота)

Функция UNIX_TIMESTAMP работает аналогично, но преобразует значение столбца в метку времени UNIX. Например,

```
SELECT UNIX_TIMESTAMP(date_column)
FROM tablename;
```

возвращает дату в формате метки времени UNIX. Затем в PHP с ней можно производить любые операции.

На практике рекомендуется использовать временную отметку UNIX при операциях с датой, а стандартный формат — для хранения или вывода дат. Арифметические действия и сравнения проще выполнять над меткой времени UNIX.

Операции над датами

Наиболее простой способ вычислить период времени между двумя датами в PHP предполагает получение разности между двумя метками времени UNIX. Такой подход продемонстрирован в листинге 18.1.

Листинг 18.1 calc_age.php — сценарий для вычисления возраста по дате рождения

```
<?
  // установить дату для расчетов
  $day = 18;
  $month = 9;
  $year = 1972;

  // дата рождения требуется в формате день/месяц/год
  $bdayunix = mktime ("", "", "", $month, $day, $year); //вычислить метку
                                                        //времени UNIX для даты рождения
  $nowunix = time(); //вычислить метку времени UNIX для текущей даты
  $ageunix = $nowunix - $bdayunix; //вычислить разность
  $age = floor($ageunix / (365 * 24 * 60 * 60)); // преобразовать из
                                                 // секунд в годы
  echo "Age is $age";
?>
```

В этом сценарии дата для подсчета возраста была установлена внутри сценария. В реальном приложении подобного рода информация обычно поступает из HTML-формы.

Сценарий начинается с обращения к функции **mktime()** с целью вычисления меток времени для даты рождения и текущей даты:

```
$bdayunix = mktime ("", "", "", $month, $day, $year);
$nowunix = mktime(); // вычислить метку времени UNIX для текущей даты
```

После этого обе даты имеют одинаковый формат, что позволяет вычислить их разность:

```
$ageunix = $nowunix - $bdayunix;
```

Далее следует более сложный фрагмент кода — обратное преобразование периода времени в единицы измерения, естественные для человека. Это уже не метка времени, а возраст, измеренный в секундах. Преобразовать его в годы можно, разделив на количество секунд в году. После этого происходит округление при помощи функции **floor(),** поскольку возраст человека составляет, например, 20 лет только по прошествии двадцатого года от рождения:

```
$age = floor($ageunix / (365 * 24 * 60 * 60)); // преобразовать из
                                               // секунд в годы
```

Обратите внимание, что этот подход несколько некорректен, так как он ограничивается диапазоном значений меток времени UNIX (т.е. 32-разрядными целыми).

Использование календарных функций

В PHP содержится набор функций, позволяющих выполнять преобразования между различными календарными системами. Наиболее распространенными календарями являются Григорианский, Юлианский и Юлианский счет дней.

Григорианский календарь используется в большинстве западных стран. Дата 15 октября 1582 г. по Григорианскому календарю совпадает с датой *5* октября 1582 г. по Юлианскому календарю. До этого момента более распространенным был Юлианский календарь. Разные страны перешли на Григорианский календарь в различное время, некоторые — лишь в начале 20-го века.

Хотя, возможно, вы слышали об двух этих календарях, скорее всего, вы не слышали о Юлианском счете дней. Во многом он похож на метки времени UNIX. Это подсчет числа дней, начиная примерно с 4000 года до н.э. Сам по себе он не имеет применения, однако полезен при преобразованиях из одного формата в другой. Для этого дата сначала преобразуется в Юлианский счет дней (JD), а затем в требуемый формат календаря.

Для того чтобы воспользоваться этими функциями, необходимо скомпилировать календарное расширение вместе с PHP.

С целью ознакомления рассмотрим прототипы функций, используемых для преобразования из Григорианского календаря в Юлианский:

```
int gregoriantojd (int month, int day, int year)
string jdtojulian(int julianday)
```

Для преобразования даты необходимо вызвать обе эти функции:

```
$jd = gregoriantojd (9, 18, 1582);
echo jdtojulian($jd);
```

В результате будет выведена дата по Юлианскому календарю в формате mm/dd/yyyy (месяц/число/год).

Существует несколько разновидностей этих функций для преобразования даты в формат Григорианского, Юлианского, Французского и Еврейского календарей, а также метки времени UNIX.

Дополнительная информация

Если хотите узнать больше о функциях обработки даты и времени в PHP и MySQL, обратитесь к соответствующим разделам справочного руководства на Web-сайтах:

```
http://php.net/manual/ref.datetime.php

http://www.mysql.com/documentation/mysql/commented/
manual.php?section=Date_and_time_functions
```

Если требуется преобразовать дату в формат различных календарей, обратитесь к руководству по календарным функциям PHP:

```
http://php.net/manual/ref.calendar.php
```

Кроме того, полезной может оказаться следующая ссылка:

```
http://genealogy.org/~scottlee/cal-overview.html
```

Что дальше

Одним из уникальных и полезных свойств PHP является создание изображений на лету. В главе 19 рассматриваются способы применения библиотечных функций для получения интересных эффектов.

Глава 19

Создание изображений

Одно из полезных применений PHP заключается в создании изображений на лету. PHP имеет несколько встроенных функций, кроме того для создания новых или изменения существующих изображений можно воспользоваться библиотекой GD. В этой главе обсуждается, как с помощью функций обработки изображений можно добиться интересных и полезных эффектов.

В главе рассматриваются следующие вопросы:

- Настройка поддержки изображений в PHP
- Форматы изображений
- Создание изображений
- Использование текста и шрифтов для создания изображений
- Рисование фигур и построение графиков

Ниже приводятся два примера: создание кнопок Web-сайта на лету и отображение диаграммы с использованием числовых значений из базы данных MySQL.

Настройка поддержки изображений в PHP

Поддержка изображений в PHP осуществляется с помощью библиотеки gd, которую можно найти по адресу

```
http://www.boutell.com/gd/
```

Версия 1.6.2 этой библиотеки включена в состав PHP 4. По умолчанию поддерживается формат PNG. Если требуется поддержка JPEG, потребуется загрузить модуль jpeg-6b и перекомпилировать библиотеку GD с включенной поддержкой jpeg. Загрузить модуль можно из сайта

```
ftp://ftp.uu.net/graphics/jpeg/
```

Новая конфигурация PHP задается опцией

```
--with-jpeg-dir=/path/to/jpeg-6b
```

Затем, как обычно, необходимо заново выполнить компиляцию.

Возможность применять в изображениях TrueType-шрифты обеспечивается библиотекой FreeType, которая входит в состав PHP 4; ее также можно выгрузить с сайта

```
http://www.freetype.org/
```

Для использования вместо TrueType шрифтов PostScript Type 1 потребуется выгрузить библиотеку t1lib, размещенную на сайте

```
ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/software/t1lib/
```

После этого следует запустить конфигурационную программу PHP с опцией

```
--with-t1lib[=path/to/t1lib]
```

Форматы изображений

Библиотека GD поддерживает форматы JPEG, PNG и WBMP. Формат GIF больше не поддерживается. Рассмотрим кратко каждый из этих форматов.

JPEG

JPEG (произносится "джей-пег") означает *Joint Photographic Experts Group (Объединенная группа экспертов по фотографии)* и является названием стандарта. Формат файла, который обозначается как JPEG, в действительности называется JFIF, что отвечает одному из стандартов, выпушенных группой JPEG.

Если вы не знакомы с этим вопросом, то следует отметить, что формат JPEG обычно используется для хранения фотографических или подобных им изображений с большим количеством цветов или оттенков. В этом формате используется сжатие с потерями, т.е. при уменьшении размера файла теряется качество. Если JPEG-файлы содержат преимущественно аналоговые изображения с оттенками цветов, то человеческий глаз может не заметить потери качества. Этот формат не подходит для сохранения векторных изображений, текста или больших областей, заполненных одним цветом.

Прочесть о JPEG/JFIF можно на официальном сайте JPEG:

```
http://www.jpeg.org/public/jpeghomepage.htm
```

PNG

- PNG (произносится "пинг") обозначает *Portable Network Graphics (Переносимая сетевая графика)*. Этот формат файла рассматривается как замена формата *GIF (Graphics Interchange Format, формат обмена графическими изображениями)* по причинам, о которых будет рассказано далее. Web-сайт PNG описывает этот формат как "формат изображений со сжатием без потерь". Поскольку потерь не происходит, этот формат подходит для изображений, содержащих текст, прямые линии и блоки одного цвета, как, например, заголовки или кнопки на Web-сайте — все те элементы, для которых ранее использовался формат GIF.

В этом формате применяется более высокая степень сжатия, чем GIF, переменная прозрачность, гамма-коррекция и двойное чередование. Однако он не поддерживает анимацию — можно воспользоваться расширенным форматом MNG, однако последний еще находится на стадии разработки.

Прочесть о формате PNG можно на официальном сайте по адресу:

```
http://www.freesoftware.com/pub/png/
```

WBMP

WBMP означает *Wireless Bitmap (Битовое изображение для беспроводной связи)*. Этот файл был специально разработан для устройств беспроводной связи. Хотя библиотека GD поддерживает этот формат, в PHP в данный момент нет элементов для использования упомянутой функциональности.

GIF

GIF означает *Graphics Interchange Format (Формат обмена графическими изображениями)*. Это формат со сжатием без потерь, широко используемый в Internet для хранения изображений, содержащих текст, линии и одноцветные блоки.

Почему же GD не поддерживает формат GIF?

Ответ заключается в том, что поддержка существовала вплоть до версии 1.3. Если вместо функций PNG требуется установить и использовать функции GIF, можно загрузить библиотеку GD версии 1.3 из Web-сайта

```
http://www.linuxguruz.org/downloads/gd1.3.tar.gz
```

Обратите, однако, внимание, что разработчики GD не рекомендуют использовать эту версию и больше не поддерживают ее. Копия этой версии с поддержкой GIF может не оказаться доступной.

Существует немаловажная причина, почему GD больше не поддерживает формат GIF. В стандарте GIF применяется форма сжатия *LZW (Lempel Ziv Welch),* которая запатентована UNISYS. Разработчики программ, читающих и записывающих файлы в формате GIF, обязаны платить UNISYS лицензионные сборы. Так делает, скажем, компания Adobe, чьи продукты, например, Photoshop, используются для создания GIF-файлов. Библиотеки кодов находятся в ситуации, когда их авторы должны платить сборы, причем это распространяется и на пользователей. Таким образом, если вы на своем Web-сайте используете версию библиотеки GD с поддержкой GIF, то это значит, что вы должны заплатить UNISYS лицензионные сборы приличных размеров.

Это довольно-таки плачевная ситуация, поскольку формат GIF применялся на протяжении многих лет, пока компания UNISYS не решила залицензировать его. Фактически, этот формат стал одним из стандартов Internet. В сообществе разработчиков Internet существует немалая доля неприязни к подобного рода патентованию. Вы можете почитать об этом (и сформировать собственное мнение) на сайте UNISYS

```
http://www.unisys.com/unisys/lzw/
```

и его оппозиции Burn All Gifs ("Сожгите все GIF-файлы")

```
http://burnallgifs.org/
```

Авторы этой книги не являются адвокатами, поэтому ничего из сказанного нельзя воспринимать как советы юриста, но по их мнению проще всего использовать файлы в формате PNG независимо от лицензионной политики GIF.

Поддержка файлов PNG в браузерах постоянно улучшается, однако срок действия патента LZW истекает 19 июня 2003 г., поэтому окончательный результат еще неизвестен.

Создание изображений

Вот четыре основных шага по созданию изображений в PHP:

1. Создание холста, предназначенного для дальнейшей работы.
2. Вычерчивание форм или вывод текста в этом изображении.
3. Вывод окончательного рисунка.
4. Освобождение ресурсов.

Начнем с рассмотрения очень простого сценария создания изображения, приведенного в листинге 19.1.

Листинг 19.1 simplegraph.php **— выводит простой линейный рисунок с текстом** Sales **(Продажи)**

```
<?
// настройка изображения
   $height = 200;
   $width = 200;
   $im = ImageCreate($width, $height);
   $white = ImageColorAllocate ($im, 255, 255, 255);
   $black = ImageColorAllocate ($im, 0, 0, 0);

// отрисовка изображения
   ImageFill($im, 0, 0, $black);
   ImageLine($im, 0, 0, $width, $height, $white);
   ImageString($im, 4, 50, 150, "Sales", $white);

// вывод изображения
   Header ("Content-type: image/png");
   ImagePng ($im);

// освобождение ресурсов
   ImageDestroy($im);
?>
```

Результаты запуска сценария показаны на рис. 19.1.

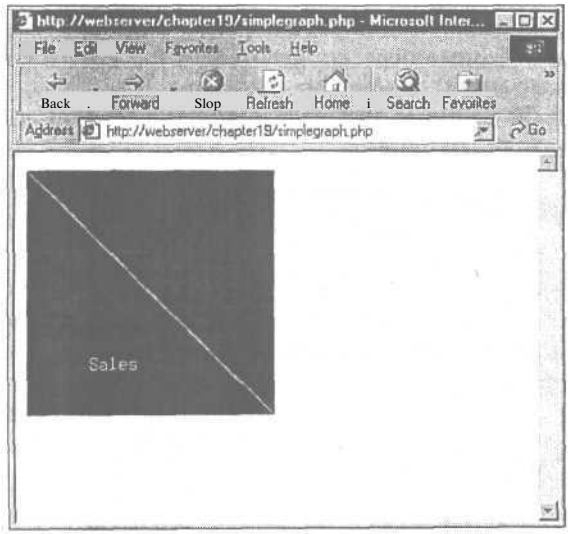


РИСУНОК 19.1
В сценарии вначале создается черный фон, а затем добавляется линия и текстовая метка.

Рассмотрим шаги создания изображения более подробно.

Создание холста

Для того чтобы приступить к созданию или изменению изображения в PHP, требуется создать идентификатор изображения. На то имеется два базовых способа. Первый заключается в создании пустого холста функцией ImageCreate(), что и сделано в сценарии:

```
$im = ImageCreate($width, $height);
```

В функцию **ImageCreate()** необходимо передать два аргумента. Первый из них является шириной нового изображения, а второй — высотой. Функция возвращает идентификатор нового изображения. (Идентификаторы во многом подобны дескрипторам файлов.)

Другой способ связан с чтением файла существующего изображения, после чего его можно подвергнуть фильтрации, изменению размеров или добавить что-либо. В зависимости от читаемого формата файла, это можно сделать при помощи функции **ImageCreateFromPNG()**, **ImageCreateFromJPEG()** или **ImageCreateFromGIF()**. Аргументом каждой из функций является имя файла, например,

```
$im = ImageCreateFromPNG("baseimage.png");
```

Пример, в котором существующее изображение используется для создания кнопок на лету, показан далее в этой главе.

Рисование или вывод текста в изображение

Рисование или вывод текста в изображение выполняется в два этапа.

Сначала необходимо выбрать цвета, которые будут использоваться при рисовании. Возможно, вы знаете, что цвета на мониторе компьютера представлены при помощи перемешивания красного, зеленого и синего. Форматы изображения используют цветовую палитру, которая состоит из указанных поднаборов всех возможных комбинаций трех цветов. Чтобы использовать цвет для рисования, его необходимо добавить к палитре изображения. Причем это следует делать для каждого цвета, даже для черного и белого.

Цвета для изображения можно выбрать, вызвав функцию **ImageColorAllocate()**. Ей необходимо передать идентификатор изображения и значения красной, зеленой и синей (RGB) компонент требуемого цвета.

В листинге 19.1 используются два цвета: черный и белый. Происходят следующие вызовы функций

```
$white = ImageColorAllocate ($im, 255, 255, 255);
$black = ImageColorAllocate ($im, 0, 0, 0) ;
```

Функция возвращает идентификатор цвета, который можно использовать для доступа к данному цвету в дальнейшем.

Далее, чтобы реально рисовать в изображении, существует несколько различных функций — в зависимости от требуемых форм — линии, дуги, многоугольники или текст.

Функции рисования используют следующие параметры:

- Идентификатор изображения
- Начальные и в ряде случаев конечные координаты изображаемого объекта

- Цвет
- Информация о шрифте для вывода текста

В сценарии использовались три функции. Рассмотрим их.

Вначале с помощью функции **ImageFill()** создается черный фон:

```
ImageFill($im, 0, 0, $black);
```

Аргументами этой функции являются идентификатор изображения, начальные координаты заполняемой области (x и y) и цвет заполнения.

ПРИМЕЧАНИЕ

Обратите внимание, что координаты точки в изображении отсчитываются от левого верхнего угла, координатами которого являются **x=0, y=0.** Правый нижний угол изображения имеет координаты **x=$width, y=$height.** Это несколько отличается от стандартных схем построения графиков, поэтому не забывайте об этом!

Затем левый верхний **(0,** 0) и правый нижний **($width, $height)** углы изображения соединяются линией:

```
ImageLine($im, 0, 0, $width, $height, $white);
```

Аргументами этой функции являются идентификатор изображения, начальные и конечные точки линии и цвет.

В заключение к изображению добавляется метка:

```
ImageString($im, 4, 50, 150, "Sales", $white);
```

Функция **ImageString()** принимает несколько иные аргументы. Ее прототип имеет вид:

```
int imagestring (int im, int font, int x, int y, string s, int col)
```

Параметрами являются идентификатор изображения, шрифт, координаты начальной точки текста x и y, непосредственно текст и цвет.

Шрифт задается числом от 1 до *5.* Этот диапазон отвечает набору встроенных шрифтов. Альтернативой этому является применение шрифтов TrueType или PostScript. Каждому из этих наборов шрифтов соответствует свой набор функций. В следующем примере будет продемонстрировано использование функций обработки шрифтов TrueType.

Одной из причин использования наборов функций других шрифтов является то, что шрифт текста, выводимого функцией **ImageStringO** и подобными ей, например, **ImageChar()** (вывод символа в изображение) является ступенчатым. Функции TrueType и PostScript воспроизводят текст, сглаживая шрифт.

Если вы незнакомы с разницей между ступенчатым и сглаженным шрифтом, внимательно ознакомьтесь с рис. 19.2.

РИСУНОК 19.2
Обычный текст имеет зазубренные края, особенно при большом размере шрифта. Сглаживание несколько выравнивает кривые и углы букв.



Там, где шрифт содержит кривые или наклонные линии, текст имеет зазубренные края. Так происходит потому, что кривая или наклонная линия достигается при помощи "эффекта лестницы". В сглаженном изображении кривые или наклонные линии содержат точки, промежуточные между цветом текста и фона. В результате текст выглядит гладким.

Вывод окончательного рисунка

Изображение можно вывести напрямую в браузер или же в файл.

В примере вывод изображения выполняется в браузер. Этот процесс состоит из двух шагов. Вначале необходимо сообщить Web-браузеру, что будет выводиться именно изображение, а не текст или HTML-код. Это достигается вызовом функции **Header(),** определяющей MIME-тип изображения:

```
Header ("Content-type: image/png");
```

Обычно при получении файла браузером MIME-тип — это первое, что отправляет Web-сервер. Для страницы в формате HTML или PHP (после выполнения кода), заголовок имеет вид

```
Content-type: text/html
```

Он сообщает браузеру, как необходимо интерпретировать последующие данные.

В данном случае требуется сообщить браузеру, что пересылается изображение, а не обычный вывод HTML. Этого можно добиться при помощи функции **Header()**, которая еще не обсуждалась.

Эта функция пересылает строки HTTP-заголовков. Другим типичным их применением является HTTP-перенаправление. Оно заставляет браузер загружать вместо запрашиваемой другую страницу. Так поступают в случае перемещения страницы. Например:

```
Header ("Location: http://www.domain.com/new_home_page.html");
```

Важно отметить, что функция **Header()** не может быть выполнена, если HTTP-заголовок уже был отправлен. PHP посылает HTTP-заголовки автоматически всякий раз, когда происходит вывод чего-либо в браузер. Следовательно, наличие любого оператора **echo** или даже просто пустого символа перед дескриптором PHP приводит к отправке заголовка, а значит, и выводе предупреждающего сообщения PHP при попытке вызова функции **Header().** Однако имеется возможность переслать несколько HTTP-заголовков и произвести несколько вызовов функции **Header()** в одном и том же сценарии, но все они должны появиться до первого вывода информации в браузер.

После отправки заголовка изображение выводится в результате вызова функции

```
ImagePng ($im);
```

В браузере выполняется вывод изображения в формате PNG. Если требуется другой формат, можно воспользоваться функцией **ImageJPEG()**, если включена поддержка JPEG, или **ImageGIF()**, если имеется более ранняя версия библиотеки GD. Конечно же, вначале следует отправить соответствующий заголовок, т.е.

```
Header ("Content-type: image/jpeg");
```

ИЛИ

```
Header ("Content-type: image/gif");
```

Вторая возможность, альтернативная всем предыдущим, заключается в выводе изображения в файл, а не браузер. Этого можно добиться, передавая необязательный второй параметр в функцию **ImagePNG()** (или подобную функцию для другого формата):

```
ImagePNG($im, $filename);
```

Помните, что при этом действуют все правила записи в файл из PHP (например, необходимость корректной настройки прав доступа).

Освобождение ресурсов

После того как работа с изображением завершена, необходимо очистить использованные ресурсы, уничтожив идентификатор изображения. Для этого следует вызвать функцию **ImageDestroy()**:

```
ImageDestroy($im);
```

Использование автоматически создаваемых изображений на других страницах

Поскольку заголовок может быть переслан лишь один раз и это единственный способ сообщить браузеру, что передается изображение, вставлять изображения, создаваемые на лету, в обычные страницы не особенно просто. Для этого существуют три следующих способа:

1. Вся страница может состоять из рисунка, как было показано в предыдущем примере.
2. Можно сохранить изображение в файле, как об этом упоминалось ранее, а затем ссылаться на него при помощи обычного тэга **<IMG>.**
3. В дескриптор изображения можно поместить сценарий, создающий изображение.

О методах 1 и 2 уже было рассказано. Рассмотрим вкратце метод 3.

В HTML-код следует поместить дескриптор изображения следующего вида:

```
<img src="simplegraph.php" height=200
   width=200 alt="Sales going down">
```

Вместо вставки рисунков PNG, JPEG или GIF напрямую, в дескрипторе в поле SRC указан сценарий, создающий изображение. Такое изображение будет получено и выведено браузером так же, как и обычное. Результат показан на рис. 19.3.

Использование текста и шрифтов при создании изображений

Рассмотрим более сложный пример. Зачастую очень удобно создавать кнопки или другие изображения для Web-сайта автоматически.



РИСУНОК 19.3 *Динамически созданный рисунок выглядит для конечного пользователя так же, как и обычный.*

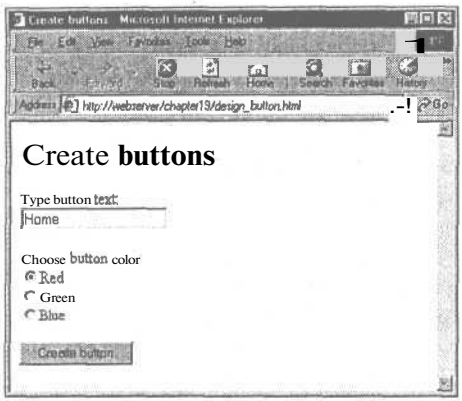


РИСУНОК 19.4 *Тестовый сценарий позволяет выбирать цвет кнопки и вводить текст.*

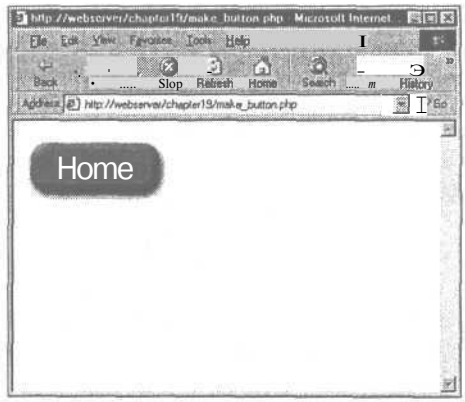


РИСУНОК 19.5 *Кнопка, созданная сценарием make_button.php.*

Простые кнопки, содержащие прямоугольник цвета фона, можно легко создать упомянутыми ранее методами.

Однако в данном примере будут созданы кнопки, использующие пустой шаблон, позволяющий создавать такие эффекты, как скошенные края, гораздо более простыми методами, нежели предлагают Photoshop, GIMP или другие графические редакторы. Используя графическую библиотеку PHP, мы начнем с базового изображения и дополним его.

Кроме того, используются шрифты TrueType, чтобы полученный текст был сглаженным. Функции работы с TrueType-шрифтами обладают своими причудами, о которых будет рассказываться далее.

Задача, в основном, состоит в том, чтобы имея некоторый текст, создать кнопку с этим текстом на ней. Текст следует центрировать по горизонтали и вертикали. Кроме того, необходимо подобрать максимально возможный размер шрифта, помещающийся в кнопку.

С целью проведения тестов и экспериментов для сценария, создающего кнопки, создан другой сценарий. Его интерфейс показан на рис. 19.4. (HTML-код для этой формы здесь не приведен, поскольку он очень прост; кроме того, его можно найти на CD-ROM в файле **design_button.html**.)

Интерфейс подобного типа применяется для автоматического создания Web-сайтов. Сценарий, представленный здесь, можно использовать для создания всех кнопок Web-сайта на лету!

Пример типичного вывода показан на рис. 19.5.

Кнопку генерирует сценарий **make_button.php**. Его текст приведен в листинге 19.2.

Листинг 19.2 make_button.php — этот сценарий можно вызвать из формы в файле **design_button.html** или же из HTML-дескриптора изображения

```
<?
// проверить, что в переменных содержатся требуемые данные
// переменными являются button-text и color

if (empty($button_text) || empty($color))
{
   echo "Could not create image - form not filled out correctly";
   exit;
}
```

```
// создать изображение с требуемым фоном и проверить его размер
$im = imagecreatefrompng ("$color-button.png");

$width_image = ImageSX($im) ;
$height_image = ImageSY($im);

// нашим изображениям требуется отступ 18 точек от края
$width_image_wo_margins = $width_image - (2 * 18);
$height_image_wo_margins = $height_image - (2 * 18) ;

// проверить, подходит ли размер шрифта, и уменьшать его до тех пор,
// пока не подойдет
// начать с наибольшего размера, который может подойти для кнопок
$font_size = 33;

do
{
   $font_size--;

   // найти размер текста при заданном размере шрифта
   $bbox=imagettfbbox ($font_size, 0, "arial.ttf", $button_text) ;

   $right_text = $bbox[2];    // правая координата
   $left_text = $bbox[0];     // левая координата
   $width_text = $right_text - $left_text;  // какова ширина надписи?
   $height_text = abs($bbox[7] - $bbox[1]);  // какова высота надписи?

} while ( $font_size>8 &&
            ( $height_text>$height image_wo_margins ||
              $width_text>$width_image_wo_margins )
          );

if ( $height_text>$height_image_wo_margins | |
      $width_text>$width_image_wo_margins )
{
   // невозможно подобрать шрифт для текста
   echo "Text given will not fit on button.<BR>" ;
}
else
{
   // найден подходящий размер шрифта
   // найти точку размещения текста

   $text_x = $width_image/2.0 - $width_text/2.0;
   $text_y = $height_image/2.0 - $height_text/2.0 ;

   if ($left_text < 0)
        $text x += abs($left_text);     // добавить коэффициент для
                                        // выступа слева

   $above_line_text = abs($bbox[7]) ;    // далеко ли до базовой линии?
   $text_y += $above_line_text;       // добавить коэффициент базовой линии

   $text_y -= 2;  // корректировка формы шаблона

   $white = ImageColorAllocate ($im, 255, 255, 255);

   ImageTTFText ($im, $font_size, 0, $text_x, $text_y, $white,
                 "arial.ttf", $button_text);

   Header ("Content-type: image/png");
   ImagePng ($im);
}
ImageDestroy ($im);
?>
```

Это один из самых длинных сценариев, приведенных в этой книге. Рассмотрим его последовательно по разделам. Он начинается с проверки ошибок, а затем производится настройка холста.

Настройка холста

В листинге 19.2 используется существующее изображение кнопки. Есть возможность выбрать кнопку одного из трех цветов: красный (red-button.png), зеленый (green-button.png) и синий (blue-button.png).

Цвет, выбранный пользователем, присваивается переменной **$color**.

Затем создается новый идентификатор изображения — выбранной пользователем кнопки:

```
$im = imagecreatefrompng ("$color-button.png");
```

Аргументом функции **ImageCreateFromPNG()** является имя файла в формате PNG, а возвращаемым значением — новый идентификатор. Обратите внимание, что этот вызов никак не изменяет исходный файл PNG. При включенной поддержке соответствующих форматов подобным образом можно использовать функции **ImageCreateFromJPEG()** и **ImageCreateFromGIF()**.

ПРИМЕЧАНИЕ

Функция **ImageCreateFromPNG()** создает изображение в памяти. Чтобы сохранить изображение в файл или вывести его в браузер, следует обратиться к функции **ImagePNG()**. О ней будет рассказано далее.

Подбор размера текста на кнопке

В переменной **$button_text** хранится текст, введенный пользователем. Этот текст необходимо вывести на кнопке шрифтом максимально допустимого размера. Задача решается с помощью итераций или, строго говоря, итеративным методом проб и ошибок.

Сначала устанавливаются все требуемые переменные. Первые две хранят высоту и ширину кнопки:

```
$width_image = ImageSX($im);
$height_image = ImageSY($im);
```

Две следующих переменных хранят отступы от краев кнопки. Кнопка является скошенной, поэтому необходимо оставить пустым некоторое пространство возле края. Если вы будете использовать другие изображения, то эти параметры будут другими! В данном случае отступ от каждого края составляет 18 точек.

```
$width_image_wo_margins = $width_image - (2 * 18) ;
$height_image_wo_margins = $height_image - (2 * 18) ;
```

Далее устанавливается начальный размер шрифта. Его значение равно 32 (фактически, 33, но затем происходит уменьшение на единицу), поскольку это шрифт наибольшего размера, который может поместиться на кнопке:

```
$font_size = 33;
```

Затем выполняется цикл, в котором размер шрифта уменьшается при каждой итерации до тех пор, пока заданный текст не поместится на кнопке:

```
do
{
   $font_size--;

   // найти размер текста при заданном размере шрифта
   $bbox=imagettfbbox ($font_size, 0, "arial.ttf", $button_text);

   $right_text = $bbox[2];    // правая координата
   $left_text = $bbox[0];     // левая координата
   $width_text = $right_text - $left_text;  // какова ширина надписи
   $height_text = abs($bbox[7] - $bbox[1]);  // какова высота надписи

} while ( $font_size>8 &&
          ( $height_text>$height_image_wo_margins ||
            $width_text>$width_image_wo_margins )
        );
```

Этот фрагмент кода проверяет размер шрифта, используя *ограничивающий прямоугольник* текста. Его размеры возвращает функция **ImageGetTTFBBox()**, входящая в состав набора функций работы с TrueType-шрифтами. После определения размера шрифта надпись выводится шрифтом TrueType при помощи функции **ImageTTFText()**.

Ограничивающий прямоугольник текста — это наименьший прямоугольник, в который можно вписать текст. Пример такого прямоугольника показан на рис. 19.6.



РИСУНОК 19.6 *Координаты ограничивающего прямоугольника задаются относительно базовой линии. Начало координат обозначено как (0,0).*

Для определения размеров прямоугольника вызывается функция

```
$bbox = imagettfbbox ($font_size, 0, "arial.ttf", $button_text);
```

Такой код гласит примерно следующее: "Вывести размеры текста **$button_text**, имеющего наклон нуль градусов, набранного шрифтом TrueType Arial размером **$font_size**".

Отметьте, что фактически функции необходимо передать путь к файлу, содержащему шрифт. В данном случае он находится в том же каталоге, что и сценарий (каталоге по умолчанию), поэтому путь здесь не указывается.

Функция возвращает массив, содержащий координаты углов ограничивающего прямоугольника. Содержимое этого массива представлено в табл. 19.1.

Таблица 19.1 Содержимое массива ограничивающего прямоугольника

Индекс *массива*	*Содержимое*
0	Координата X левого нижнего угла
1	Координата Y левого нижнего угла
2	Координата X правого нижнего угла
3	Координата Y правого нижнего угла
4	Координата X правого верхнего угла
5	Координата Y правого верхнего угла
6	Координата X левого верхнего угла
7	Координата Y левого верхнего угла

Чтобы запомнить, как расположены координаты в массиве, достаточно знать, что нумерация начинается с левого нижнего угла и продолжается против часовой стрелки.

Значения, возвращаемые функцией **ImageTTFBBox()**, имеют одно неочевидное свойство. Они являются координатами, отсчитываемыми от некоторого начала. Однако, в отличие от координат изображения, центром для которых служит левый верхний угол, они отсчитываются от базовой линии.

Вернемся вновь к рис. 19.6. Линия, соприкасающаяся с большинством букв снизу, и называется *базовой линией*. Части некоторых букв оказываются под ней, как "у" в примере. Они называются *свисающими элементами букв (descenders)*.

Левая точка базовой линии является началом координат (0,0). Координата Y над базовой линией имеет положительное значение, а под ней — отрицательное.

Кроме того, текст может выступать за пределы ограничивающего прямоугольника. Например, начало текста может, в действительности, иметь координату X, равную -1.

Все это значит, что при выполнении операций над координатами требуется проявить некоторую осторожность.

Определение ширины и высоты текста происходит следующим образом:

```
$right_text = $bbox[2];        // правая координата
$left_text = $bbox[0];         // левая координата
$width_text = $right_text — $left_text;   // какова ширина надписи?
$height_text = abs($bbox[7] — $bbox[1]); // какова высота надписи?
```

После этого проверяется условие цикла:

```
> while ( $font_size>8 &&
            ( $height_text>$height_image_wo_margins ||
              $width text>$width_image_wo_margins )
        );
```

Здесь есть два набора условий. Первый: шрифт все еще является читаемым, поскольку не имеет смысла выводить надпись шрифтом, размер которого меньше 8.

Второй: помещается ли текст внутри отведенной для него области.

Затем, если в результате действия итеративной процедуры не удалось подобрать подходящий размер шрифта, выводится сообщение об ошибке:

```
if ( $height_text>$height_image_wo_margins ||
     $width_text>$width image_wo margins )
{
  // невозможно подобрать шрифт для текста
  echo "Text given will not fit on button.<BR>";
}
```

Позиционирование текста

Если предыдущая часть сценария выполнилась успешно, потребуется определить точку, в которой будет начало текста. Это центр прямоугольника, отведенного под текст.

```
$text_x = $width_image/2.0 - $width_text/2.0;
$text_y = $height_image/2.0 - $height_text/2.0 ;
```

Из-за сложностей, связанных с координатной системой базовой линии, необходимо добавить корректирующие коэффициенты:

```
if ($left_text < 0)
  $text_x += abs($left_text);//добавить коэффициент для выступа слева
$above_line_text = abs($bbox[7]); //далеко ли до базовой линии?
$text_y += $above_line_text;      //добавить коэффициент базовой линии
$text_y -= 2;                     //корректировка формы шаблона
```

Корректирующие коэффициенты позволяют немного выровнять изображение, которое слегка "перегружено сверху".

Вывод текста на кнопку

Теперь остается лишь вывести текст. Для него требуется белый цвет, поэтому:

```
$white = ImageColorAllocate ($im, 255, 255, 255);
```

Затем для вывода текста используется функция **ImageTTFText()**:

```
ImageTTFText ($im, $font_size, 0, $text_x, $text_y, $white,
   "arial.ttf", $button_text);
```

Эта функция требует нескольких параметров: идентификатор изображения, размер шрифта в точках, угол наклона текста, координаты X и Y начальной точки, цвет, файл шрифта и, в заключение, собственно текст.

ПРИМЕЧАНИЕ

Файл шрифта должен быть доступен на сервере, однако на клиентской машине его присутствия не требуется, поскольку к пользователю поступает уже готовое изображение. По умолчанию функция ищет шрифт в том же каталоге, из которого запущен сценарий. Конечно же, можно указать и полный путь к этому шрифту.

Заключительные действия

Теперь изображение с кнопкой выводится в браузер пользователя:

```
Header ("Content-type: image/png");
ImagePng ($im);
```

После этого выполняется освобождение ресурсов:

```
ImageDestroy ($im);
```

Вот и все! Если все в порядке, в окне браузера отображается кнопка, показанная на рис. 19.5.

Рисование фигур и построение графиков

В последнем приложении рассматривалась обработка существующих изображений и текста. Здесь на примере будет рассказано об изображении фигур.

В данном примере создается система голосования, размещенная на Web-сайте. Результаты будут сохраняться в базе данных MySQL, и по ним с помощью PHP-функций будет строиться гистограмма.

Одно из применений функций работы с изображениями — вычерчивание графиков. Данными могут быть объемы продаж, количество посещений Web-сайта и вообще все, что угодно.

Для этого примера необходимо создать базу данных MySQL с именем **poll.** Она содержит одну таблицу **poll_results,** состоящую из двух колонок: **candidate** — имена кандидатов и **num_votes** — количество голосов. Для доступа к базе данных создается пользователь с именем **poll** и паролем **poll.** Для настройки требуется не более пяти минут – достаточно запустить SQL-сценарий, приведенный в листинге 19.3. Можно просто перенаправить этот сценарий клиенту **mysql**, зарегистрировавшись как root:

```
mysql -u root -p < pollsetup.sql
```

Конечно же, можно воспользоваться учетной записью любого пользователя, имеющего необходимые права доступа MySQL.

Листинг 19.3 pollsetup.sql — настройка базы данных poll

```
create database poll;
use poll;
create table poll_results (
   candidate varchar(30),
   num_votes int
);
insert into poll_results values
   ('John Smith', 0),
   ('Mary Jones', 0),
   ('Fred Bloggs', 0)
;
grant all privileges
on poll.*
to poll@localhost
identified by 'poll';
```

База данных содержит информацию о трех кандидатах. Интерфейс для голосования обеспечивается страницей **vote.html,** код которой приведен в листинге 19.4.

Листинг 19.4 vote.html — здесь пользователи могут проголосовать

```
<html>
<head>
   <title>Polling</title>
<head>
<body>
<h1>Pop Poll</h1>
<p>Who will you vote for in the election?</p>
<form method=post action="show_poll.php">
<input type=radio name=vote value="John Smith">John Smith<br>
<input type=radio name=vote value="Mary Jones">Mary Jones<br>
<input type=radio name=vote value="Fred Bloggs">Fred Bloggs<br><br>
<input type=submit value="Show results">
</form>
</body>
```

Вид страницы в браузере показан на рис. 19.7.

Когда пользователь нажимает кнопку submit, его голос добавляется в базу данных, затем оттуда читаются все голоса и выводится гистограмма текущего состояния результатов.

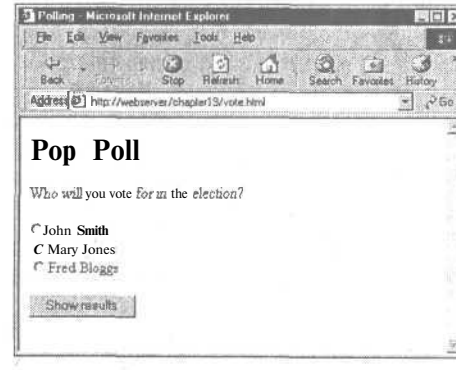


РИСУНОК 19.7
На этой странице пользователи могут проголосовать, а нажатие кнопки submit (отправить) выводит на экран текущее состояние результатов голосования.

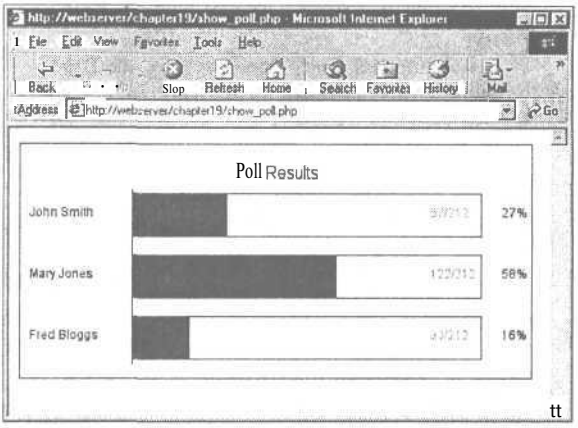


РИСУНОК 19.8
Результаты голосования создаются путем отрисовки на холсте набора линий, прямоугольников и текстовых элементов.

Пример гистограммы после 212 голосований показан на рис. 19.8.

Сценарий, создающий это изображение, достаточно объемен. Поэтому он разбит на четыре части, которые обсуждаются отдельно.

Большая часть сценария должна выглядеть знакомой, поскольку ранее приводилось достаточно много похожих примеров работы с MySQL, а также создания одноцветного холста и печати на нем текстовых меток.

Новыми в сценарии являются фрагменты, отвечающие за рисование линий и прямоугольников. Внимание прежде всего будет сосредоточено на них. Часть 1 (сценария из четырех частей) приведена в листинге 19.5.1.

Листинг 19.5.1 showpoll.php — часть 1 обновляет базу данных и запрашивает из нее новое состояние результатов

```
<?
/*****************************************************************
   Запрос базы данных для получения информации о голосовании
*****************************************************************/
// соединение с базой данных и регистрация
if (!$db_conn = @mysql_connect("localhost", "poll", "poll"))
{
   echo "Could not connect to db<br>";
   exit;
};
@mysql_select_db("poll");

if (!empty($vote))   // если форма заполнена, добавить голос
{
   $vote = addslashes($vote);
   $query = "update poll_results
             set num_votes = num_votes + 1
             where candidate = '$vote'";
   if(!($result = @mysql_query($query, $db_conn)))
   {
      echo "Could not connect to db<br>";
      exit;
   }
};

// запросить текущие результаты голосования независимо от того,
// проголосовал ли данный пользователь
$query = "select * from poll_results";
```

```
if(! ($result = @mysql_query($query, $db_conn)))
{
   echo "Could not connect to db<br>";
   exit;
}
$num_candidates = mysql_num_rows($result);
// подсчитать общее количество голосов
$total_votes=0;
while ($row = mysql_fetch_object ($result))
{
     $total_votes += $row->num_votes;
}
mysql_data_seek($result, 0);  // обнулить указатель на результаты
```

Часть 1, приведенная в листинге 19.5.1, соединяется с базой данных MySQL, обновляет данные на основе пользовательского ввода и запрашивает новое состояние результатов. Обладая этой информацией, можно произвести расчеты, необходимые для построения графика. Часть 2 сценария показана в листинге 19.5.2.

Листинг 19.5.2 showpoll.php — часть 2 отвечает за настройку всех переменных, необходимых для рисования

```
/*****************************************
  Начальные расчеты для построения графика
*****************************************/
// установка значений констант
$width=500;          // ширина изображения в точках — рисунок поместится
                     // в браузере при экранном разрешении 640x480
$left_margin = 50;   // пространство слева от изображения
$right_margin = 50; // пространство справа от изображения
$bar_height = 40;
$bar_spacing = $bar_height/2;
$font = "arial.ttf";
$title_size = 16;    // размер в точках
$main_size = 12;     // размер в точках
$small_size = 12;    // размер в точках
$text_indent = 10;   // положение текстовых меток слева

// установка начальной точки для рисования
$x = $left_margin + 60;   // местоположение базовой линии рисунка
$y = 50;
$bar_unit = ($width-($x+$right_margin)) / 100;    // одна "точка" на
                                                  // гистограмме

// подсчитать высоту прямоугольников плюс некоторый отступ
$height = $num_candidates * ($bar_height + $bar_spacing) + 50;
```

В части 2 устанавливаются переменные, необходимые для изображения графика.

Определение значений этих переменных несколько утомительно, однако предусмотрительность в том, как должно выглядеть окончательное изображение, сделает процесс рисования более простым. Значения, используемые в сценарии, были получены при разработке макета на бумаге и приближенном вычислении требуемых пропорций.

Переменная **$width** содержит ширину изображения, **$left_margin** и **$right_margin** – отступ слева и справа соответственно, **$bar_height** и **$bar_spacing** — "толщину" и расстояние между полосками, **$font, $title_size, $main_size, $small_size** и **$text_indent** — шрифт, его размеры и положение меток.

Эти базовые значения позволяют рассчитать остальные. Вначале необходимо нарисовать базовую линию, от которой будут начинаться все прямоугольники. Ее положение задается отступом слева и длиной текстовых меток (координата X), а также приближенными оценками макета (координата Y).

Далее определяются два важных значения: первое — расстояние на гистограмме, изображающее единицу:

```
$bar_unit = ($width-($x+$right_margin)) / 100;    // одна "точка"
                                                  // на гистограмме
```

Это максимальная длина полоски — от базовой линии до отступа справа — деленная на 100, так как график отображает значения в процентах.

Второе значение — полная высота изображения:

```
$height = $num_candidates * ($bar_height + $bar_spacing) + 50;
```

Это высота одной полоски, умноженная на их количество, плюс расстояние, оставленное под заголовок. Часть 3 сценария приведена в листинге 19.5.3.

Листинг 19.5.3 showpoll.php — часть 3 готовит изображение к выводу всех графических элементов

```
/*****************************************
  Настройка опорного изображения
*****************************************/
// создать пустой холст
$im = imagecreate($width,$height);

// выбрать цвета
$white=ImageColorAllocate($im,255,255,255);
$blue=ImageColorAllocate($im,0,64,128);
$black=ImageColorAllocate($im,0,0,0);
$pink = ImageColorAllocate($im,255,78,243);

$text_color = $black;
$percent_color = $black;
$bg_color = $white;
$line_color = $black;
$bar_color = $blue;
$number_color = $pink;

// залить холст цветом фона
ImageFilledRectangle($im,0,0,$width,$height,$bg_color);

// контур холста
ImageRectangle($im,0,0,$width-1,$height-1,$line_color);

// вывести заголовок
$title = "Poll Results";
$title_dimensions = ImageTTFBBox($title_size, 0, $font, $title);
$title_length = $title_dimensions[2] - $title_dimensions[0];
$title_height = abs($title_dimensions[7] - $title_dimensions[1]);
$title_above_line = abs($title_dimensions[7]);
$title_x = ($width-$title_length)/2;   // центрировать по x
$title_y = ($y - $title_height)/2 + $title_above_line; // центрировать
                        // по y в области, оставленной для заголовка
ImageTTFText($im, $title_size, 0, $title_x, $title_y,
             $text_color, $font, $title);

// начертить базовую линию, начиная чуть выше первой полоски
// и завершая чуть ниже последней
ImageLine($im, $x, $y-5, $x, $height-15,  $line_color);
```

В части 3 происходит подготовка опорного изображения, выбор цветов и вывод части графика.

На этот раз фон заполняется цветом следующим образом

```
ImageFilledRectangle($im,0,0,$width,$height,$bg_color);
```

Функция **ImageFilledRectangle()**, как можно предположить по названию, отображает заполненный прямоугольник. Как всегда, первый параметр — идентификатор изображения. За ним следуют координаты X и Y начальной и конечной точек. Ими являются, соответственно, левый верхний и правый нижний углы прямоугольника. В данном случае весь холст заполняется цветом фона — белым, который является последним аргументом функции.

Затем происходит вызов функции

```
ImageRectangle($im,0,0,$width-1,$height-1,$line_color);
```

для отображения контура по краям холста. Функция выводит незаполненный прямоугольник и имеет такие же аргументы. Обратите внимание, что конечная точка имеет координаты **$width-1** и **$height-1**. Если бы они были равны **$width** и **$height**, прямоугольник оказался бы за пределами холста.

Для центрирования и вывода заголовка гистограммы применяется та же логика и те же функции, что и в предыдущем сценарии.

В заключение, выводится базовая линия:

```
ImageLine($im, $x, $y-5, $x, $height-15, $line_color);
```

Эта функция отображает линию в изображении **$im**, начиная от точки ($x, $y-5) и заканчивая в точке **($x, $height-15),** цветом **$line_color**.

Базовая линия начинается несколько выше первой полоски и заканчивается ниже последней.

Теперь все готово к выводу данных. Часть 4 приведена на листинге 19.5.4.

Листинг 19.5.4 showpoll.php — часть 4 выводит на графике непосредственно данные и производит заключительные операции

```
/*****************************************
  Вывести данные на графике
*****************************************/
// Запросить данные для каждого кандидата и отобразить
// соответствующую полоску
while ($row = mysql_fetch_object ($result))
{
  if ($total_votes > 0)
    $percent = intval(round(($row->num_votes/$total_votes)*100));
  else
    $percent = 0;

  // вывести проценты
  ImageTTFText($im, $main_size, 0, $width-30, $y+($bar_height/2),
               $percent_color, $font, $percent."%");
  if ($total_votes > 0)
    $right_value = intval(round(($row->num_votes/$total_votes)*100));
  else
    $right_value = 0;

  // длина полоски для данного значения
  $bar_length = $x + ($right_value * $bar_unit);

  // нарисовать полоску для данного значения
```

```
    ImageFilledRectangle($im, $x, $y-2, $bar_length, $y+$bar_height,
    $bar_color);

    // вывести заголовок для данного значения
    ImageTTFText($im, $main_size, 0, $text_indent, $y+($bar_height/2),
                 $text_color, $font, "$row->candidate");

    // нарисовать контур, отвечающий значению 100%
    ImageRectangle($im, $bar_length+1, $y-2,
                   ($x+(100*$bar_unit)), $y+$bar_height, $line_color);

    // вывести числа
    ImageTTFText($im, $small_size, 0, $x+(100*$bar_unit)-50,
    $y+($bar_height/2),
                 $number_color, $font, $row->num_votes."/".$total_votes);

    // перейти к следующей полоске
    $y=$y+($bar_height+$bar_spacing);
}
/*****************************************
  Вывести изображение
*****************************************/
Header("Content-type:  image/png");
ImagePng($im);

/*****************************************
  Освободить ресурсы
*****************************************/
ImageDestroy ($im);
?>
```

В части 4 из базы данных запрашиваются данные для каждого кандидата, вычисляется процент голосов, а затем выводятся прямоугольники и текстовые метки.

Как и ранее, текст выводится при помощи функции **ImageTTFText()**. Заполненные прямоугольники отображаются функцией **ImageFilledRectangle()**:

```
ImageFilledRectangle($im, $x, $y-2, $bar_length, $y+$bar_height,
   $bar_color);
```

Контур, отвечающий 100%, отображается функцией **ImageRectangle()**:

```
ImageRectangle($im, $bar_length+1, $y-2,
                   ($x+(100*$bar_unit)), $y+$bar_height, $line_color);
```

После вывода всех прямоугольников изображение пересылается в браузер функцией **ImagePNG()**, а затем функция **ImageDestroy()** освобождает соответствующие ресурсы.

Это объемный сценарий, но его несложно адаптировать под свои нужды. Важным свойством сценария, является, к сожалению, то, что он лишен "противообманного" механизма. Пользователи смогут очень быстро обнаружить, что можно голосовать повторно, а это делает результаты бессмысленными.

Если вы достаточно владеете математикой, то можете воспользоваться таким подходом для вывода линейных графиков или даже секторных диаграмм.

Другие функции обработки изображений

Кроме функций, рассмотренных в этой главе, существуют и такие, которые позволяют выводить кривые (**ImageArc()**) и многоугольники (**ImagePolygon()**), а также вариации уже изученных функций. Создание любого изображения следует начинать с макета на бумаге. Затем всегда стоит обратиться к руководству, поскольку там могут содержаться дополнительные функции, полезные в конкретном случае.

Дополнительная информация

Немало полезных материалов доступно в Internet. Если вы испытываете затруднения при использовании функций работы с изображениями, стоит обратиться к документации по библиотеке GD, так как PHP-функции являются лишь оболочками для данной библиотеки. Документация по GD находится на сайте

```
http://www.boutell.com/gd/
```

Существуют также замечательные обучающие курсы по определенным типам графических приложений на сайтах Zend и Devshed:

```
http://www.zend.com
http://devshed.com
```

Идеи приложения, создающего гистограмму, основаны на сценарии Стива Маранды (Steve Maranda), доступном на сайте Devshed.

Что дальше

В следующей главе рассматривается технология управления сеансами — предельно удобная функциональность, появившаяся в PHP 4.

Глава 20

Управление сеансами в PHP

В этой главе вы познакомитесь с функциональными средствами управления сеансами в PHP 4.

В главе рассматриваются следующие вопросы:

- Что такое управление сеансом
- cookie-наборы
- Открытие сеанса
- Переменные сеанса
- Сеансы и аутентификация

Что такое управление сеансом

Возможно, вам доводилось слышать, что HTTP иногда называют "протоколом без состояния". Это означает, что данный протокол не имеет встроенного способа поддержки состояния между двумя транзакциями. Когда пользователь запрашивает друг за другом две страницы, HTTP не обеспечивает возможности уведомить, что оба запроса исходят от одного и того же пользователя.

Таким образом, идея управления сеансами заключается в обеспечении отслеживания пользователя в течение одного сеанса связи с Web-сайтом.

Если это удастся осуществить, мы сможем легко поддерживать подключение пользователя и предоставление ему содержимого сайта в соответствии с его уровнем прав доступа или персональными настройками. Мы сумеем отслеживать поведение пользователя. Кроме того, мы сможем реализовать покупательские тележки (shopping carts).

В более ранних версиях PHP управление сеансами осуществлялось средствами PHPLib, базовой библиотеки PHP, которая и сейчас является полезным набором инструментов. Об этом можно прочесть на

`http://phplib.netuse.de/index.php3`

Четвертая версия PHP включает собственные встроенные функции управления сеансом. Концептуально они подобны PHPLib, но PHPLib помимо этого обеспечивает еще и ряд дополнительных функциональных возможностей. Так что если окажется, что эти собственные функции не вполне отвечают вашим требованиям, ничто не мешает рассмотреть возможность использования PHPLib.

Основные функциональные средства управления сеансом

Для запуска сеанса в PHP используется уникальный идентификатор сеанса, представляющий собой зашифрованное случайное число. Идентификатор сеанса генерируется PHP и сохраняется на стороне клиента в течение всего времени жизни сеанса. Для хранения идентификатора сеанса используется либо cookie-набор на компьютере пользователя, либо URL.

Идентификатор сеанса играет роль ключа, обеспечивающего возможность регистрации некоторых специфических переменных в качестве так называемых переменных сеанса. Содержимое этих переменных сохраняется на сервере. Единственной информацией, "видимой" на стороне клиента, является идентификатор сеанса. Если во время определенного подключения к вашему сайту идентификатор сеанса является "видимым" либо в cookie-наборе, либо в URL, имеется возможность получить доступ к переменным сеанса, которые сохранены на сервере для данного сеанса. По умолчанию переменные сеанса хранятся в двумерных файлах на сервере (при желании способ хранения можно изменить и использовать вместо двумерного файла базу данных, но для этого потребуется написать собственную функцию — более подробно об этом читайте в разделе "Конфигурация управления сеансами").

Скорее всего, придется иметь дело с Web-сайтами, на которых для хранения идентификатора сеанса используется URL. Если в вашем URL имеется строка данных, которые выглядят случайными, то это, скорее всего, свидетельствует об использовании одной из двух описанных здесь разновидностей управления сеансом.

Другим решением проблемы сохранения состояния на протяжении некоторого количества транзакций, при наличии чистого внешнего вида URL, являются cookie-наборы.

Что такое cookie-набор?

cookie-набор — это небольшой фрагмент информации, который сценарии сохраняют на клиентской машине. Чтобы установить cookie-набор на машине пользователя, необходимо отправить ему HTTP-заголовок, содержащий данные в следующем формате:

```
Set-Cookie: NAME=VALUE; [expires=DATE;] [path=PATH;]
  [domain=DOMAIN_NAME;] [secure]
```

Это создаст cookie-набор с именем NAME и значением **VALUE.** Все остальные параметры являются необязательными. В **expires** задается дата истечения срока действия, после наступления которой cookie-набор перестанет рассматриваться как актуальный (заметим, что если дата истечения срока действия не задана, cookie-набор будет постоянным, пока его кто-нибудь не удалит вручную — либо вы, либо сам пользователь). Два параметра **path и domain** применяются для определения одного или нескольких URL, к которым относится данный cookie-набор. Ключевое слово **secure** означает, что cookie-набор не может отправляться через простое HTTP-соединение.

Когда браузер соединяется с URL, он сначала ищет cookie-наборы, хранящиеся локально. Если какие-либо из них относятся к URL, с которым установлено соединение, они передаются обратно на сервер.

Установка cookie-наборов из PHP

cookie-наборы в PHP можно установить вручную, используя функцию **setcookie()**. Она имеет следующий прототип:

```
int setcookie (string name [, string value [, int expire [, string path
[, string domain [, int secure]]]]])
```

Параметры в точности соответствуют тем, которые используются в описанном выше заголовке Set-Cookie.

Если cookie-набор установлен как

```
setcookie ("mycookie", "value");
```

то когда пользователь обращается к следующей странице на вашем сайте (или перезагружает текущую страницу), вы получаете доступ к переменной с именем **$mycookie**, которая содержит значение **"value".** Доступ к этой переменной можно получить также через **$HTTP_COOKIE_VARS["mycookie"]**.

Для удаления cookie-набора необходимо вызвать **setcookie()** с тем же именем, но без указания значения. Если cookie-набор устанавливался с другими параметрами (такими как специфические URL или даты истечения), потребуется отправить те же параметры повторно, иначе cookie-набор удален не будет.

Для установки cookie-набора вручную можно воспользоваться также функцией **Header()** и описанным выше синтаксисом представления cookie-набора. Однако при этом следует иметь в виду, что заголовки cookie-наборов должны отправляться перед всеми другими заголовками (иначе заголовки cookie-наборов работать не будут).

Использование cookie-наборов в сеансах

При использовании cookie-наборов возникают некоторые проблемы: есть браузеры, которые не принимают cookie-наборы, а есть пользователи, которые запрещают использование cookie-наборов в своих браузерах. Это одна из причин, по которым в сеансах PHP используются двойной метод cookie-набор/адрес URL (ниже этот вопрос рассматривается более подробно).

В сеансе PHP нет необходимости задавать cookie-наборы вручную. Это за вас сделают функции сеанса.

Для того чтобы просмотреть содержимое cookie-набора, установленное при управлении сеансом, можно воспользоваться функцией **session_get_cookie_params()**. Она возвращает ассоциативный массив, содержащий элементы **lifetime, path и domain.**

Можно использовать также:

```
session_set_cookie_params($lifetime, $path, $domain);
```

этот оператор устанавливает параметры cookie-набора для сеанса.

Если возникнет желание получить более подробную информацию о cookie-наборах, то за консультациями по спецификации cookie-наборов следует обратиться на сайт компании Netscape:

```
http://home.netscape.com/newsref/std/cookie_spec.html
```

Сохранение идентификатора сеанса

В PHP cookie-наборы в сеансах используются по умолчанию. Если есть возможность установить cookie-наборы, то для сохранения идентификатора сеанса будет использоваться именно этот способ.

Другой метод, который может применяться в PHP, заключается в добавлении идентификатора сеанса к адресу URL. Можно сделать так, чтобы идентификатор сеанса добавлялся к URL автоматически — для этого следует скомпилировать PHP с опцией **--enable-trans-sid**.

Можно поступить и по-другому — встроить идентификатор сеанса в ссылку, чтобы обеспечить его передачу. Идентификатор сеанса будет запоминаться в константе SID, Для того чтобы передать его вручную, его потребуется добавить в конец ссылки, аналогично параметру **GET:**

```
<A HREF="link.php?<?=SID?>">
```

В общем случае проще компилировать PHP с **--enable-trans-sid**, если только это возможно (заметим попутно, что константа SID может использоваться для вышеописанных целей только в том случае, если конфигурация PHP выполнялась с **--enable-track-vars)**.

Реализация управления простым сеансом

Основными этапами использования сеанса являются следующие:

- Запуск сеанса
- Регистрация переменных сеанса
- Использование переменных сеанса
- Отмена регистрации переменных и закрытие сеанса

Заметим, что все перечисленные этапы не обязательно могут содержаться в одном сценарии, и некоторые из них могут находиться в нескольких сценариях. Рассмотрим каждый из этих этапов последовательно.

Запуск сеанса

Прежде чем можно будет воспользоваться функциональными возможностями сеанса, следует запустить сам сеанс. Существует три способа сделать это.

Первый (и самый простой) заключается в том, что сценарий начинается с вызова функции **session_start()**:

```
session_start();
```

Эта функция проверяет, существует ли идентификатор текущего сеанса. Если нет, она его создает. Если же идентификатор текущего сеанса уже существует, она загружает зарегистрированные переменные сеанса, чтобы они стали доступными для использования.

Надо отметить, что это прекрасный способ — вызов **session_start()** в начале сценариев, в которых используются сеансы.

Второй способ заключается в том, что сеанс запускается при попытке зарегистрировать переменные сеанса (см. следующий раздел).

Третий способ запустить сеанс — задать установки PHP, при которых сеанс будет запускаться автоматически, как только кто-либо посетит ваш сайт. Для этого следует

воспользоваться опцией **session.auto_start** в файле **php.ini** (более подробно указанный способ будет описан при рассмотрении конфигурации).

Регистрация переменных сеанса

Для того чтобы получить возможность отслеживать переменные от одного сценария к другому, их необходимо зарегистрировать. Это делается путем вызова функции **session_register().** Например, для регистрации переменной **$myvar** применяется следующий код:

```
$myvar = 5;
session_register("myvar");
```

Обратите внимание: вы должны передать в функцию **session_register()** строку, содержащую имя переменной. Эта строка не должна включать символ $.

Данный оператор регистрирует имя переменной и отслеживает ее значение. Отслеживание переменной будет осуществляться, пока не завершится сеанс либо пока вручную не отменится ее регистрация.

За один прием можно зарегистрировать более одной переменной, передав разделенный запятыми список имен переменных:

```
session_register("myvar1", "myvar2");
```

Использование переменных сеанса

Чтобы сделать переменную сеанса доступной для использования, сначала необходимо запустить сеанс, воспользовавшись одним из описанных выше способов.

После этого появляется доступ к этой переменной. Если опция **register_globals** включена (см. главу 1), то доступ к этой переменной можно получить через сокращенную форму ее имени, например, **$myvar**. Если же упомянутая опция не включена, получить доступ к переменной можно через ассоциативный массив **$HTTP_SESSION_VARS**, например, **$HTTP_SESSION_VARS["myvar"]**.

Переменные сеанса не могут быть перезаписаны данными **GET** или **POST.** Это хорошо с точки зрения обеспечения безопасности, однако сопряжено с некоторыми ограничениями при кодировании.

С другой стороны, от вас потребуется тщательность при проверке на предмет того, установлены ли уже переменные сеанса (например, с использованием isset() либо **empty()**). Кроме того, следует иметь в виду, что переменные могут быть установлены пользователем через **GET** или **POST.** Проверить, является ли переменная зарегистрированной переменной сеанса, можно обратившись к функции **session_is_registered().** Вызов функции выполняется следующим образом:

```
$result = session_is_registered("myvar");
```

Эта функция проверит, является ли **$myvar** зарегистрированной переменной сеанса, и вернет **true** или **false.**

Можно поступить и по-другому — проверить массив **$HTTP_SESSION_VARS** на предмет наличия в нем переменной.

Отмена регистрации переменных и завершение сеанса

После окончания работы с переменной сеанса ее регистрацию можно отменить, воспользовавшись функцией **session_unregister()**:

```
session_unregister("myvar");
```

Подобно функции регистрации, эта функция требует указания имени переменной, регистрацию которой необходимо отменить, в виде строки, не включающей символ $. Данная функция за один раз может отменить регистрацию только одной переменной сеанса (в противоположность **session_register()**). Однако, для отмены регистрации всех переменных текущего сеанса можно обратиться к **session_unset()**.

По завершении сеанса сначала потребуется отменить регистрацию всех переменных, а затем вызвать

```
session_destroy();
```

для обнуления идентификатора сеанса.

Пример простого сеанса

Изложенный выше материал может показаться несколько абстрактным, поэтому сейчас вашему вниманию предлагается пример. Приведенный в нем код обеспечивает обработку трех страниц.

На первой странице мы запустим сеанс и зарегистрируем переменную $sess_var. Код, позволяющий сделать это, показан в листинге 20.1.

Листинг 20.1 page1.php — запуск сеанса и регистрация переменной

```
<?
  session_start();
   session_register("sess_var");
   $sess_var = "Hello world!";
   echo "The content of \ $sess_var is $sess_var<br>";
?>
<a href = "page2.php">Next page</a>
```

Мы зарегистрировали переменную и установили ее значение. Результат работы этого сценария показан на рис. 20.1.

Заметим, что мы изменили значение переменной уже после ее регистрации. Можно, однако, сделать и наоборот — установить значение, а после этого зарегистрировать переменную. *Конечное* значение переменной на странице — это то значение, которое будет доступно на последующих страницах. В конце сценария переменная сеанса *преобразуется в последовательную форму (сериализуется),* или замораживается, до своей перезагрузки через следующий вызов **session_start()**.

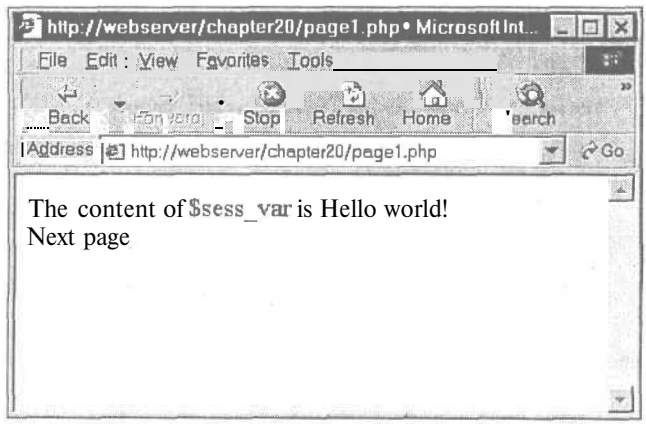


РИСУНОК 20.1
Исходное значение переменной сеанса, которое отображает page1.php.

Таким образом, следующий сценарий начинается с вызова **session_start()**. Сценарий показан в листинге 20.2.

Листинг 20.2 page2.php — получение доступа к переменной сеанса и отмена ее регистрации

```
<?
   session_start();
   echo "The content of \ $sess_var ia $sess_var<br>";
   session_unregister("sess_var");
?>
<a href = "page3.php">Next page</a>
```

После вызова **session_start()** переменная **$sess_var** станет доступной, а ее значением будет то, которое сохранено в предыдущем сеансе (см. рис. 20.2).

Сделав с переменной все необходимые действия, мы вызываем **session_unregister()** для отмены ее регистрации. Обратите внимание: сеанс еще существует, но переменная $sess_var уже больше не является зарегистрированной.

И наконец, мы переходим к page3.php, последнему сценарию в рассматриваемом примере. Код этого сценария показан в листинге 20.3.

Листинг 20.3 page3.php — завершение сеанса

```
<?
   session_start();
   echo "The content of \ $sess_var is $sess_var<br>";
   session_destroy();
?>
```

Как можно видеть из рис. 20.3, доступа к значению **$sess_var** больше нет.

И в завершение — вызов **session_destroy()** для разрушения идентификатора сеанса.

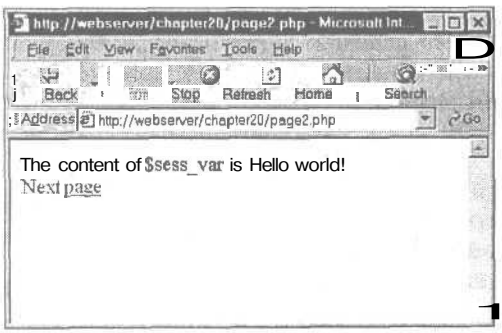


РИСУНОК 20.2 *Значение переменной сеанса, которая была передана через идентификатор сеанса странице page2.php.*

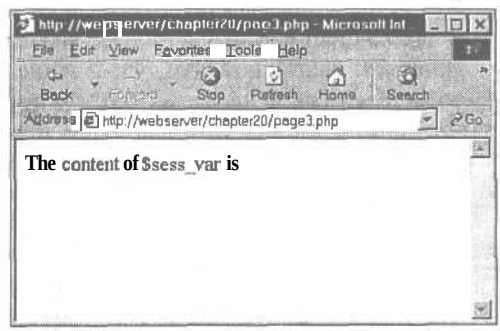


РИСУНОК 20.3 *После отмены регистрации переменная больше не доступна.*

Конфигурирование управления сеанса

А сейчас мы предлагаем ознакомиться с набором опций конфигурации для сеансов, которые можно установить в своем файле **php.ini.** В табл. 20.1 перечисляются некоторые из наиболее полезных опций вместе с их кратким описанием.

Таблица 20.1 Опции конфигурации сеанса

Имя опции	**Значение** *по умолчанию*	*Действие*
session.auto_start	О (запретить)	Автоматический запуск сеансов.
session.cache_expire	180	Установка времени жизни для кэшированных станиц сеанса (в минутах).
session.cookie_domain	none	Домен для установки в cookie-наборе сеанса.
session.cookie_lifetime	0	Определяет продолжительность существования cookie-набора идентификатора сеанса на машине пользователя. По умолчанию 0 — пока не будет закрыт браузер.
session.cookie_path	/	Путь для установки в cookie-наборе сеанса.
session.name	PHPSESSID	Имя сеанса, которое в системе пользователя используется как имя cookie-набора.
session.save_handler	файлы	Определяет место хранения данных сеанса. Здесь можно указать базу данных, однако для этого потребуется реализовать собственные функции.
session.save_path	/tmp	Путь к месту хранения данных сеанса. В более общем случае для определения и обработки передаваемых на хранение аргументов используется session.save_handler.
session.use_cookies	1 (разрешить)	Конфигурация сеанса с возможностью использования cookie-наборов на стороне клиента.

Выполнение аутентификации пользователей средствами управления сеансом

В завершение рассмотрим более важный пример использования контроля сеанса.

Наиболее часто, пожалуй, управление сеансом применяется в целях отслеживания пользователей после того, как они были аутентифицированы через механизм входной регистрации. В предлагаемом примере можно видеть, как эти функциональные возможности обеспечиваются за счет сочетания аутентификации при помощи базы данных MySQL и использования механизма управления сеансом.

Эти функциональные возможности составят основу проекта из главы 24, а впоследствии будут повторно использоваться и в других проектах.

В нашем примере мы воспользуемся базой данных аутентификации, которая была создана в главе 14. Это требуется для работы с модулем **mod_auth_mysql**. Дабы освежить в памяти подробности, касающиеся этой базы данных, следует обратиться к листингу 14.3 из главы 14.

Пример включает три простых сценария. Первый, **authmain.php,** обеспечивает форму для входной регистрации и аутентификации пользователей Web-сайта. Второй, **members_only.php**, представляет информацию только для тех пользователей, которые

успешно прошли входную регистрацию. Третий, **logout.php,** реализует выход пользователей из системы.

Чтобы понять, как все это работает, посмотрите на рис. 20.4. Это исходная страница, отображаемая сценарием **authmain.php**.

Данная страница предоставляет пользователю возможность войти в систему. В случае, если он предпримет попытку получить доступ к секции Members, не пройдя входную регистрацию, будет выдано сообщение, показанное на рис. 20.5.



РИСУНОК 20.4 *Поскольку пользователь еще не вошел в систему, для него отображается входная страница*



РИСУНОК 20.5 *Пользователи, которые не прошли входную регистрацию, не смогут получить доступ к содержимому сайта. Вместо этого они увидят показанное сообщение*

Если же пользователь сначала прошел входную регистрацию (с именем пользователя: testuser и паролем: test123, как было задано в главе 14), а после этого предпринял попытку войти на страницу Members, он увидит то, что представлено на рис. 20.6.

Давайте посмотрим на код приложения. Большая часть кода сосредоточена в сценарии **authmain.php,** приведенном в листинге 20.4. Давайте изучим его более подробно.

Листинг 20.4 authmain.php — Основная часть приложения аутентификации

```
<?
session_start();
if ($userid && $password)
{
   // если пользователь как раз пытается зарегистрироваться
   $db_conn = mysql_connect("localhost", "webauth", "webauth");
   mysql_select_db("auth", $db_conn);
   $query = "select * from auth "
            ."where name='$userid' "
            ." and pass=password('$password')";
   $result = mysql_query($query, $db_conn);
   if (mysql_num_rows($result) >0 )
   {
      // если пользователь найден в базе данных,
      // зарегистрировать его идентификатор
      $valid_user = $userid;
      session_register("valid_user");
   }
}
```

```
?>
<html>
<body>
<hl>Home page</hl>
<?
   if (session_is_registered("valid_user"))
   {
      echo "You are logged in as: $valid_user <br>";
      echo "<a href=\"logout.php\">Log out</a><br>";
   }
   else
   {
      if (isset($userid))
      {
         // если пользователь пытался зарегистрироваться,
         // но возникла ошибка
         echo "Could not log you in";
      }
      else
      {
         // если пользователь либо не пытался зарегистрироваться,
         // либо покинул сайт
         echo "You are not logged in.<br>";
      }
      // форма для аутентификации
      echo "<form method=post action=\"authmain.php\">";
      echo "<table>";
      echo "<tr><td>Userid:</td>";
      echo "<td><input type=text name=userid></td></tr>";
      echo "<tr><td>Password:</td>";
      echo "<td><input type=password name=password></td></tr>";
      echo "<tr><td colspan=2 align=center>";
      echo "<input type=submit value=\"Log in\"></td></tr>";
      echo "</table></form>";
   }
?>
<br>
<a href="members_only.php">Members section</a>
</body>
</html>
```

Данный сценарий отличается сложной (в разумных пределах) логикой, но иначе нельзя: ведь он осуществляет представление формы для входной регистрации и ее обработку.

Работа этого сценария сосредоточена вокруг переменной сеанса **$valid_user**. Основная идея здесь заключается в следующем: если кто-либо успешно прошел процедуру входной регистрации, мы регистрируем переменную сеанса с именем **$valid_user,** которая содержит идентификатор пользователя.

Так что же первым делом выполняется в сценарии? Правильно, вызов **session_start()**. Эта функция загружает переменную сеанса **$valid_user,** если последняя была зарегистрирована.



РИСУНОК 20.6 *После прохождения входной регистрации пользователь может получить доступ к частям сайта, доступным для зарегистрированных пользователей сайта*

При первом проходе по сценарию ни один из условных операторов **if** не сработает и неудачливому пользователю к концу сценария останется лишь внимательно прочесть сообщение о том, что он не прошел процедуру входной регистрации. После этого мы предоставляем ему форму, при помощи которой он сможет это сделать:

```
echo "<form method=post action=\ "authmain.php\ ">";
echo "<table>";
echo "<tr><td>Userid:</td>";
echo "<td><input type=text name=userid></td></tr>";
echo "<tr><td>Password:</td>";
echo "<td><input type=password name=password></td></tr>";
echo "<tr><td colspan=2 align=center>";
echo "<input type=submit value=\ "Log in\ "></td></tr>";
echo "</table></form>";
```

Когда пользователь нажмет кнопку отправки (Submit), сценарий вызывается заново и вновь все начинается с начала. На этот раз в нашем распоряжении будут имя пользователя и пароль, позволяющие его аутентифицировать (они хранятся в **$userid** и **$password**). Если эти переменные установлены, переходим к блоку аутентификации:

```
if ($userid && $password)
{
  // если пользователь как раз пытается зарегистрироваться
  $db_conn = mysql_connect("localhost", "webauth", "webauth");
  mysql_select_db("auth", $db_conn);
  $query = "select * from auth "
           ."where name='$userid' "
           ." and pass=password('$password')";
  $result = mysql_query($query, $db_conn);
```

И вот мы подключаемся к базе данных MySQL и проверяем имя пользователя и пароль. Если в базе данных существует соответствие этой паре, мы регистрируем переменную **$valid_user**, которая содержит идентификатор для конкретного пользователя. Таким образом, мы знаем, кто вошел в систему и, соответственно, будем его отслеживать.

```
if (mysql_num_rows($result) >0 )
  {
    // если пользователь найден в базе данных,
    // зарегистрировать его идентификатор
    $valid_user = $userid;
    session_register("validuser");
  }
}
```

Поскольку уже известно, кто сейчас посещает сайт, то повторно предоставлять ему форму входной регистрации нет необходимости. Вместо этого мы сообщаем пользователю, что мы знаем, кто он такой, и даем ему возможность выхода из системы:

```
if (session_is_registered("valid_user"))
{
  echo "You are logged in as: $valid_user <br>";
  echo "<a href=\"logout.php\">Log out</a><br>";
}
```

Если же при попытке произвести входную регистрацию пользователя мы по какой-то причине терпим неудачу, то у нас имеется идентификатор пользователя, но нет переменной **$valid_user,** и ничего не остается, кроме как выдать сообщение об ошибке:

```
if (isset($userid))
{
   // если пользователь пытался зарегистрироваться, но возникла ошибка
   echo "Could not log you in";
}
```

Поскольку **$valid_user** является зарегистрированной переменной сеанса, ее нельзя перезаписать путем передачи другого значения через URL, например так:

```
members_only.php?valid_user=testuser
```

С основным сценарием, похоже, все понятно. А теперь посмотрим на страницу Members. Код этого сценария показан в листинге 20.5.

Листинг 20.5 members_only.php — код для секции Members в процедуре проверки Web-сайтом достоверности пользователя

```
<?
   session_start();
   echo "<h1>Members only</h1>";
   // проверить переменные сеанса
   if (session_is_registered("valid_user"))
   {
      echo "<p>You are logged in as $valid_user.</p>";
      echo "<p>Members only content goes here</p>";
   )
   else
   {
      echo "<p>You are not logged in.</p>";
      echo "<p>Only logged in members may see this page.</p>";
   >
   echo "<a href=\"authmain.php\">Back to main page</a>";
?>
```

Приведенный выше код очень прост. Все, что он делает — это запуск сеанса и проверка того, содержит ли текущий сеанс зарегистрированного пользователя, с использованием функции **session_registered_user()**. Если пользователь прошел процедуру входной регистрации, мы отображаем содержимое сайта для зарегистрированных пользователей, в противном случае мы сообщаем ему, что у него нет соответствующих полномочий.

И в завершение рассмотрим сценарий **logout.php,** который завершает регистрацию пользователя в системе. Код сценария показан в листинге 20.6.

Листинг 20.6 logout.php — этот сценарий выполняет отмену регистрации переменных сеанса и завершает сеанс

```
<?
   session_start();
   $old_user = $valid_user;   // сохранить для проверки,
                              // регистрировался ли пользователь
   $result = session_unregister("valid_user");
   session_destroy();
?>
<html>
<body>
<h1>Log out</h1>
<?
   if (!empty($old_user))
   {
      if ($result)
```

```
    {
      // если пользователь был зарегистрирован и не покинул систему
      echo "Logged out.<br>";
    }
    else
    {
    // если пользователь был зарегистрирован и не может покинуть систему
      echo "Could not log you out.<br>";
    }
  }
  else
  {
    // если пользователь не был зарегистрирован,
     // но каким-то образом попал на эту страницу
    echo "You were not logged in, and so have not been logged out.<br>";
  }
?>
```

Приведенный код очень прост, но позволяет выполнить несколько изящных танцевальных па. Мы запускаем сеанс, запоминаем старое имя пользователя, отменяем регистрацию переменной **$valid_user** и завершаем сеанс. После этого мы выдаем пользователю одно из следующих сообщений: он вышел из системы, не может выйти из системы или не может выйти из системы, поскольку первоначально даже не регистрировался.

Осталось отметить, что приведенный выше набор простых сценариев послужит основой для многих проектов, которыми мы займемся в дальнейших главах.

Дополнительная информация

Хотя встроенный механизм сеансов является новинкой PHP 4, в более ранних версиях сеансы отчасти обеспечивались средствами PHPLib. Чтобы получить больше информации об этом, лучше всего обратиться на страницу PHPLib и к спецификациям cookie-наборов. В данной главе мы уже приводили оба требуемых адреса URL, но будет нелишним повторить адреса еще раз:

```
http://phplib.netuse.de/index.php3
http://home.netscape.com/newsref/std/cookie_spec.html
```

Что дальше

Работа над очередной частью книги практически завершена.

Но прежде чем перейти к проектам, мы кратко рассмотрим некоторые полезные мелочи PHP, о которых до сих пор еще не упоминалось.

Глава 21

Другие полезные свойства PHP

В этой главе рассказано о тех свойствах и функциях PHP, которые не подпадают под какую-либо определенную категорию.

Здесь рассматриваются:

- Использование магических кавычек
- Выполнение команд, содержащихся в строке, – функция eval()
- Прерывание выполнения: **die** и **exit**
- Сериализация
- Получение информации о рабочей среде PHP
- Временное изменение среды исполнения
- Загрузка расширений PHP
- Выделение элементов исходного кода

Использование магических кавычек

Возможно, вы заметили, что использование символов кавычек (' и ") и символа обратной косой черты (\) внутри строк требует некоторой осторожности. Интерпретатор PHP легко запутать выражением наподобие

```
echo "color = "#FFFFFF"";
```

в результате чего он выдаст ошибку. Чтобы разместить кавычки внутри строки, необходимо, чтобы они отличались от кавычек, содержащих строку. Например, и вариант

```
echo "color = '#FFFFFF'";
```

и вариант

```
echo 'color = "#FFFFFF"';
```

будут правильными.

Та же проблема возникает с пользовательским вводом, а также вводом и выводом других программ.

Попытка запустить запрос **mysql** наподобие

```
insert into company values ('Bob's Auto Parts');
```

точно так же запутает синтаксический анализатор MySQL.

Здесь уже рассматривались функции **addslashes() и stripslashes()**, позволяющие использовать внутри строки символы кавычек, двойных кавычек, обратной косой черты и NUL-символы.

PHP имеют полезную способность автоматически (или "магически") добавлять и убирать управляющие символы косой черты. С помощью двух установок в файле **php.ini** можно включить или выключить магические кавычки для данных **GET, POST,** cookie-наборов и других источников.

Значение директивы **magic_quotes_gpc** управляет применением волшебных кавычек для операций **GET, POST** и обработки cookie-наборов.

Если при включенной опции **magic_quotes_gpc** пользователь введет в форме "Bob's Auto Parts", сценарий получит строку "Bob\'s Auto Parts", поскольку символ кавычки автоматически модифицируется.

Функция **get_magic_quotes_gpc()** возвращает 1 или 0, что отвечает текущему состоянию директивы **magic_quotes_gpc**. Ею можно воспользоваться для проверки, нужно ли вызывать функцию **stripslashes()** для данных, полученных от пользователя.

Значение **magic_quotes_runtime** управляет использованием магических кавычек в функциях, получающих данные из баз данных и файлов.

Для определения текущего значения **magic_quotes_runtime** применяется функция **get_magic_quotes_runtime()**, которая также возвращает 1 или 0. Магические кавычки можно включить или выключить непосредственно внутри сценария при помощи функции **set_magic_quotes_runtime()**.

Выполнение команд, содержащихся в строке, - функция eval()

Функция **eval()** позволяет выполнить строку как PHP-код.

Например, вызов

```
eval ( "echo 'Hello World';" );
```

выполняет содержимое строки. Этот оператор производит точно такой же вывод, как

```
echo 'Hello World';
```

Есть несколько ситуаций, в которых функция **eval()** может оказаться полезной. Например, можно сохранить фрагмент кода в базе данных, а затем выполнить его с помощью функции **eval(),** или генерировать код в цикле (сохраняя его в строке), а по его окончании выполнить также с помощью этой функции.

Кроме того, **eval()** можно применять для обновления или исправления существующего кода. Если имеется большой набор сценариев, требующих однотипных изменений, можно (но неэффективно) написать сценарий, реализующий следующий алгоритм: загрузить старый сценарий в строку, произвести изменения с помощью **regexp,** запустить старый сценарий функцией **eval().**

Вполне вероятно, что может потребоваться позволить вводить PHP-код и запускать на сервере, — здесь снова потребуется функция **eval().**

Прерывание выполнения: die и exit

В этой книге для останова выполнения сценария применялся оператор **exit.** Вспомните, что он имеет вид:

```
exit;
```

и ничего не возвращает.

Более полезной функцией прекращения выполнения сценария является **die().** Она позволяет вывести сообщение об ошибке или запустить другую функцию до останова сценария. Она знакома программистам на языке Perl.

Ее можно использовать саму по себе подобно **exit:**

```
die("Script ending now");
```

Однако чаще она объединяется оператором or с выражением, которое может завершиться неудачей, например, открытием файла или соединением с базой данных:

```
mysql_query($query) or die("Could not execute query");
```

Вместо того чтобы просто вывести сообщение, перед завершением сценария можно запустить одну функцию:

```
function err_msg()
{
     echo "MySQL error was: ";
     echo mysql_error() ;
)
mysql_query ($query) or die (err_msg());
```

Это позволяет вывести пользователю причину, по которой сценарий завершился неудачей.

Подобным образом можно отправить себе сообщение об ошибке по электронной почте или добавить его в журнальный файл.

Сериализация

Сериализация представляет собой процесс превращения того, что могут хранить в себе переменная или объект PHP, в поток байтов, который можно сохранить в базе данных или передавать от страницы к странице. Без этой возможности было бы трудно сохранить или переслать целиком содержимое массива или объекта.

Полезность этого свойства несколько уменьшилась с появлением управления сеансами. Фактически, Сериализация использовалась для того, для чего теперь служит управление сеансами. Функции управления сеансами сериализуют переменные сеанса для того, чтобы сохранять их между HTTP-запросами.

Тем не менее, может возникнуть необходимость сохранить массив или объект PHP в файле или базе данных. В этом случае требуются две функции: **serialize() и unserialize().**

Вызов функции **serialize()** имеет вид:

```
$serial_object = serialize($my_object);
```

Чтобы понять, что происходит при сериализации, достаточно изучить значение, возвращаемое этой функцией. Она превращает содержимое объекта или массива в строку.

Например, можно испытать функцию **serialize()** на простом объекта **employee,** имеющем вид:

```
class employee
{
     var  $name;
     var  $employee_id;
};
$this_emp = new employee;
$this_emp->name = "Fred";
$this_emp->employee_id = 5324;
```

Вывод в браузер результата сериализации данного объекта выглядит так

```
O:8:"employee":2:{s:4:"name";s:4:"Fred";s:11:"employee_id";i:5324;}
```

Здесь легко заметить взаимосвязь между исходным объектом и сериализованными данными.

Поскольку сериализованные данные представляют собой обычный текст, его можно записать в базу данных или распорядиться с ним любым другим образом. Как и всегда, до записи в базу данных следует воспользоваться функцией **addslashes().** Необходимость в этом очевидна, хотя бы по причине наличия в строке кавычек.

Для получения объекта используется обратная функция — **unserialize()**:

```
$new_object = unserialize($serial_object);
```

Очевидно, что до обратного действия следует вызвать функцию stripslashes(), если перед помещением объекта в базу данных использовалась функция **addslashes()**.

Получение информации о рабочей среде PHP

Для получения информации о том, как сконфигурирован PHP, существует несколько функций.

Определение загруженных расширений

Доступные наборы функций, а также конкретные функции в этих наборах легко определить, используя функции **get_loaded_extensions() и get_extension_funcs()**.

Функция **get_loaded_extensions()** возвращает массив, содержащий все наборы функций, доступные PHP в текущий момент. Имя определенного набора или расширения является аргументом функции **get_extension_funcs(),** которая возвращает массив имен функций в данном наборе.

Сценарий из листинга 21.1 выводит список всех функций, доступных в данной инсталляции PHP.

Листинг 21.1 list_functions.php **— сценарий выводит все расширения PHP, а также функции, содержащиеся в каждом из них**

```
<?
    echo "Function sets supported in this install are:<br>";
    $extensions = get_loaded_extensions();
    foreach  ($extensions as $each_ext)
    {
      echo "$each_ext <br>";
      echo "<ul>";
      $ext_funcs = get_extension_funcs($each_ext);
      foreach($ext_funcs as $func)
      {
          echo "<li> $func";
      }
      echo "</ul>";
    }
?>
```

Обратите внимание, что функция **get_loaded_extensions()** вообще не имеет аргументов, а **get_extension_funcs()** принимает только один — имя расширения.

Подобная информация оказывается полезной, если необходимо определить, было ли расширение установлено успешно.

Определение владельца сценария

Определить, какой пользователь является владельцем текущего процесса (запущенного сценария) можно при помощи функции **get_current_user()**:

```
echo get_current_user();
```

Эта информация может потребоваться при решении вопросов с правами доступа.

Определение даты последнего изменения сценария

Сейчас распространена тенденция помещать на страницу дату последней модификации.

Для определения даты последнего изменения сценария используется функция **getlastmod()** (обратите внимание на отсутствие символов подчеркивания в имени функции):

```
echo date("g:i a, j M Y",getlastmod());
```

Функция возвращает метку времени UNIX, которая затем передается в функцию date() с целью придания результату более читаемого формата.

Динамическая загрузка расширений

Библиотеки расширений можно загружать во время выполнения, используя функцию **dl()**.

Аргументом функции является имя библиотечного файла. В UNIX имена таких файлов заканчиваются расширением **.so**, в Windows — расширением **.dll**.

Вот пример вызова функции **dl()**

```
dl("php_ftp.dll");
```

При этом динамически загружается расширение FTP (на Windows-машине).

Здесь не нужно указывать каталог, в котором содержится файл: это следует настроить в конфигурационном файле **php.ini.** Директива **extension_dir** задает каталог, где PHP должен искать библиотеки для динамической загрузки.

Если при динамической загрузке расширений возникают трудности, необходимо проверить директиву **enable_dl** в файле **php.ini.** Если она отключена, то динамически загружать расширения нельзя. Эта директива может быть отключена по соображениям защиты. Если PHP запущен в безопасном режиме, то функцию **dl()** также нельзя использовать.

Временное изменение среды исполнения

Набор директив в файле **php.ini** можно просматривать и даже изменять на время выполнения сценария. Это может оказаться чрезвычайно полезным, например, для директивы **max_execution_time**, когда необходимо увеличить допустимое время выполнения сценария.

Просматривать и изменять директивы можно при помощи функций **ini_get()** и **ini_set()**. Пример использования этих функций приведен в простом сценарии в листинге 21.2.

Листинг **21.2** iniset.php — сценарий инициализирует переменные из файла php.ini

```
<?
    $old_max_execution_time = ini_set(max_execution_time, 120) ;
    echo "old timeout is $old_max_execution_time <br>";
    $max_execution_time = ini_get(max_execution_time);
    echo "new timeout is $max_execution_time <br>";
?>
```

Функция **ini_set()** принимает два аргумента. Первый представляет собой имя конфигурационной директивы из файла **php.ini,** а второй — ее новое значение. Функция возвращает предыдущее значение этой директивы.

В данном случае значение максимального времени выполнения сценария устанавливается равным 120 секунд (вместо 30 по умолчанию).

Функция **ini_get()** просто возвращает значение директивы, имя которой является аргументом, задаваемым как строковое выражение. В сценарии эта функция используется только для проверки того, что значение директивы действительно изменилось.

Выделение элементов исходного кода

В PHP встроена система выделения синтаксиса, подобная многим интегрированным средам разработки. Она удобна для передачи кода другим разработчикам или презентации его на Web-странице для обсуждения.

Функции **show_source()** и **highlight_file()** являются идентичными. (Фактически, **show_source()** представляет собой просто псевдоним функции **highlight_file()**.)

Аргументом обеих функций является имя файла. (Этот файл должен содержать PHP-код, иначе вызов функций не даст какого-либо приемлемого результата.) Например,

```
show_source("list_functions.php");
```

Файл отображается в браузере, причем текст выделяется различными цветами в зависимости от того, является ли он строкой, комментарием, ключевым словом или HTML-дескриптором. Вывод печатается на заданном цвете фона. Содержимое, не подпадающее ни под одну из перечисленных категорий, выводится цветом по умолчанию.

Функция **highlight_string()** работает подобным образом, но ее аргументом является строка, а результатом — вывод в браузере в формате с выделенным синтаксисом.

Цвета для выделения синтаксиса можно установить в файле **php.ini.** Соответствующий раздел файла выглядит следующим образом:

```
; Colors for Syntax Highlighting mode
highlight.string    =    #DD0000
highlight.comment   =    #FF8000
highlight.keyword   =    #007700
highlight.bg        =    #FFFFFF
highlight.default   =    #0000BB
highlight.html      =    #000000
```

Цвета заданы в стандартном RGB-формате HTML.

Что дальше

В части V рассматриваются несколько достаточно сложных проектов, реализованных при помощи PHP и MySQL. Они представляют собой полезные примеры распространенных задач, демонстрирующие применение PHP и MySQL в больших проектах.

В главе 22 изложены основные проблемы, возникающие при создании больших проектов на PHP, а также такие основные принципы разработки программного обеспечения, как проектирование, документирование и управление изменениями.

Часть 5

Разработка практических приложений на PHP и MySQL

В этой части:

22 Применение PHP и MySQL при разработке крупных проектов

23 Отладка

24 Аутентификация и персонализация пользователей

25 Создание покупательской тележки

26 Построение системы управления содержимым

27 Построение почтовой службы, основанной на Web

28 Создание менеджера списков рассылки

29 Создание Web-форумов

30 Генерация персонифицированных документов в формате переносимых документов (PDF)

Глава 22

Применение PHP и MySQL при разработке крупных проектов

В предыдущих частях книги обсуждались различные компоненты и применения PHP и MySQL. Мы старались сделать все примеры интересными и актуальными, но они довольно-таки просты и включают один-два сценария длиной до 100 строк кода.

Создание реальных Web-приложений редко бывает настоль простым. Несколько лет назад "интерактивный" Web-сайт предоставлял возможность отправки формы по электронной почте и это было все. В наши дни Web-сайты стали Web-приложениями, другими словами, настоящими программными средствами, доступными через Web. В результате вместо коротких сценариев Web-сайты содержат тысячи и тысячи строк кода. Проекты подобного масштаба требуют планирования и управления в такой же степени, как и разработка любых других программных средств.

Прежде чем приступить к обзору проектов, рассмотрим некоторые приемы управления крупными Web-проектами. Это постоянно развивающееся искусство, и, как показывает изучение рынка, постигнуть его не так-то просто.

В этой главе рассматриваются такие вопросы:

- Применение методов проектирования программного обеспечения при разработке Web-приложений
- Планирование и сопровождение проекта Web-приложения
- Повторное использование кода
- Написание удобного для сопровождения кода
- Управление версиями
- Выбор среды разработки

- Документирование проектов
- Моделирование
- Разделение логики, содержимого и представления: PHP, HTML и CSS
- Оптимизация кода

Применение методов проектирования программного обеспечения при разработке Web-приложений

Возможно, вам уже известно, что проектирование — это применение методов систематизации и количественных измерений к разработке программных средств. Другими словами, это приложение принципов проектирования по отношению к разработке программ.

Отметим, что данный подход заметно отсутствует во многих Web-проектах, что вызвано двумя основными причинами.

Во-первых, разработка Web-приложений зачастую напоминает создание отчетов. Она предполагает построение структуры документа, графическое оформление и публикацию. Этот подход ориентирован на документ. Он вполне применим для статических сайтов малого и среднего размеров. Однако, с возрастанием динамического содержимого Web-сайтов до уровня, когда они предоставляют не столько документы, сколько услуги, данный принцип становится непригодным. Многим вовсе не приходит в голову воспользоваться принципами проектирования программного обеспечения.

Вторая причина состоит в том, что условия разработки Web-приложений во многом отличаются от разработки обычных программных систем. Работа ведется в исключительно сжатые сроки и под постоянным нажимом создать сайт немедленно. Ведение проектов обычных программ предполагает последовательность и методичность, а на планирование специально выделяется время. При разработке Web-проектов часто господствует ощущение, что на планирование вообще нет времени.

Отсутствие планирования Web-проектов приводит к таким же результатам, как и при отсутствии планирования для любых других программ: ошибки в коде, нарушение сроков и неудобочитабельный код.

Трудность заключается в выборе методов сопровождения проектов программного обеспечения, пригодных для разработки Web-приложений, и отказе от всех остальных.

Планирование и сопровождение проекта Web-приложения

Не существует универсального метода планирования жизненного цикла Web-проектов. Однако есть ряд моментов, которые необходимо учесть. Ниже приводится их перечень, а более подробное обсуждение содержится в последующих разделах. Необязательно следовать этим рекомендациям в порядке их изложения, если это не подходит к определенному проекту. Главное здесь — иметь представление о данных вопросах и выбирать рекомендации, применимые к конкретному случаю.

- Для начала следует продумать конечную цель создаваемого продукта. Необходимо уяснить конечные цели. Многие технически совершенные Web-проекты провалились именно потому, что никто не проверил, существуют ли пользователи, заинтересованные в подобного рода приложениях.
- Постарайтесь разбить приложение на отдельные компоненты. Каковы этапы разработки приложения? Как будет действовать каждый компонент? Как компонен-

ты будут взаимно дополнять друг друга? Здесь помогут сценарии, эскизы и даже случаи использования (use cases).

- После составления списка компонентов следует выяснить, какие из них уже существуют. Если ранее созданный модуль обладает необходимыми функциями, возможно, имеет смысл использовать именно его. Не забывайте искать готовый код как в своей организации, так и за ее пределами. В частности, сообщество открытого исходного кода (Open Source) бесплатно предлагает множество компонентов. Определите, какой код придется создавать с нуля, и, приблизительно, насколь это окажется трудоемкой задачей.
- Продумайте организацию процесса. В Web-проектах этим зачастую пренебрегают. Здесь подразумеваются стандарты написания кода, структура каталогов, управление версиями, среда разработки, уровень и стандарты документирования, а также распределение задач между членами группы разработчиков.
- Постройте модель на основе ранее изложенных соображений. Продемонстрируйте ее пользователям. Внесите в модель требуемые изменения.
- Помните, что на всех этапах важно разграничивать содержимое и логику приложения. Эта идея более подробно рассматривается ниже.
- Выполните необходимую оптимизацию.
- Выполняйте тестирование столь же тщательно, как и для любого другого программного проекта.

Повторное использование кода

Программисты часто по ошибке создают код, который уже существует. Если известны необходимые компоненты приложения либо хотя бы необходимые функции, перед тем как приступить к разработке, проверьте, что имеется в наличии.

Иногда программисты повторно реализуют функции потому, что не прочли в руководстве о том, что существующие функции предоставляют необходимые возможности. Всегда обеспечивайте возможность быстрого вызова руководства в интерактивном режиме либо загружайте текущую версию и просматривайте ее в автономном режиме. Однако имейте в виду, что интерактивное руководство довольно часто обновляется. Кроме того, такое руководство содержит ссылки, что делает его великолепным ресурсом с комментариями, рекомендациями и примерами кода других пользователей. Оно часто содержит ответы на вопросы, которые обычно возникают после чтения основной страницы руководства. Вот адрес, где находится руководство по PHP:

```
http://www.php.net/manual/
```

Некоторые неанглоязычные программисты поддаются искушению писать функции-упаковщики, которые фактически присваивают PHP-функциям новые имена на родном для разработчика языке. Эту практику часто называют "синтаксическим сахаром" (syntactic sugar). Так делать не рекомендуется — это усложняет для других чтение и сопровождение кода. Если вы изучаете новый язык, учитесь правильно его использовать. Кроме того, подобное добавление уровня вызова функций замедляет выполнение кода. С учетом всех обстоятельств, подобного подхода следует избегать.

Если выяснилось, что необходимые функциональные возможности не обеспечиваются базовой библиотекой PHP, существует два пути. Для простых задач имеет смысл создать собственную функцию или объект. Однако при написании достаточно сложного кода, такого как покупательская тележка, система электронной почты Web или Web-

форумы, нередко обнаруживается, что работа уже кем-то проделана. Одно из преимуществ работы с сообществом Open Source состоит в том, что код компонентов подобных приложений, как правило, распространяется бесплатно. Если найден компонент, похожий на требуемый, пусть даже и не вполне соответствующий, можно просмотреть исходный код и на его основе выполнить модификацию или создать собственную программу.

После завершения разработки собственных функций или компонентов имеет смысл всерьез задуматься над тем, чтобы предоставить их сообществу PHP. Именно этот принцип делает сообщество разработчиков на PHP таким полезным, активным и информированным.

Написание удобного для сопровождения кода

Проблема сопровождения кода в Web-приложениях часто игнорируется. Обычно это происходит по причине спешки. Быстро начать и завершить работу иногда представляется более важным, чем выполнять предварительное планирование. Однако небольшие затраты времени в начале могут сэкономить много времени в дальнейшем, когда потребуется создать следующую версию приложения.

Стандарты написания кода

Большинство крупных организаций, занимающихся информационными технологиями, устанавливают собственные стандарты кодирования — правила именования файлов и переменных, написания комментариев, применения выравнивания в коде и т.п.

Поскольку ранее разработка Web-приложений основывалась на понятии документа, стандартами кодирования в этой области нередко пренебрегали. Когда код пишется самостоятельно либо силами небольшой группы, значение стандартизации легко недооценить. Однако так не следует поступать, поскольку состав рабочей группы и масштаб проекта могут вырасти. В конечном итоге получится неразбериха, и программисты не смогут разобраться в существующем коде.

Правила именования

Цель выработки правил именования заключается в следующем:

- Сделать код простым для восприятия. Осмысленное именование переменных и функций позволяет читать код почти как обычный текст или хотя бы псевдокод.
- Упростить запоминание идентификаторов. Если идентификаторы сформированы единообразно, будет проще вспомнить название определенной переменной или функции.

Имена переменных должны описывать данные, которые они содержат. Если в переменной хранится фамилия, назовите ее **$surname**. Необходимо найти оптимальное соотношение между краткостью и читабельностью имен. Например, сохранение имени в переменной $n упростит набор кода, но усложнит его понимание. Имя **$surname_of_the_current_user** (фамилия текущего пользователя) более информативное, но неоправданно длинное (неудобно печатать и выше вероятность допустить опечатку).

Необходимо выработать правила использования регистров. Как уже упоминалось, в PHP имена переменных зависят от регистра. Потребуется решить, как будут записываться имена переменных: в нижнем регистре, верхнем регистре либо в их комбинации. Например, можно принять, что первые буквы слов будут прописными. Мы пред-

почитаем использовать только нижний регистр, поскольку это проще для запоминания.

Неплохо использовать регистр для различения переменных и констант. Обычно для переменных используют символы нижнего регистра (например, **$result**), а для констант — верхнего (например, PI).

Некоторые программисты присваивают двум переменным одно и то же имя, но с символами разного регистра, например, **$name** и **$Name**. Надеюсь, не стоит объяснять, почему подобная практика не рекомендуется.

Лучше избегать сложных схем применения регистров, например, $WaReZ. Они трудны для запоминания.

Кроме того, стоит выбрать конструкцию имен переменных, состоящих из нескольких слов. Доводилось встречаться со всеми перечисленными ниже конструкциями:

```
$username
$user_name
$UserName
```

Совершенно не важно, какой из них будет отдано предпочтение, главное — применять ее последовательно. Имеет также смысл придерживаться разумных ограничений в количестве слов — не более двух-трех.

Для имен функций применимы многие из рассмотренных соображений, но с парочкой отличий. Имена функций обычно основываются на глаголах. Например, следующие имена встроенных PHP-функций описывают действия, выполняемые над передаваемыми параметрами: **addslashes()** (добавить обратные косые) и mysql_connect() (подключиться к MySQL). Это существенно упрощает восприятие кода. Обратите внимание, что в приведенных выше именах функций применены различные схемы именования переменных с использованием нескольких слов. В этом отношении PHP-функции непоследовательны. В какой-то мере это связано с тем, что они написаны большой группой программистов.

Кроме того, следует учитывать, что в PHP имена функций не зависят от регистра. Тем не менее, во избежание путаницы лучше придерживаться определенного формата.

Иногда применяют модульную схему именования, которая встречается во многих PHP-модулях — имя модуля служит префиксом имени функции. Например, имена всех функций модуля MySQL начинаются с префикса mysql_, функций ШАР — с префикса imap_. Так, для функций модуля покупательской тележки (shopping cart) имеет смысл использовать префикс cart.

В заключение отметим, что сами по себе правила именования не имеют большого значения. Важно лишь последовательно придерживаться выбранной схемы.

Комментирование кода

Все программы в разумных пределах должны комментироваться. Может возникнуть вопрос, каковы эти пределы. Обычно имеет смысл добавлять комментарии к каждому из следующих элементов:

- Файлам, будь то завершенные сценарии либо подключаемые (через **include)** файлы. Комментарии каждого файла должны информировать о его назначении, авторе и времени обновления.
- Функциям. Необходимо указать, какие действия функция выполняет, какие данные вводятся и что возвращается.

- Классам. Описывается назначение класса. Методы классов должны сопровождаться комментариями того же типа и уровня, что и все другие функции.
- Блокам кода внутри сценария или функции. Мы часто начинаем писать сценарий с набора комментариев в стиле псевдокода, а затем заполняем кодом каждый раздел. Например, на начальном этапе сценарий может иметь следующий вид:

```
<?
// Проверка входных данных
// Отправка информации в базу данных
// Выдача результатов
?>
```

  Это довольно удобно, поскольку после заполнения всех разделов код уже будет содержать комментарии.
- Сложным элементам кода. Если какой-либо фрагмент потребовал долгих раздумий либо содержит хитрый трюк, стоит написать к нему пояснение. Тогда при последующем просмотре кода не придется чесать затылок и думать: "Что бы это могло значить?".

Следующая общая рекомендация — писать комментарии в процессе работы над кодом. Не стоит рассчитывать на возможность вернуться к коду после завершения работы над проектом и внести в него комментарии. Уверен, что это никогда не удается, если только у вас не окажется намного менее напряженный график работы и гораздо больше самодисциплины, нежели у нас.

Выравнивание кода

В любом языке программирования необходимо осмысленное и единообразное применение выравнивания (отступов) при оформлении кода. Это напоминает форматирование резюме или делового письма. Выравнивание существенно упрощает восприятие кода.

Обычно любой блок программы, принадлежащий управляющей структуре, выделяется отступом относительно окружающего кода. Отступ должен быть хорошо заметным (более одного пробела), но не слишком большим. Обычно мы против применения табуляции. Это ускоряет печать, но для многих мониторов отнимает слишком много экранного пространства. Во всех проектах мы используем отступы размером в два или три пробела.

Расположение фигурных скобок также заслуживает внимания. Ниже показаны два распространенных варианта:

Вариант 1:

```
if (condition) {
   // какая-то обработка
}
```

Вариант 2:

```
if (condition)
{
   // какая-то обработка
}
```

Какой из них использовать — дело вкуса. Опять-таки, во избежание неразберихи выбранный стиль должен последовательно применяться на протяжении всего проекта.

Фрагментирование кода

Монолитный код огромных размеров крайне неудобен. Некоторые программисты создают один большой сценарий, который все операции выполняет в одной главной строке кода. Намного лучше разбить код на функции и/или классы, а также поместить взаимосвязанные элементы в подключаемые классы. Например, имеет смысл включить все связанные с базами данных функции в файл с именем **dbfunctions.php**.

Ниже перечислены преимущества логического разбиения кода на блоки:

- Упрощается чтение и понимание кода.
- Улучшаются возможности повторного использования кода и сводится к минимуму его избыточность. Например, указанный выше файл **dbfunctions.php** можно будет использовать в любом сценарии, где требуется подключение к базе данных. Если необходимо внести изменения в процесс подключения, достаточно это выполнить лишь в одном файле.
- Создаются предпосылки для совместного труда команды разработчиков. Когда код разбит на компоненты, можно возложить ответственность за каждый из них на членов команды. Кроме того, исключается ситуация, когда одному программисту для продолжения работы приходится ожидать, пока коллега завершит работу над файлом **GiantScript.php** ("гигантский сценарий").

На начальном этапе работы над проектом необходимо уделить время разбиению проекта на компоненты, которые включаются в график работ. Придется рисовать схему функциональных модулей, но не следует излишне вдаваться в детали, поскольку схема может изменяться в процессе работы над проектом. Кроме того, необходимо решить, какие компоненты должны создаваться в первую очередь, какие зависят от других компонентов, и построить график разработки каждого из них.

Даже если все члены группы должны работать над всеми составляющими кода, обычно имеет смысл возложить персональную ответственность за каждый компонент на определенное лицо. В конечном счете этот человек будет отвечать за неполадки, связанные с данным компонентом. Кроме того, кто-то должен возложить на себя обязанности руководителя работ (build manager). Руководитель должен обеспечить продвижение работ над всеми компонентами первой очереди и работать над остальными. Обычно это же лицо осуществляет управление версиями, о чем речь пойдет ниже. Данный сотрудник может быть назначен руководителем проекта либо обладать особыми полномочиями.

Использование стандартной структуры каталогов

Приступая к работе над проектом, необходимо продумать отражение структуры компонентов в структуре каталогов Web-сайта. Обычно содержать все компоненты в одном каталоге так же нецелесообразно, как и помещать все функции в один гигантский сценарий. Имеет смысл распределить каталоги между компонентами, логикой, содержимым и библиотеками общего кода. Структуру каталогов потребуется задокументировать и предоставить копию описания каждому разработчику проекта. Отсюда следующий момент.

Документирование и распределение функций собственной разработки

Создавая библиотеки функций, необходимо делать их доступными для других программистов команды. Обычно каждый программист создает собственный набор баз данных или функций отладки. Это приводит к непроизводительным затратам времени. Следует предоставлять доступ к функциям и классам другим членам команды.

Помните, даже если код хранится в общедоступном каталоге, сотрудники не будут знать об этом, если только им не рассказать. Создайте систему документирования библиотек функций собственной разработки и сделайте ее доступной для остальных программистов.

Управление версиями

Применительно к разработке программных средств, управление версиями — это искусство управления параллельными изменениями. Обычно системы управления версиями действуют как центральные *хранилища* или архивы, и предоставляют управляемый интерфейс для доступа и коллективного использования кода (и, возможно, документации).

Представьте ситуацию, когда двум членам группы необходимо работать над одним и тем же файлом. Они могут открыть и редактировать файл одновременно, перезаписывая изменения друг друга. Либо каждый из них располагает собственной копией файла и редактирует ее на свой лад в автономном режиме. Бывает также, что один программист просто бездействует в ожидании, пока другой не завершит редактирование файла.

Система управления версиями позволяет решить все эти проблемы.

Подобные системы способны отслеживать изменения каждого файла в архиве таким образом, чтобы пользователь мог видеть не только его текущее состояние, но и содержимое в любой момент времени в прошлом. Эта функция позволяет выполнять откат ошибочного кода, чтобы вернуться к работоспособной версии. Определенный набор экземпляров файла можно пометить как окончательную версию. Это означает, что можно продолжать разработку кода, но всегда иметь доступ к копии версии, которая на данный момент считается окончательной.

Кроме того, системы управления версиями помогают нескольким программистам одновременно работать над кодом. Каждый программист может получить копию кода в архиве (этот процесс называется *выдачей)*. После внесения изменений новый вариант можно поместить обратно в архив. В этом случае считается, что версия *принята* или *предоставлена*. Поэтому системы управления версиями могут отслеживать, какие пользователи вносят изменения.

Обычно подобные системы способны управлять одновременными обновлениями. Это означает, что два программиста могут одновременно модифицировать один и тот же файл. Например, Джон и Мэри получили по копии самой последней версии проекта. Джон завершает модификацию определенного файла и предоставляет его системе. Мэри также изменяет этот файл и пытается предоставить его. Если внесенные изменения касаются не одной и той же части файла, система выполнит слияние двух версий. Если изменения конфликтуют между собой, для Мэри выводится соответствующее уведомление и будут показаны две различные версии. Затем она сможет переделать свою версию кода, дабы устранить конфликты.

Система управления версиями, используемая большинством разработчиков UNIX и/или Open Source, называется CVS (Concurrent Versions System — система параллельных версий). Система CVS относится к категории программного обеспечения с открытым исходным кодом и входит в поставку практически каждой версии UNIX. Кроме того, ее можно приобрести для систем DOS, Windows и Macintosh. Она поддерживает модель клиент/сервер, поэтому можно просматривать код на любом компьютере через Internet-соединение, если сервер CVS доступен в сети. По этой причине система использовалась при разработке PHP, Apache, Mozilla и других крупных проектов, по крайней мере, частично.

CVS можно выгрузить из Web-сайта по адресу:

```
http://www.cvshome.org/
```

Хотя базовая версия CVS является инструментом командной строки, различные дополнения, включая модули на основе Java и Windows, реализуют для нее более привлекательный интерфейс. Дополнения также можно выгрузить из упомянутого выше сайта.

Существуют коммерческие альтернативы CVS. Одна из них — Microsoft Visual Source Safe — тесно интегрирована с продуктами Visual Studio.

Выбор среды разработки

Управление версиями можно развить в более широкую тему среды разработки. Для программирования абсолютно необходимы лишь текстовый редактор и браузер для целей тестирования. Однако многие пользователи PHP выражают заинтересованность в интегрированной среде разработки (IDE — integrated development environment). В настоящее время не существует совершенной IDE для программистов на PHP. Конечно, можно выгрузить либо написать собственное расширение, реализующее выделение элементов синтаксиса, для существующего редактора либо готовых пакетов Web-программирования. Многие из подобных продуктов доступны в сети (их слишком много, чтобы перечислять здесь — необходимо подбирать те, что соответствуют применяемому редактору).

Существует ряд незавершенных проектов специализированных IDE для PHP. На момент написания этой книги к ним относятся:

- Zend IDE для Windows или UNIX:

  ```
  http://www.zend.com/
  ```

  Эта система, построенная по принципу клиент/сервер, обладает обычными возможностями редактора, а также встроенными функциями отладки, такими как окна просмотра и точки останова. Требует лицензирования для сервера и каждого клиента. Индивидуальные лицензии либо лицензии для сайтов можно приобрести через Web-сайт.

- KPHPDevelop для K Desktop Environment под управлением Linux:

  ```
  http://kphpdev.sourceforge.net/
  ```

  Эта IDE разработана специально для групп, совместно работающих над проектом, которая использует механизм блокировки файлов. (В будущих версиях планируется реализовать поддержку CVS.) Она поддерживает выделение элементов синтаксиса и доступ к базам данных, таким как MySQL, PostgreSQL и Sybase.

- PHPCoder для платформ Win32:

  ```
  http://phpcoder.stsoft.cjb.net
  ```

  Эта среда разработки столь же эффективна, сколь и нова. Она поддерживает режим предварительного просмотра сценариев за счет подключения к выполняемому файлу PHP, а также объединения с документацией для простоты использования. Программа PHPCoder распространяется бесплатно.

- PHPEdit для платформ Win32:

  ```
  http://www.phpedit.com
  ```

На момент написания данной книги эта программа еще не выпущена, поэтому прокомментировать ее трудно. Планируется поддержка всех обычных функций и наличие встроенного руководства.

- PHPGem:

  `http://phpgem.ru.net:8100/`

  На основе предоставляемых параметров базы данных программа PHPGem генерирует базовый код интерфейса, который взаимодействует, помимо прочих, с MySQL и PostgreSQL. Программа PHPGem сама является PHP-сценарием, поэтому она может взаимодействовать с любой средой.

Несомненно, ведется работа и над другими проектами. Будет интересно узнать, какие из них займут лидирующие позиции.

Документирование проектов

Ниже приводится неполный список различных типов документации для программных проектов:

- Проектная документация
- Техническая документация/руководство разработчика
- Словарь данных (включая документацию по классам)
- Руководство пользователя (хотя большинство Web-приложений должны быть самоочевидными)

Здесь наша цель состоит не в том, чтобы обучать написанию документации, а в рекомендациях по снижению трудозатрат за счет частичной автоматизации процесса.

В некоторых языках существуют методы автоматической генерации некоторых из перечисленных документов — в частности, технической документации и словарей данных. Например, программа **javadoc** генерирует дерево HTML-файлов, содержащее модели и описания членов классов для Java-программ.

Для PHP существует немало утилит подобного рода. Ниже перечислены некоторые из них:

- **phpDoc:**

  `http://sourceforge.net/projects/phpdoc`

  Эта утилита сохраняет документацию в базе данных MySQL. Обратите внимание, что термин phpDoc используется для описания нескольких проектов данного типа.

- **PHPDocumentator:**

  `http://phpdocu.sourceforge.net`

  Позволяет получать документы, во многом подобные генерируемым программой **javadoc,** и довольно надежна в работе.

- **phpautodoc**:

  `http://sourceforge.net/projects/phpautodoc/`

  Эта программа также генерирует вывод, подобный утилите **javadoc.**

Для поиска других приложений данного типа (и PHP-компонентов вообще) хорошо подходит сайт SourceForge:

```
http://sourceforge.net
```

Продукты SourceForge, в основном, используются сообществом пользователей UNIX/Linux, но существует множество проектов и для других платформ.

Создание прототипов

Созданием прототипов называют определенный этап разработки, широко используемый для написания Web-приложений. Прототип служит удобным средством отработки требований заказчика. Обычно прототип представляет собой частично работающую версию приложения, которую можно обсуждать с клиентом и которая служит основой окончательной версии. Часто окончательная версия получается в результате многочисленных переработок модели. Преимущество такого подхода состоит в возможности тесного взаимодействия с клиентом или конечным пользователем с целью создания приемлемой системы. Кроме того, в какой-то мере клиент становится совладельцем продукта.

Чтобы быстро "склепать" модель, необходимы определенные навыки и утилиты. Вот где себя оправдывает модульный принцип проектирования. Наличие доступа к набору готовых компонентов существенно ускоряет построение модели. Другим удобным средством быстрого создания моделей служат шаблоны, которые рассматриваются в следующем разделе.

Построение моделей связано с двумя основными проблемами. Необходимо иметь о них представление, чтобы избегать затруднений, а также использовать этот подход с максимальной эффективностью.

Первая проблема связана с тем, что программисты часто находят затруднительным отбрасывать код, который они по той или иной причине написали. Модели обычно создаются быстро, но впоследствии становится очевидным, что прототип построен не самым оптимальным образом. Нелепые фрагменты кода еще можно исправить, но когда структура в целом неприемлема, положение становится серьезным. Дело в том, что Web-приложения обычно создаются в предельно сжатые сроки, и времени на исправление может попросту не остаться. В результате получается неудачно построенная система, которой трудно управлять.

Во избежание этого необходимо применить элементы планирования, о чем речь шла выше. Иногда проще что-то переделать заново, нежели пытаться исправить. Может показаться, что на планирование просто нет времени, но впоследствии оно избавит от множества забот.

Вторая проблема состоит в том, что система может превратиться в вечный прототип. Каждый раз когда, казалось бы, работа завершена, заказчик предлагает новые усовершенствования, дополнительные функциональные возможности и обновления сайта. Из-за такого наплыва требований проект, возможно, никогда не сможет быть завершен.

Во избежание подобной ситуации, составьте план с фиксированным количеством версий и датой, после истечения которой нельзя добавлять новые функции без повторного планирования, составления сметы и графика.

Разделение логики и содержимого

Возможно, вам уже знакома идея использования HTML для описания структуры Web-документов и применения *каскадных таблиц стилей* (CSS — cascading style sheets) для описания их оформления. Этот принцип отделения содержимого от представления можно распространить и на создание сценариев. Обычно долговременное содержание сайтов осуществляется проще, когда реализовано разграничение логики и содержимого. Это сводится к разделению PHP-кода и HTML-кода.

Для простых проектов с небольшим количеством строк кода эффект от реализации данного принципа может не оправдать затрачиваемых усилий. По мере разрастания проекта становится важным найти способ разграничения логики и содержимого. В противном случае кодом будет все сложнее и сложнее управлять. Если возникает необходимость изменить оформление Web-сайта, а элементы форматирования тесно связаны с HTML-кодом, подобная работа может превратиться в настоящий кошмар.

Ниже перечислены три основных подхода к разделению логики и содержимого:

- Для хранения различных частей содержимого используйте подключаемые файлы. Это упрощенный подход, но вполне эффективный, если сайт в основном статичный. Данное решение обсуждалось в примере для TLA Consulting в главе 5.
- Воспользуйтесь API-интерфейсом функций либо классов с набором собственных функций для подключения динамического содержимого к статическим шаблонам страницы. Такой подход рассматривался в главе 6.
- Воспользуйтесь системой шаблонов. Она анализирует статические шаблоны и применяет регулярные выражения для замены дескрипторов-заполнителей динамическими данными. Главное преимущество данного решения состоит в том, что проектировать шаблоны может кто-то другой, кому совершенно не требуется вникать в PHP-код. Необходимо иметь возможность использования предоставляемых шаблонов с минимальными изменениями.

Существует целый ряд систем шаблонов. Возможно, наиболее популярной из них является FastTemplate:

```
http://thewebmasters.net
```

Оптимизация кода

Для тех, чей опыт программирования не связан с Web, оптимизация может иметь большое значение. В случае использования языка PHP, большая часть времени ожидания пользователя Web-приложения связана с подключением и загрузкой из сети. По сравнению с этим эффект от оптимизации кода оказывается несущественным.

Использование простой оптимизации

Тем не менее, существует несколько простых методов оптимизации, которые создают ощутимый эффект. Многие из них связаны с приложениями, которые через PHP-код взаимодействуют с базами данных, такими как MySQL. Некоторые из методов перечисляются ниже:

- Уменьшение подключений к базам данных. Подключение к базе данных часто является самой медленно выполняемой частью любого сценария. Выйти из положения можно с помощью постоянных соединений.

- Ускорение запросов в базы данных. Необходимо уменьшать количество запросов и обеспечить их оптимизацию. Для сложных (а, следовательно, медленных) запросов обычно существует несколько методов оптимизации. Выполняйте запросы через интерфейс командной строки базы данных и пробуйте различные варианты их ускорения. В MySQL для выявления ошибочных запросов можно применять оператор **EXPLAIN.** (Использование этого оператора рассматривалось в главе 11.) Обычно идея заключается в сведении к минимуму количества соединений и максимальному использованию индексации.
- Сведение к минимуму генерации статического содержимого из PHP-кода. Если каждый фрагмент HTML-кода генерируется операторами **echo** или **print()**, приложение работает намного медленнее. (Это один из аргументов в пользу разграничения логики и содержимого.) То же относится к динамическому генерированию графических кнопок. Лучше сгенерировать все кнопки средствами PHP один раз, а затем повторно использовать их по мере необходимости. Если выполняется генерация статического содержимого из функций или шаблонов при каждой загрузке страницы, имеет смысл реализовать однократный вызов функций либо использование шаблонов и сохранение результатов.
- Использование функций обработки строк вместо регулярных выражений при любой возможности. Эти функции обладают более высоким быстродействием.
- Использование строк с одинарными кавычками вместо строк с двойными кавычками, где только это возможно. PHP выполняет анализ строк с двойными кавычками, выискивая переменные, имена которых необходимо заменить значениями. Строки с одинарными кавычками не анализируются. С другой стороны, одинарные кавычки чаще применяют для статического содержимого. Попытайтесь отыскать способ избавиться от строк вообще, заменив их статическим HTML-содержимым.

Использование продуктов компании Zend

Компания Zend Technologies разработала механизм сценариев (с открытым исходным кодом) PHP, начиная с версии 4. По быстродействию он намного (как заявлено, от двух до десяти раз) превышает механизм версии 3. Поэтому тем, кто еще не выполнил обновление, имеет смысл установить версию 4.

Помимо базового механизма можно загрузить утилиту Zend Optimizer. Эта программа в несколько проходов выполняет оптимизацию кода и может повысить его быстродействие от 40% до 100%. Для выполнения профаммы оптимизации необходима версия PHP 4.0.2 или выше. Исходный код профаммы не предоставляется, однако ее можно бесплатно выгрузить из Web-сайта Zend по адресу:

```
http://www.zend.com
```

Этот подключаемый модуль выполняет оптимизацию кода, получаемого в результате компиляции времени выполнения сценария. Ожидается выпуск нового продукта Zend Cache, который выполняет компиляцию PHP-сценариев и буферизирует их. Таким образом, повторная компиляция сценариев потребуется только в случае их изменения. Это позволит еще больше повысить быстродействие.

Тестирование

Просмотр и тестирование кода является еще одним важным этапом разработки программ, который часто упускают, занимаясь программированием для Web. Очень легко два-три раза запустить программу и отметить, что она нормально работает. Это распространенная ошибка. Прежде чем выпустить продукт, необходимо тщательно проанализировать и испытать несколько сценариев.

Мы рекомендуем два метода снижения количества ошибок в коде. (Полностью избавиться от ошибок никогда не удастся, но большинства из них вполне можно избежать.)

Во-первых, практикуйте проверку кода, когда его просматривает другой программист и предлагает усовершенствования. Подобный анализ часто выявляет:

- Допущенные ошибки
- Тестовые варианты, которые не учел разработчик
- Возможности оптимизации
- Возможности совершенствования степени защищенности
- Существующие компоненты, которые можно использовать для усовершенствования фрагментов кода
- Дополнительные функциональные возможности

Даже если код разрабатывается в одиночку, имеет смысл найти коллегу, находящегося в подобной ситуации, и анализировать код друг друга.

Второй метод предусматривает поиск лиц, которые выполнят тестирование Web-приложений, ставя себя на место конечных пользователей продукта. Главное отличие Web-приложений от настольных систем состоит в том, что с Web-приложениями работает самая широкая публика. Здесь не следует рассчитывать, что пользователи разбираются в компьютерной технике. Их нельзя снабдить длинным руководством либо кратким справочником. Вместо этого Web-приложения должны быть самодокументируемыми и самоочевидными. Необходимо учесть возможные способы использования приложения. Удобство работы с приложением имеет абсолютный приоритет.

Опытному программисту или пользователю Web иногда трудно понять проблемы неискушенных пользователей. Одно из решений — найти тестеров, которые представляют типовых пользователей.

В прошлом применялся следующий подход — сначала выпускались лишь бета-версии Web-приложений. Когда, как предполагалось, большинство ошибок было исправлено, приложение публиковалось для небольшой группы пользователей и невысокой интенсивности трафика сайта. Предложите первым 100 пользователям бесплатные услуги в обмен за отзывы о сайте. Гарантируем, что они укажут на такие комбинации данных или варианты использования, о которых разработчик даже и не подозревал. Если создание Web-сайта заказывает некоторая компания, она обычно может предоставить для его тестирования достаточно неопытных пользователей в лице своих сотрудников. (Существенное преимущество такого подхода состоит в укреплении чувства причастности к сайту со стороны клиента.)

Дополнительная информация

Искусству разработки программ посвящено очень много книг. Это исключительно обширная тема.

Противопоставление Web-сайта как документа и Web-сайта как приложения хорошо описывается в книге *Web Site Engineering: Beyond Web Page Design* Томаса Пауэла (Thomas A. Powell). Полезной может оказаться также и любая книга по разработке программ.

Информацию по управлению версиями можно найти на Web-сайте CVS по адресу:

```
http://www.cvshome.org
```

Управлению версиями посвящено не так много книг (несмотря на важность этой темы!), тем не менее, можно остановиться на *Open Source Development with CVS* Карла Франца Фогеля (Karl Franz Fogel) либо *CVS Pocket Reference* Грегора Пурди (Gregor N. Purdy).

PHP-компоненты, IDE и системы документирования можно найти на сайте SourceForge:

```
http://sourceforge.net
```

Многие темы этой главы обсуждаются в статьях Web-сайта компании Zend. Из них можно почерпнуть дополнительную информацию. Заодно имеет смысл выгрузить утилиту оптимизации.

```
http://www.zend.com
```

Если материал главы вас заинтересовал, рекомендуем ознакомиться с методологией разработки программного обеспечения, посвященной областям, где требования часто меняются, к которым относятся и Web-приложения. Адрес Web-сайта Extreme Programming:

```
http://www.extremeprogramming.org
```

Что дальше

В главе 23 рассматриваются различные виды ошибок программирования, сообщения об ошибках PHP, а также технология обнаружения и эффективного устранения неполадок.

Глава 23

Отладка

Эта глава посвящена отладке PHP-сценариев. Если вы разобрали некоторые примеры книги либо ранее работали с PHP, то наверняка выработали собственные навыки и методы отладки. С возрастанием сложности проектов отладка становится все более затруднительной. Навыки разработчика совершенствуются, но ошибки могут затрагивать несколько файлов либо появляться в результате взаимодействия кода, созданного разными программистами.

В главе рассматриваются следующие вопросы:

- Типы программных ошибок
  - Синтаксические ошибки
  - Ошибки времени выполнения
  - Логические ошибки
- Сообщения об ошибках
- Уровни ошибок
- Генерация собственных ошибок
- Эффективная обработка ошибок
- Удаленная отладка

Программные ошибки

Независимо от используемого языка существует три основных типа программных ошибок:

- Синтаксические ошибки
- Ошибки времени выполнения
- Логические ошибки

Ниже представлен краткий обзор каждого типа, после чего последует обсуждение тактики обнаружения, обработки, предупреждения и устранения ошибок.

Синтаксические ошибки

Языки характеризуются набором правил, называемых *синтаксисом,* который касается правильного использования выражений. Это относится как к естественным языкам, например, английскому, так и языкам программирования, вроде PHP. Если выражение не соответствует правилам языка, говорят, что оно содержит синтаксическую ошибку. Синтаксические ошибки часто называют ошибками разбора в случае интерпретируемых языков, таких как PHP, либо ошибками компиляции, когда речь идет о компилируемых языках, таких как C и Java.

Если мы нарушаем синтаксис английского языка, скорее всего, люди нас поймут. С языками программирования такое случается редко.

Когда в сценарии нарушается синтаксис PHP, программа анализа не сможет обработать сценарий частично или полностью. Люди хорошо умеют извлекать информацию из неполных или противоречивых данных. Компьютеры такой способностью не обладают.

Помимо прочего, синтаксис PHP требует, чтобы операторы завершались точкой с запятой, строки заключались в кавычки, а передаваемые функциям параметры отделялись запятыми и заключались в скобки. Если нарушить эти правила, сценарий окажется неработоспособным, а при первой же попытке его выполнения будет сгенерировано сообщение об ошибке.

Одна из сильных сторон PHP заключается в информативных сообщениях об ошибках. Обычно они указывают характер неполадки, какой файл содержит ошибку и в какой строке она обнаружена.

Ниже приводится пример сообщения об ошибке:

```
Parse error: parse error in
/home/book/public_html/chapter23/error.php on line 2
```

Эту ошибку сгенерировал следующий сценарий:

```
<?
    $date = date(m.d.y');
?>
```

Здесь предпринимается попытка передать в функцию date() строку, но по ошибке пропущена открывающая кавычка, которая отмечает начало строки.

Простые синтаксические ошибки, подобные данной, обычно легко обнаруживаются. Можно допустить подобную, однако более сложную в обнаружении ошибку, если не завершить строку кавычкой, как в следующем примере:

```
<?
    $date = date('m.d.y);
?>
```

Этот сценарий генерирует следующую синтаксическую ошибку:

```
Parse error: parse error in
/home/book/public_html/chapter23/error.php on line 4
```

Очевидно, ошибка не может содержаться в четвертой строке, поскольку сценарий состоит лишь из трех строк. Подобным образом генерируются сообщения об ошибках, когда пропущены закрывающие одинарные или двойные кавычки, а также скобки любого вида.

Следующий сценарий генерирует аналогичную синтаксическую ошибку:

```
<?
   if   (true)
   {
      echo  "error here";
?>
```

Обнаружение подобных ошибок может быть затруднено, если они появляются в результате комбинирования нескольких файлов. То же относится к случаю, когда ошибка содержится в файле большого размера. Сообщение об ошибке в строке 1001 **(parse error on line 1001)** файла, который содержит 1000 строк, может создать большие затруднения.

По общепринятому мнению, синтаксические ошибки наиболее просты в обнаружении. Сообщения PHP указывают, где их следует искать.

Ошибки времени выполнения

Ошибки времени выполнения обычно сложнее выявлять и исправлять. Синтаксические ошибки явно содержатся в сценарии и обнаруживаются программой анализа. Ошибки времени выполнения не вызываются исключительно содержимым сценария. Они могут зависеть от взаимодействия сценариев с другими событиями или условиями.

Следующее выражение

```
include  ("filename.php");
```

вполне допустимо. Оно не содержит синтаксических ошибок.

Однако оно может генерировать ошибку времени выполнения. Если его выполнить, когда файл **filename.php** не существует, либо выполняющему сценарий пользователю отказано в правах чтения, будет получено примерно такое сообщение об ошибке:

```
Fatal error: Failed opening required 'filename.php'
(include_path='.:/usr/local/lib/php') in
/home/book/public_html/chapter23/error.php on line 1
```

Код написан вполне корректно, но указываемый в нем файл в момент выполнения сценария может либо существовать либо отсутствовать. Поэтому возможна ошибка времени выполнения.

Следующие три строки являются допустимыми выражениями PHP. К сожалению, в совокупности они приводят к попытке выполнить невозможное действие — деление на нуль.

```
$i = 10;
$j = 0;
$k = $i/$k;
```

Этот фрагмент кода генерирует следующее предупреждение:

```
Warning: Division by zero in
/home/book/public_html/chapter23/div0.php on line 3
```

Оно существенно упрощает исправление ошибки. Не очень многие будут намеренно задавать в коде деление на нуль, но отсутствие проверки пользовательского ввода часто приводит к ошибкам подобного типа.

Это один из многочисленных примеров ошибок времени выполнения, которые могут всплывать в процессе тестирования кода.

Ниже перечислены распространенные причины ошибок времени выполнения:

- вызов несуществующих функций
- чтение или запись в файлы
- взаимодействие с MySQL и другими базами данных
- подключение к сетевым службам
- отсутствие проверки вводимых данных

Все причины будут кратко рассмотрены.

Вызов несуществующих функций

Подобную ошибку легко допустить случайно. Имена встроенных функций часто бывают неоднородными. Почему в имени **strip_tags()** применен символ подчеркивания, а в имени **stripslashes()** его нет?

Кроме того, возможен вызов собственных функций, которые не содержатся в текущем сценарии, но могут существовать в другом месте. Если код содержит вызов несуществующей функции, такой как

```
nonexistent_function();
```

будет получено примерно следующее сообщение об ошибке:

```
Fatal error: Call to undefined function: nonexistent_function()
in /home/book/public_html/chapter23/error.php on line 1
```

Точно так же, если вызвать существующую функцию, но с неверным количеством параметров, будет выведено предупреждение.

Функция **strstr()** требует передачи двух строк — в одной из них ведется поиск, а другая служит искомым фрагментом. Если вызвать функцию следующим образом:

```
strstr();
```

выведется следующее предупреждение:

```
Warning: Wrong parameter count for strstr() in
/home/book/public_html/chapter23/error.php on line 1
```

Поскольку PHP не допускает перегрузку функций, эта строка всегда является ошибочной, но предупреждение выводится не во всех случаях.

Следующий сценарий содержит такую же ошибку:

```
<?
   if($var == 4)
   {
      strstr();
   }
?>
```

Однако вызов функции **strstr()** не осуществляется за исключением тех случаев, когда переменная $var принимает значение *4*. При этом предупреждение также не выводится.

Неправильный вызов функции легко допустить, но сообщение об ошибке точно идентифицирует строку и имя функции, что делает исправление ошибки таким же простым. Обнаружение подобных ошибок затруднено лишь в случае, когда процесс тестирования несовершенен и не проверяется весь условно выполняемый код. Одна из задач тестирования состоит в выполнении каждой строки кода. Вторая задача — проверка всех граничных условий и классов ввода.

Чтение и запись в файлы

В процессе использования программы могут происходить любые ошибки, но одни из них случаются чаще других. Ошибки доступа к файлам достаточно вероятны, чтобы предусмотреть методы их эффективной обработки. Жесткие диски могут давать сбои либо переполняться, а ошибки пользователей приводят к изменению прав доступа к каталогам.

Обычно в часто приводящих к ошибкам функциях, таких как **fopen(),** предусматривается возвращаемое значение, указывающее на ошибку. Для функции **fopen()** таким значением является **false.**

Для подобных функций необходимо тщательно проверять возвращаемое значение при каждом вызове и обрабатывать ошибки.

Взаимодействие с MySQL и другими базами данных

Подключение к базе данных MySQL и ее использование может вызвать генерацию множества ошибок. Только функция **mysql_connect()** может приводить к следующим ошибкам:

- `MySQL Connection Failed: Can't connect to MySQL server on 'hostname' (111)`
- `MySQL Connection Failed: Can't connect to local MySQL server through socket '/tmp/mysql.sock' (111)`
- `MySQL Connection Failed: Unknown MySQL Server Host 'hostname' (2)`
- `MySQL Connection Failed: Access denied for user: 'username@localhost' (Using password: YES)`

Несложно догадаться, что функция **mysql_connect()** в случае ошибки возвращает значение **false.** Это означает, что данные типы распространенных ошибок легко отслеживать и обрабатывать.

Если не остановить нормальное выполнение сценария и не предусмотреть обработку ошибок, он попытается продолжить взаимодействие с базой данных. Попытка выполнения запросов и получения результатов без нормального соединения с MySQL заставит посетителей лицезреть непрофессионального вида экран, полный сообщений об ошибках.

Многие другие часто используемые PHP-функции, которые связаны с MySQL, также возвращают значение **false** в случае ошибки. К подобным функциям относятся **mysql_pconnect(), mysql_select_db() и mysql_query().**

Если ошибка все же возникает, для получения текста сообщения можно воспользоваться функцией **mysql_error(),** а для вывода кода ошибки — функцией **mysql_errno().** Если последняя функция MySQL не сгенерировала ошибки, **mysql_error()** возвращает пустую строку, а **mysql_errno()** — значение 0.

Предположим, что выполнено подключение к серверу и выбрана база данных для использования. Тогда следующий фрагмент кода:

```
$result = mysql_query( 'select * from does_not_exist' );
echo mysql_errno();
echo '<BR>';
echo mysql_error();
```

может сгенерировать следующее сообщение:

```
1146
Table 'dbname.does_not_exist' doesn't exist
```

Обратите внимание, что вывод указанных функций относится к выполнению последней функции MySQL (кроме **mysql_error()** и **mysql_errno()**). Если необходимо знать результаты выполнения команды, следует обязательно выполнить проверку до обращения к другим функциям.

Подобно сбоям доступа к файлам, возникают и сбои при взаимодействии с базами данных. Даже после тщательной разработки и тестирования службы случается, что демон MySQL (**mysqld**) дает сбой либо остается без доступных соединений. Если база данных содержится на другом компьютере, ее работа зависит от другого набора аппаратных и программных компонентов, которые могут сбоить. Это относится к сетевым соединениям, сетевым картам, маршрутизаторам и другим средствам связи между Web-сервером и компьютером с базой данных.

Необходимо проверять успешность запросов базы данных прежде чем использовать результаты. Нет смысла пытаться выполнить запрос после сбоя соединения с базой данных, а также извлекать и обрабатывать результаты запроса, которые не был успешно выполнен.

Здесь следует отметить различие между сбоем запроса и случаем, когда запрос просто не возвращает данные либо не изменяет какие-либо строки таблицы.

SQL-запрос, который содержит синтаксические ошибки SQL либо относится к несуществующим базам данных, таблицам или столбцам, может давать сбой. Например, следующий запрос:

```
select * from does_not_exist;
```

генерирует ошибку, поскольку таблица с именем **does_not_exist** не существует. Сообщение об ошибке и ее номер можно извлечь с помощью функций **mysql_errno()** и **mysql_error()**.

Синтаксически правильный SQL-запрос, который также адресуется только к существующим базам данных, таблицам и столбцам, обычно не генерирует ошибку. Тем не менее, если запрашивается пустая таблица, либо ведется поиск несуществующих данных, возврата результатов может не быть. Предположим, что выполнено успешное соединение с базой данных, а также существует таблица с именем exists и столбец с именем column name, следующий запрос:

```
select * from exists where column_name = 'not in database';
```

будет успешным, но возврата результатов не последует.

Прежде чем использовать результаты запроса, необходимо осуществить проверку на возможное присутствие ошибки, а также на предмет отсутствия возвращаемых данных.

Подключения к сетевым службам

Устройства и программы локальной системы могут давать сбой, но это бывает редко кроме случаев, когда они плохого качества. При использовании сети для подключения к другим компьютерам и программ, выполняемых на них, необходимо учитывать, что определенная часть системы будет часто давать сбой. При соединении между двумя компьютерами задействуются многочисленные устройства и службы, которые пользователем не контролируются.

Еще раз подчеркнем, что необходимо тщательно проверять значения, возвращаемые функциями, которые призваны взаимодействовать с сетевыми службами.

Следующий вызов функции:

```
$sp = fsockopen ( "localhost", 5000 );
```

не сгенерирует сообщение об ошибке в случае неуспешной попытки подключения к порту 5000 компьютера **localhost**.

Если переписать фрагмент кода следующим образом:

```
$sp = fsockopen ( "localhost", 5000, &$errorno, &$errorstr );
if(!$sp)
   echo "ERROR: $errorno: $errorstr";
```

будет проверяться возвращаемое значение на предмет возникновения ошибки и отображаться сообщение, которое поможет ее исправить. В этом случае вывод будет таким:

```
ERROR: 111: Connection refused
```

Ошибки времени выполнения сложнее устранить по сравнению с синтаксическими, поскольку программа анализа (интерпретатор) не может их выявить при первом выполнении кода. Поскольку ошибки времени выполнения возникают в результате комбинации событий, могут возникать трудности в их обнаружении и устранении. Программа анализа не может автоматически указать, что определенная строка сгенерирует ошибку. Задача тестирования состоит в том, чтобы создать одну из ситуаций, приводящих к генерации ошибки.

Предупреждение ошибок времени выполнения требует в некоторой степени предвидения возможных ошибочных ситуаций с тем, чтобы предпринять соответствующие действия. Кроме того, необходимо тщательное тестирование с имитацией каждого класса ошибок времени выполнения, которые могут произойти.

Это не означает, что необходимо пытаться имитировать все возможные ошибки. Например, MySQL может генерировать около 200 различных номеров ошибок и сообщений. Следует имитировать ошибки в каждом вызове функции, которая может вызвать сбой, а также ошибки каждого типа, обрабатываемого отдельным блоком кода.

Отсутствие проверки вводимых данных

Мы часто делаем предположения относительно вводимых пользователями данных. Если эти данные не оправдывают наших ожиданий, они могут вызвать ошибку — времени выполнения либо логическую.

Классический пример ошибки времени выполнения состоит в обработке вводимых пользователем данных, когда к ним забывают применить функцию **AddSlashes()** (добавления слешей). В таких случаях, если имя пользователя содержит апостроф, например, O'Grady, функция базы данных сгенерирует ошибку.

Ошибки в результате предположений относительно вводимых данных более подробно рассматриваются в следующем разделе.

Логические ошибки

Логические ошибки могут оказаться наиболее трудными для обнаружения и устранения. К ним относятся случаи, когда вполне допустимый код выполняется как запрограммировано, но автор кода имел другие намерения.

Логические ошибки могут вызываться простыми опечатками, как в следующем примере:

```
for ( $i = 0; $i < 10; $i++ );
{
   echo "doing something<BR>";
}
```

Этот фрагмент кода вполне допустим. Синтаксис PHP здесь не нарушается. Какие-либо внешние службы не задействуются, поэтому ошибки времени выполнения маловероятны. Код выполняет не те действия, которые могут показаться заданными на первый взгляд.

Может показаться, что строка **"doing something"** должна выводиться в цикле for 10 раз. Наличие лишней точки с запятой в конце первой строки означает, что цикл не распространяется на последующие строки. Цикл for будет выполнен 10 раз безрезультатно, а затем один раз будет выполнен оператор **echo.**

Поскольку этот код вполне допустим, хотя и не дает эффекта, программа анализа не выведет никаких сообщений. Компьютеры неплохо справляются с определенными задачами, но они не обладают здравым смыслом или интеллектом. Машина в точности выполняет то, что ей указывают. Необходимо добиться, чтобы инструкции точно соответствовали замыслу разработчика.

Логические ошибки не означают невозможность выполнения кода, а вызываются лишь неуспешной попыткой программиста точно выразить посредством кода свои намерения. Поэтому они не могут обнаруживаться автоматически. Ни код ошибки, ни сообщение о ней не выводятся. Логические ошибки выявляются только в результате тщательного тестирования.

Ошибку, подобную рассмотренной в предыдущем примере, легко допустить, однако столь же легко исправить. Дело в том, что при первом же выполнении кода вывод будет отличаться от ожидаемого. Большинство логических ошибок гораздо более коварно.

Логические ошибки, вызывающие затруднения, обычно получаются в результате неверных предположений разработчиков. В главе 22 рекомендуется задействовать других программистов для проверки кода и подсказки дополнительных тестовых вариантов, а также привлекать на тестирование представителей конечных пользователей вместо разработчиков. Когда разработчик выполняет тестирование самостоятельно, весьма вероятно, что код останется построенным в предположении, что пользователи будут вводить лишь данные определенного типа.

Предположим, что коммерческий сайт содержит текстовое поле Order Quantity (размер заказа). Можно ли предположить, что пользователи будут вводить только положительные числа? Если посетитель введет "минус десять", будет ли программа увеличивать остаток кредитной карточки на сумму десятикратной стоимости данного товара?

Предположим, форма содержит поле ввода суммы в долларах. Допускается ли ввод суммы со знаком доллара, либо без него? Допускается ли разделение тысяч запятыми? Некоторые проверки можно выполнять на стороне клиента (например, с помощью JavaScript), чтобы несколько разгрузить сервер.

Если информация передается другой странице, возможна ли ситуация, когда передаваемая строка содержит недопустимые символы для URL-адреса, такие как пробелы?

Количество возможных логических ошибок не ограничено. Не существует автоматизированного метода их выявления. Единственное решение — во-первых, избегать в коде сценария явных предположений о вводе пользователя и, во-вторых, тщательно проверять все возможные типы допустимого и недопустимого ввода, чтобы в любом случае получался ожидаемый результат.

Вспомогательное средство отладки переменных

С возрастанием сложности проектов возникает необходимость в утилите, которая помогает выявлять причины ошибок. Листинг 23.1 содержит фрагмент кода, который может оказаться полезным. Этот код выводит содержимое переменных, передаваемых странице.

Листинг 23.1 dump_variables.php — этот код может быть включен в страницы для вывода содержимого переменных для целей отладки

```
<?
   // в этих строках вывод форматируется в виде HTML-
   // комментариев и последовательно вызывается массив
   echo "\n<!-- BEGIN VARIABLE DUMP -->\n\n";
   echo "<!-- BEGIN GET VARS -->\n";
   echo "<!-- ".dump_array($HTTP_GET_VARS)." -->\n";
   echo "<!-- BEGIN POST VARS -->\n";
   echo "<!-- ".dump_array($HTTP_POST_VARS)." -->\n";
   echo "<!-- BEGIN SESSION VARS -->\n";
   echo "<!-- ".dump_array($HTTP_SESSION_VARS)." -->\n";
   echo "<!-- BEGIN COOKIE VARS -->\n";
   echo "<!-- ".dump_array($HTTP_COOKIE_VARS)." -->\n";
   echo "\n<!-- END VARIABLE DUMP -->\n";
// dump_array() принимает массив в качестве параметра
// функция выполняет проход по нему, создавая строку для
// представления массива в качестве набора элементов
function dump_array($array)
{
   if(is_array($array))
   {
      $size = count($array);
      $string = "";
      if($size)
      {
         $count = 0;
         $string .= "{ ";
         // добавление ключа и значения каждого элемента в массив
         foreach($array as $var => $value)
         {
            $string .= "$var = $value";
            if($count++ < ($size-1))
            {
               $string .= ", ";
            }
         }
         $string .= " }";
      }
      return $string;
   }
   else
   {
      // если параметр не является массивом, он просто возвращается
      return $array;
   }
}
?>
```

Этот код выполняет обход четырех массивов переменных, принимаемых страницей. Если страница вызывается с GET-переменными, POST-переменными, cookie-наборами либо переменными сеанса, все они будут задействованы в выводе.

Для каждого типа переменных генерируется HTML-строка.

Получаемые строки имеют примерно следующий вид:

```
<!-- { var1 = 'value1', var2 = 'value2', var3 = 'value3' } -->
```

Выводимые значения заключены в HTML-дескрипторы комментариев, чтобы они были доступными для просмотра, но не конфликтовали с методами обработки браузером видимых элементов страницы. Это хороший способ вывода отладочной информации.

Вывод зависит от передаваемых странице переменных. Если код утилиты добавить в листинг 20.4, один из примеров аутентификации из главы 20, сценарий дополнительно сгенерирует следующие строки:

```
<!-- BEGIN VARIABLE DUMP -->
<!-- BEGIN GET VARS -->
<!--  -->
<!-- BEGIN POST VARS -->
<!-- { userid = testuser, password = test123 } -->
<!-- BEGIN SESSION VARS -->
<!-- { valid_user = testuser } -->
<!-- BEGIN COOKIE VARS -->
<!-- { PHPSESSID = 2552dc82bb465af56d65e9300f75fd68 } -->
<!-- END VARIABLE DUMP -->
```

Можно заметить, что отображаются POST-переменные, отправленные из регистрационной формы предыдущей страницы — **userid и password.** Кроме того, выводится содержимое переменной сеанса, используемое для хранения имени пользователя — **valid_user**. Как упоминалось в главе 20, PHP использует cookie-набор для связывания переменных сеанса с определенными пользователями. Сценарий выводит псевдослучайное число **PHPSESSID,** хранимое в данном ключе для идентификации определенного пользователя.

Уровни сообщений об ошибках

PHP позволяет устанавливать степень тщательности обработки ошибок. Можно задавать типы событий, генерирующие сообщения. По умолчанию PHP выводит сообщения обо всех ошибках, которые не являются уведомлениями.

Для установки уровня сообщений используется набор предопределенных констант, перечисленных в табл. 23.1.

Каждая константа представляет тип ошибок, генерирующих сообщение либо игнорируемых. Например, если указать уровень ошибок **E_ERROR**, будут выводиться сообщения только о неисправимых ошибках. Допускается объединение констант методами двоичной арифметики для получения различных уровней сообщений об ошибках.

Устанавливаемый по умолчанию уровень (все сообщения кроме уведомлений) указывается следующим образом:

```
E_ALL & ~E_NOTICE
```

Это выражение включает две предопределенных константы, объединенных с помощью операторов побитовых арифметических действий. Амперсанд (&) задает битовую операцию AND, а тильда (~) — битовую операцию NOT. Выражение можно прочитать так: E_ALL AND NOT E_NOTICE.

Таблица 23.1 Константы уровней сообщений об ошибках

Значение	*Имя*	***Описание***
1	E_ERROR	Сообщения о неисправимых ошибках времени выполнения
2	E_WARNING	Сообщения об исправимых ошибках времени выполнения
4	E_PARSE	Сообщения об ошибках программы анализа
8	E_NOTICE	Уведомления и предупреждения, что выполненные действия могут быть ошибочными
16	E_CORE_ERROR	Сообщения о сбоях запуска механизма PHP
32	E_CORE_WARNING	Сообщения об исправимых ошибках в процессе запуска механизма PHP
64	E_COMPILE_ERROR	Сообщения об ошибках компиляции
128	E_COMPILE_WARN!NG	Сообщения об исправимых ошибках компиляции
256	E_USER_ERROR	Сообщения о сгенерированных пользователем ошибках
512	E_USER_WARNING	Сообщения о сгенерированных пользователем предупреждениях
1024	E_USER_NOTICE	Сообщения о сгенерированных пользователем уведомлениях
2048	E_ALL	Сообщения обо всех ошибках и предупреждениях

Константа **E_ALL** представляет собой комбинацию всех типов ошибок. Ее можно заменить, связав все остальные константы битовыми операциями OR (|):

```
E_ERROR | E_WARNING | E_PARSE | E_NOTICE | E_CORE_ERROR | E_CORE_WARNING |
E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING |
E_USER_NOTICE
```

Подобным же образом реализуется устанавливаемый по умолчанию уровень сообщений за исключением того, что отсутствует уровень уведомлений (константа **E_NOTICE**):

```
E_ERROR | E_WARNING | E_PARSE | E_CORE_ERROR |
E_CORE_WARNING | E_COMPILE_ERROR |
E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE
```

Изменение настроек сообщений об ошибках

Настройки сообщений об ошибках могут изменяться глобально в файле **php.ini** либо для каждого сценария в отдельности.

Чтобы изменить уровень сообщений для всех сценариев, необходимо модифицировать следующие четыре строки стандартного файла **php.ini:**

```
error_reporting     =     E_ALL & ~E_NOTICE
display_errors      =     On
log_errors          =     Off
track_errors        =     Off
```

Стандартные глобальные настройки задают:

- Вывод всех сообщений, кроме уведомлений
- Направление сообщений об ошибках в виде HTML на стандартное устройство вывода

- Отсутствие протоколирования сообщений на диске
- Отсутствие отслеживания ошибок, сохранение сообщений в переменной **$php_errormsg**

Чаще всего уровень сообщений изменяют таким образом, чтобы он соответствовал константе **E_ALL**. В результате будет выводиться множество уведомлений. Они могут указывать не только на ошибки, но и на неэффективное использование возможностей PHP, а также на свойства языка автоматически присваивать переменным значение О в процессе инициализации.

Иногда во время отладки имеет смысл устанавливать более высокий уровень сообщений (**error_reporting**). Если разработчик самостоятельно создает информативные сообщения об ошибках, в окончательном варианте кода имеет смысл отключить опцию отображения сообщений на экране (**display_errors**) и задействовать опцию протоколирования ошибок (**log_errors**). При этом уровень сообщений остается высоким. В случае сбоев можно будет просмотреть подробную информацию в файлах протоколов. Вместе с тем поведение программы будет восприниматься как более профессиональное.

Включение опции отслеживания ошибок (**track_errors**) помогает выявлять ошибки в собственном коде вместо того, чтобы позволить среде PHP предоставлять стандартные функциональные возможности. Хотя PHP выводит информативные сообщения об ошибках, когда возникают сбои, стандартное поведение среды выглядит непривлекательно.

Когда происходит неисправимая ошибка, по умолчанию PHP выводит следующее:

```
<br>
<b>Error Type</b>:  error message in <b>path/file.php</b>
on line <b>lineNumber</b><br>
```

и прекращает выполнение сценария. В случае исправимой ошибки выводится тот же текст, но выполнение сценария может продолжаться.

Выводимый HTML-код описывает ошибку, но выглядит непрофессионально. Стиль сообщения об ошибке вряд ли будет соответствовать оформлению сайта. Кроме того, если содержимое страницы отображается в таблице, пользователи Netscape могут вообще не увидеть вывода. Дело в том, что HTML-код, который открывает, но не закрывает элементы таблицы, например:

```
<table>
<trxtd>
<br>
<b>Error Type</b>:  error message in <b>path/file.php</b>
on line <b>lineNumber</b><br>
```

отображается в Netscape в виде пустого экрана.

Не обязательно сохранять стандартный режим обработки ошибок PHP либо даже использовать одинаковые настройки для всех файлов. Для изменения уровня сообщений об ошибках в текущем сценарии можно обратиться к функции **error_reporting()**.

Передача константы либо комбинации констант в эту функцию устанавливает уровень таким же образом, что и аналогичная директива файла **php.ini.** Функция возвращает предыдущий уровень сообщений. Ниже приводится распространенный метод использования функции:

```
// отключение сообщений об ошибках
$old_level = error_reporting(0);
// здесь помещается код, который генерирует предупреждения
// повторное включение режима сообщений об ошибках
error_reporting($old_level);
```

В этом фрагменте кода отключается режим вывода сообщений, что позволяет выполнять код, способный генерировать предупреждения, который нежелательно отображать.

Не стоит отключать сообщения навсегда, поскольку это затруднит поиск и исправление ошибок.

Генерация собственных ошибок

Для генерации собственных ошибок применяется функция **trigger_error()**. Созданные таким образом ошибки будут обрабатываться так же, как и обычные ошибки PHP.

Функции необходимо передать сообщение об ошибке, а также, необязательно, тип ошибки. Возможен один из следующих типов: **E_USER_ERROR**, **E_USER_WARNING** либо **E_USER_NOTICE.** Если тип не указан, по умолчанию принимается значение **E_USER_NOTICE.**

Ниже приводится пример использования функции **trigger_error()**:

```
trigger_error("This computer will self destruct in 15 seconds",
    E_USER_WARNING);
```

(Обратите внимание, что эта функция появилась только в версии PHP 4.0.1.)

Эффективная обработка ошибок

В языке PHP можно избегать обработки исключений, характерной для программирования на C++ и Java. Исключения позволяют функциям предупреждать об ошибках и задействовать обработчики исключений. Хотя в PHP исключения не существуют, версия 4.0.1 обладает механизмом, который может применяться очень похожим образом.

Как уже упоминалось, можно генерировать собственные ошибки, а также поддерживать собственные обработчики ошибок.

Функция **set_error_handler()** дает возможность предоставлять функцию, вызываемую когда происходят ошибки уровня пользователя, предупреждения и уведомления. При вызове **set_error_handler()** указывается имя функции, которая будет использоваться в качестве обработчика ошибок.

Функция обработки ошибок должна принимать два параметра — тип ошибки и сообщение. В зависимости от этих двух переменных, функция может выбирать способ обработки ошибки. Тип ошибки должен соответствовать одной из предопределенных констант. Сообщение представляет собой строку описания.

Ниже приводится пример вызова функции **set_error_handler()**:

```
set_error_handler("myErrorHandler");
```

Указав среде PHP использовать **myErrorHandler()**, необходимо затем подготовить функцию с таким именем. Ниже представлен прототип функции:

```
myErrorHandler(int error_type, string error_msg)
```

Реализуемые ею операции определяются разработчиком.

Возможны следующие логические действия:

- Отображение заданного сообщения об ошибке (**error_msg**)
- Сохранение информации в журнальном файле
- Отправка сообщения об ошибке по заданному адресу электронной почты
- Завершение сценария с помощью оператора **exit**

Листинг 23.2 содержит сценарий, в котором объявляется обработчик ошибок, устанавливается обработчик ошибок с помощью функции **set_error_handler()**, а затем генерируются ошибки.

Листинг 23.2 handle.php — этот сценарий объявляет пользовательский обработчик ошибок и генерирует различные ошибки

```
<?
   // Функция обработки ошибок
function myErrorHandler  ($errno,  $errstr)
   {
      echo "<br><table bgcolor = '#cccccc'><tr><td>
            <PXB>ERROR:</B>  $errstr
            <P>Please  try  again,  or  contact  us  and  tell  us  that
            the  error  occurred  in  line  ".__LINE__." of file  '".__FILE__."'";
      if  ($errno  ==  E_USER_ERROR| |$errno  ==  E_ERROR)
      {
         echo  "<P>This error was fatal, program ending";
         echo "</td></tr></table>";
         //Закрыть открытые ресурсы, включая колонтитул страницы и т.п.
         exit;
      }
      echo "</td></tr></table>";
   }
   // Установка обработчика ошибок
   set_error_handler("myErrorHandler");

   //Генерирование различных уровней ошибок
   trigger_error("Trigger function called", E_USER_NOTICE);
   fopen("nofile", "r");
   trigger_error("This computer is beige", E_USER_WARNING);
   include  ("nofile");
   trigger_error("This computer will self destruct in 15 seconds",
     E_USER_ERROR);
?>
```

Вывод сценария показан на рис. 23.1.

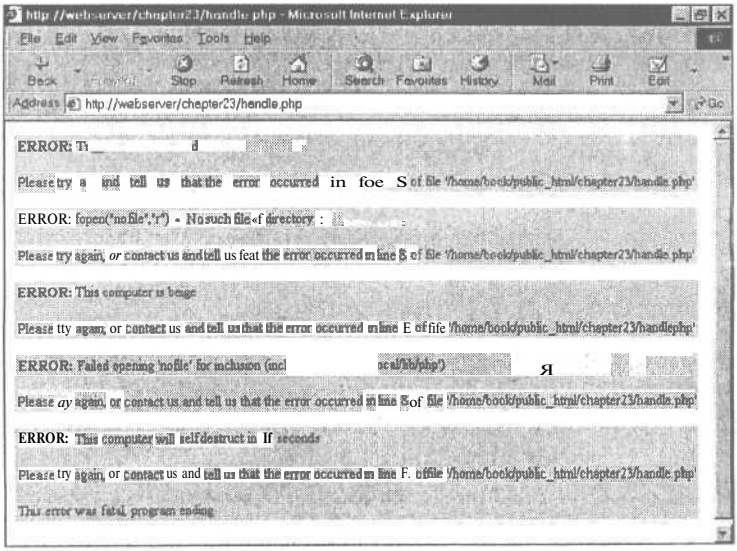


РИСУНОК 23.1
Используя собственный обработчик ошибок, можно выводить более понятные сообщения.

Данный пользовательский обработчик ошибок реализует лишь стандартное поведение. Поскольку этот код создается разработчиком, в нем можно предпринять любые действия. Это дает возможность сообщить посетителям страницы о возникших неполадках, а также представить информацию так, чтобы она соответствовала оформлению сайта. Что еще важнее, предоставляется гибкость выбора дальнейших действий. Должен ли сценарий продолжать выполнение? Сообщение будет протоколироваться или отображаться? Необходимо ли автоматическое уведомление службы технической поддержки?

Важно отметить, что устанавливаемый обработчик ошибок не обязательно должен охватывать все типы ошибок. Некоторые из них, такие как ошибки интерпретатора и неисправимые ошибки времени выполнения, могут по-прежнему вызывать стандартную реакцию. Если это важно, следует тщательно проверять параметры до передачи их в функцию, которая может генерировать неисправимые ошибки, и запускать собственный уровень **E_USER_ERROR**, если параметры вызывают сбой.

Удаленная отладка

В версию PHP 3 включен удаленный отладчик. Его назначение состоит в том, чтобы во время выполнения кода направлять подробную информацию об ошибках и событиях в порт, который можно контролировать.

Эта функция недоступна в версии 4, если только не приобрести интегрированную среду разработки Zend.

Если вы по-прежнему используете версию PHP 3 и возникла коварная ошибка, которую невозможно отследить другим способом, имеет смысл найти директиву конфигурации PHP **--enable-debugger**, а также параметры файла **php.ini: debugger.host**, **debugger.port и debugger.enabled**.

Если удаленный отладчик задействован, выводятся сотни строк информации, точно указывающей события, вызванные сценарием.

Что дальше

В главе 24 мы начнем работу над первым проектом. Этот проект демонстрирует методы распознавания пользователей и настройки содержимого сайта в соответствии с их потребностями.

Глава 24

Аутентификация и персонализация пользователей

В этом проекте будет реализована регистрация пользователей на Web-сайте. После этого станет возможным отслеживание интересов посетителей и отображение для них соответствующего содержимого. Последнее называется персонализацией пользователей.

Данный проект дает пользователям возможность создать в Web набор закладок (bookmarks) и предложить другие ссылки, которые могут заинтересовать посетителей, исходя из их поведения в предыдущих сеансах. В более обобщенном виде персонализация пользователей может применяться практически в любом Web-приложении, чтобы отобразить для них желаемое содержимое в предпочитаемом формате.

В этом, а также в последующих проектах мы начнем с обзора набора требований, подобных тем, что выдвигает клиент. Мы преобразуем эти требования в набор компонентов решения, построим схему их объединения, а затем реализуем каждый компонент.

Проект реализует следующие функциональные возможности:

- Регистрация и аутентификация пользователей
- Управление паролями
- Запись предпочтений пользователей
- Персонализация содержимого
- Рекомендация содержимого в зависимости от имеющихся сведений о пользователе

Задача

Требуется создать прототип интерактивной системы закладок, называемой PHPBookmark и подобной (но более функционально ограниченной) системе, разработанной Backflip:

```
http://backflip.com
```

Наша система должна предоставлять пользователям возможность входить в нее и хранить персональные закладки, а также получать рекомендации относительно других сайтов, подобранных на основе предпочтений посетителей.

Требования к системе можно разбить на три основных группы.

Во-первых, необходимо иметь возможность идентифицировать отдельных пользователей. Кроме того, следует располагать методом их аутентификации.

Во-вторых, необходимо иметь возможность хранения закладок для отдельного пользователя. Пользователи должны иметь возможность добавлять и удалять закладки.

В-третьих, требуется располагать способом рекомендации пользователям сайтов исходя из доступных сведений о клиентах.

Компоненты решения

Теперь, когда требования к системе определены, можно приступить к разработке решения и его компонентов. Рассмотрим возможные решения для каждого из трех главных требований, перечисленных ранее.

Идентификация и персонализация пользователей

Как упоминалось ранее, существует несколько альтернатив аутентификации пользователей. Поскольку требуется связать с пользователем некоторую личную информацию, его входное имя и пароль будут храниться в базе данных MySQL и применяться для аутентификации.

Если необходимо предоставить пользователям возможность входить в систему, указывая свое имя и пароль, возникает потребность в следующих компонентах:

- Пользователи должны иметь возможность регистрировать имя и пароль. Необходимо определить ограничения относительно длины и формата имени и пароля. Из соображений безопасности пароли следует хранить в зашифрованном виде.
- Пользователям необходимо позволить входить в систему с указанием сведений, которые они предоставили в процессе регистрации.
- Пользователи должны иметь возможность выходить из системы после завершения работы с сайтом. Это не особенно важно для лиц, посещающих сайт из домашних компьютеров, но очень существенно с точки зрения защищенности, когда применяются компьютеры общего пользования.
- Для сайта необходима возможность проверки, вошел ли пользователь в систему, а также предоставления данных тем, кто эту процедуру выполнил.
- Пользователи должны иметь возможность изменять пароль с целью усиления защищенности.
- Пользователи иногда забывают свои пароли. Им следует предоставить возможность переустанавливать пароль без помощи администратора. Обычный метод состоит в отправке пароля пользователю по адресу электронной почты, указанному при регистрации. Это означает необходимость сохранения адреса электронной почты в процессе регистрации. Поскольку пароли хранятся в зашифрованном виде и дешифрация их невозможна, на самом деле потребуется сгенерировать новый пароль и отправить его пользователю.

Мы создадим функции для реализации всех перечисленных возможностей. Большинство функций могут повторно использоваться без изменений либо с небольшими изменениями в других проектах.

Хранение закладок

Для хранения закладок пользователя требуется создать некоторое пространство в базе данных MySQL. Необходимо реализовать следующие возможности:

- Пользователи могут извлекать и просматривать свои закладки.
- Пользователи могут добавлять новые закладки. Необходимо проверять, что они являются допустимыми URL-адресами.
- Пользователи могут удалять закладки.

Рекомендация закладок

Для рекомендации закладок пользователю можно применять различные подходы. Можно выбирать наиболее популярные либо самые популярные в конкретной области закладки. В данном проекте будет реализована система рекомендаций, основанная на принципе "подобия образа мышления". Эта система выполняет поиск пользователей, имеющих ту же закладку, что и у вошедшего в систему клиента, и предлагает ему остальные закладки этих пользователей. Чтобы не рекомендовать закладки, соответствующие индивидуальным особенностям, выбираются лишь те из них, которые хранятся более чем у одного пользователя.

Для реализации упомянутых возможностей можно написать еще одну функцию.

Обзор проекта

После составления ряда эскизов получилась блок-схема, показанная на рис. 24.1.



РИСУНОК 24.1 *Эта диаграмма показывает возможные пути прохождения в системе PHPBookmark.*

Для каждого элемента диаграммы будет построен модуль. Некоторые из них потребуют одного сценария, а другие — двух. Кроме того, будут установлены библиотеки функций для:

- Аутентификации пользователей.
- Хранения и извлечения закладок.
- Проверки данных на допустимость.
- Соединений с базой данных.
- Вывода в окно браузера. Генерация HTML-кода будет возложена на библиотеку функций. Это обеспечит единообразие визуального представления в рамках всего сайта. (В этом и состоит принцип API — разделение логики и содержимого.)

Кроме того, потребуется создать серверную базу данных для системы.

Проект будет рассматриваться очень подробно; полный исходный код приложения можно найти в каталоге **chapter24** на CD-ROM. Перечень подключаемых файлов приводится в табл. 24.1.

Таблица 24,1 Файлы приложения PHPBookmark

Имя файла	Описание
bookmarks.sql	SQL-операторы для создания базы данных PHPBookmark
login.php	Титульная страница система с формой входа в систему
register_form.php	Форма регистрации пользователей в системе
register_new.php	Сценарий обработки новых регистрационных записей
forgot_form.php	Форма, заполняемая пользователями, забывшими пароль
forgot_passwd.php	Сценарий переустановки забытых паролей
member.php	Главная страница пользователей с окном просмотра всех текущих закладок
add_bm_form.php	Форма для добавления новых закладок
add_bms.php	Сценарий добавления новых закладок в базу данных
delete_bms.php	Сценарий удаления выбранных закладок из списка пользователей
recommend.php	Сценарий выдачи рекомендаций, основанных на пользователях со сходными интересами
change_passwd_form.php	Форма, заполняемая желающими сменить пароль
change_passwd.php	Сценарий смены пароля в базе данных
logout.php	Сценарий выхода пользователя из приложения
bookmark_fns.php	Набор подключаемых модулей для приложения
data_valid_fns.php	Функции проверки допустимости вводимых пользователем данных
db_fns.php	Функции для подключения к базе данных
user_auth_fns.php	Функции аутентификации пользователей
url_fns.php	Функции добавления и удаления закладок, а также выработки рекомендаций
output_fns.php	Функции, форматирующие вывод в виде HTML-кода
bookmark.gif	Логотип программы PHPBookmark

Начнем с реализации базы данных MySQL, поскольку она необходима для осуществления почти всех функциональных возможностей приложения.

Затем мы приступим к исследованию кода в порядке его написания. Начнем с титульной страницы, перейдем к аутентификации пользователей, хранению и извлечению закладок, и завершим выработкой рекомендаций. Эта последовательность вполне логична — определяются зависимости и создаются в первую очередь элементы, которые впоследствии понадобятся для других модулей.

ПРИМЕЧАНИЕ

Чтобы код проекта работал в соответствии с описанием, необходимо включить режим magic quotes. Если это не выполнено, ко вводимым данным в базу данных MySQL потребуется применить функцию **addslashes()**, а к извлекаемым данным — функцию **stripslashesl).** Этот режим уже использовался в качестве удобного сокращения.

Реализация базы данных

Для базы данных PHPBookmark используется достаточно простая схема. Необходимо хранение имен пользователей, их адресов электронной почты и паролей. Кроме того, следует хранить URL-адреса закладок. Один пользователь может иметь множество закладок, а одну и ту же закладку может зарегистрировать несколько пользователей. Поэтому база данных содержит две таблицы — пользователей и закладок, как показано на рис. 24.2.

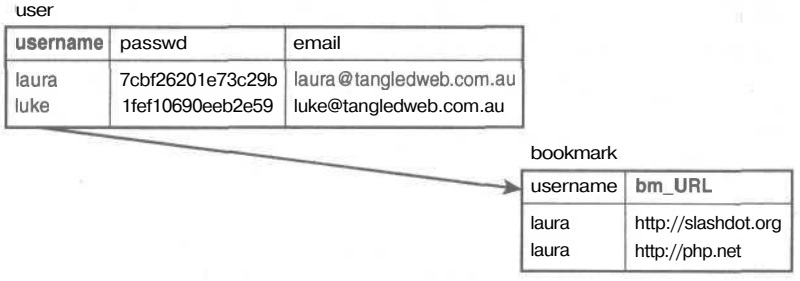


РИСУНОК 24.2 *Схема базы данных для системы PHPBookmark.*

Таблица пользователей содержит имена пользователей (как первичный ключ), пароли и адреса электронной почты.

Таблица закладок содержит пары имен пользователей и закладок (bm_URL). Имена пользователей этой таблицы ссылаются на соответствующие значения таблицы пользователей.

Листинг 24.1 содержит SQL-код для создания этой базы данных, а также пользователя для подключения к ней из Web. Если этот код планируется применить в своей системе, его следует отредактировать — заменить пароль пользователя более надежным!

Листинг 24.1 bookmark.sql — SQL-файл для создания базы данных закладок

```
create database bookmarks;
use bookmarks;

create table user   (
  username varchar(16)  primary key,
  passwd char(16) not null,
  email varchar(100)  not null
);
```

```
create table bookmark (
   username varchar(16) not null,
   bm_URL varchar(255) not null,
   index (username),
   index (bm_URL)
);

grant select, insert, update, delete
on bookmarks.*
to bm_user@localhost identified by 'password';
```

Эту базу данных можно создать, выполнив данный набор команд после регистрации в качестве привилегированного (root) пользователя MySQL. Этого можно достичь с помощью следующей командной строки:

```
mysql -u root -p < bookmarks.sql
```

Затем будет предложено ввести пароль.

Теперь, когда база данных готова, приступим к построению основы сайта.

Реализация основы сайта

Первая страница будет называться login.php, поскольку предоставляет пользователям возможность входа в систему. Код первой страницы показан в листинге 24.2.

Листинг 24.2 login.php — титульная страница системы PHPBookmark_

```
<?
 require_once("bookmark_fns.php");
 do_html_header("");

 display_site_info();
 display_login_form();

 do_html_footer();
?>
```

Этот код выглядит очень простым, поскольку в нем, в основном, вызываются функции из API-интерфейса, который мы построим для данного приложения. Подробное описание функций следует ниже. Несложно заметить, что выполняется подключение файла (содержащего функции), а затем вызываются функции визуализации HTML-заголовка, отображения содержимого и отображения нижнего колонтитула.

Вывод сценария показан на рис. 24.3.

Функции системы включены в файл **bookmark fns.php,** показанный в листинге 24.3.

Листинг 24.3 bookmark_fns.php — подключаемый файл с функциями для приложения PHP Bookmark

```
<?
   // Этот файл можно подключить ко всем остальным файлам
   // Таким образом, каждый файл будет содержать все функции
   require_once("data_valid_fns.php");
   require_once("db_fns.php");
   require_once("user_auth_fns.php");
   require_once("output_fns.php");
   require_once("url_fns.php");
?>
```

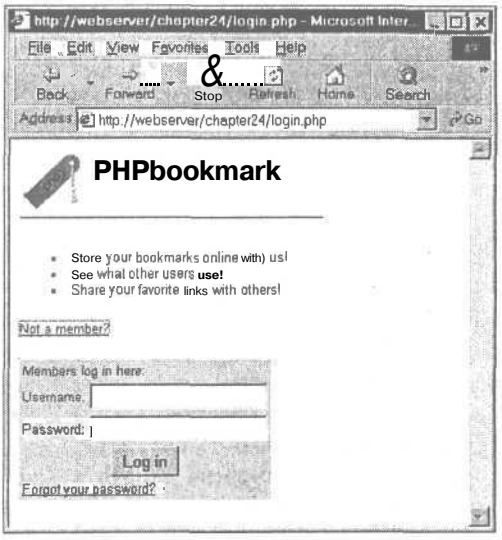


РИСУНОК 24.3
Титульная страница системы PHPBookmark генерируется функциями визуализации HTML из файла login.php.

Можно видеть, что этот файл служит лишь контейнером для пяти других подключаемых файлов, которые будут использоваться в этом приложении. Данная структура объясняется тем, что функции разбиваются на логические группы. Некоторые из групп могут применяться для других проектов, поэтому каждая группа помещается в отдельный файл. Файл **bookmark_fns.php** создан потому, что большая часть из пяти файлов функций будут использоваться в большинстве сценариев. Проще подключать один файл к каждому сценарию, чем использовать пять операторов **include.**

Обратите внимание, что конструкция **require_once()** существует только в PHP, начиная с версии 4.0.1pl2. Если используется более ранняя версия, необходимо применять функцию **require()** либо **include()** и обеспечить, чтобы файлы не загружались по нескольку раз.

В этом отдельном случае используются функции файла **output_fns.php**. Они реализуют вывод простого HTML-содержимого. Данный файл включает четыре функции, которые были использованы в файле **login.php — do_html_header(), display_site_info(), display_login_form() и do_html_footer()**, а также ряд других функций.

Мы не будем подробно изучать все функции, но одну из них рассмотрим в качестве примера. Код функции **do_html_header()** показан в листинге 24.4.

Листинг 24.4 Функция do_html_header() из файла output_fns.php - эта функция выводит стандартный заголовок, который отображается на каждой странице приложения

```
function do_html_header($title)
{
   // печать HTML-заголовка
?>
   <html>
   <head>
      <titleX?=$title?X/title>
      <style>
         body { font-family: Arial, Helvetica, sans-serif; font-size: 13px }
         li, td { font-family: Arial, Helvetica, sans-serif; font-size: 13px}
         hr { color: #3333cc; width=300; text-align=left}
         a { color: #000000 }
```

```
    </style>
  </head>
  <body>
  <img src="bookmark.gif" alt="PHPbookmark logo" border=0
       align=left valign=bottom height = 55 width = 57>
  <h1> PHPbookmark</h1>
  <hr>
<?
  if ($title)
    do_html_heading($title);
}
```

Можно видеть, что логика функции сводится к добавлению заголовка и логотипа к странице. Остальные функции, использованные в файле **login.php**, подобны данной. Функция **display_site_info()** добавляет текстовое описание сайта; **display_login_form()** отображает форму входа в систему, показанную на рис. 24.3; **do_html_footer()** включает в страницу стандартный нижний колонтитул HTML.

(Преимущества изоляции либо удаления HTML-кода из главного потока логики обсуждались в главе 22. Здесь используется подход, основанный на функциях API. Для сравнения в следующей главе рассматривается подход, основанный на шаблонах.)

На рис. 24.3 показано, что страница предоставляет три опции — пользователь может зарегистрироваться, войти в систему, если он уже зарегистрирован, либо переустановить забытый пароль. Для реализации этих модулей перейдем к следующему разделу — аутентификации пользователей.

Аутентификация пользователей

Модуль аутентификации пользователей содержит четыре главных элемента: регистрацию пользователей, вход и выход из системы, смену паролей и переустановку паролей. Рассмотрим все элементы поочередно.

Регистрация

Чтобы зарегистрировать пользователя, необходимо через форму получить сведения о нем и поместить их в базу данных.

Когда пользователь выполняет щелчок на ссылке на страницу **login.php** ("Not a member?"), для него выводится форма регистрации, сгенерированная сценарием **register_form.php**. Этот сценарий показан в листинге 24.5.

Листинг 24.5 **register_form.php** — эта форма дает пользователям возможность зарегистрироваться в системе **PHPBookmark**

```
<?
 require_once("bookmark_fns.php");
 do_html_header("User Registration");

 display_registration_form();

 do_html_footer();
?>
```

Эта страница также довольно проста и осуществляет лишь вызов функций из библиотеки вывода — **output fns.php**. Вывод сценария показан на рис. 24.4.

Форма серого цвета на этой странице представляет собой вывод функции **display_registration_form()**, которая содержится в файле **output_fns.php**. Когда пользователь выполняет щелчок на кнопке Register, выполняется сценарий **register_new.php**, показанный в листинге 24.6.

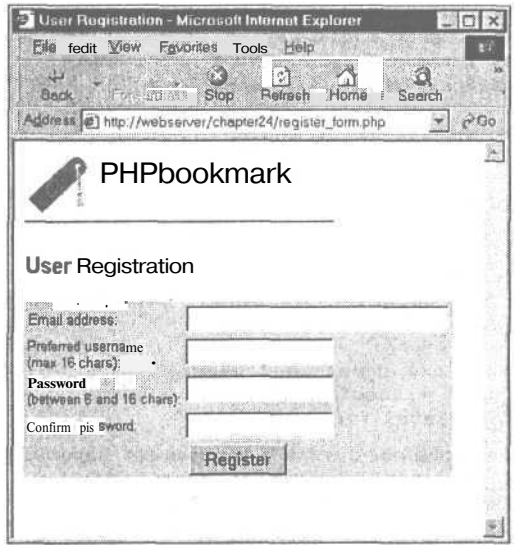


РИСУНОК 24.4
Регистрационная форма извлекает сведения, необходимые для занесения в базу данных. Во избежание ошибок пользователям предлагается ввести пароль дважды.

Листинг 24.6 **register_new.php** — этот сценарий проверяет допустимость вводимой пользователем информации и помещает ее в базу данных

```
<?
    // включаемые файлы функций для данного приложения
    require_once("bookmark_fns.php");

    // запуск сеанса, который может потребоваться поаже
    // следует запустить его сейчас, поскольку он должен
    // следовать перед заголовками
    session_start();

    // проверка заполненных форм
    if (!filled_out($HTTP_POST_VARS))
    {
        do_html_header("Problem:");
        echo "You have not filled the form out correctly - please go back"
             ." and try again.";
        do_html_footer();
        exit;
    }

    // недопустимый адрес электронной почты
    if (!valid_email($email))
    {
        do_html_header("Problem:");
        echo "That is not a valid email address.  Please go back "
             ." and try again.";
        do_html_footer();
        exit;
    }

    // пароли не совпадают
    if ($passwd != $passwd2)
    {
        do_html_heading("Problem:");
        echo "The passwords you entered do not match - please go back"
             ." and try again.";
        do_html_footer();
        exit;
    }
```

```
    // проверка длины пароля
    // усечение имени пользователя не вызывает проблем, но
    // слишком длинные пароли будут искажаться.
    if (strlen($passwd)<6 || strlen($passwd) >16)
    {
        do_html_header("Problem:");
        echo "Your password must be between 6 and 16 characters."
             ."Please go back and try again.";
        do_html_footer();
        exit;
    }
    // попытка регистрации
    $reg_result = register($username, $email, $passwd);
    if ($reg_result == "true")
    {
       // регистрация переменной сеанса
       $valid_user = $username;
       session_register("valid_user");

       // предоставление ссылки на страницу зарегистрированных пользователей
       do_html_header("Registration successful");
       echo "Your registration" was successful. Go to the members page "
            ."to start setting up your bookmarks!";
       do_HTML_URL("member.php", "Go to members page");
    }
    else
    {
       // в случае безуспешной регистрации предоставляется ссылка
       // назад с предложением повторить попытку
       do_html_header("Problem:");
       echo $reg_result;
       do_html_footer();
       exit;
    }
    // конец страницы
    do_html_footer();
?>
```

Это первый более-менее сложный сценарий приложения.

Он начинается с подключения файлов функций и запуска сеанса. (После регистрации пользователя создается переменная сеанса, содержащая имя пользователя, как это было в главе 20.)

Затем выполняется проверка допустимости данных, введенных пользователем. Потребуется предпринять ряд проверок:

- Проверить, что форма заполнена. Для этого применяется вызов функции **fiUed_out():**

  ```
  if (!filled_out($HTTP_POST_VARS))
  ```

  Эту функцию мы написали самостоятельно. Она содержится в библиотеке **data_valid_fns.php** и будет рассматриваться ниже.

- Проверить допустимость предоставленного адреса электронной почты:

  ```
  if (valid_email($email))
  ```

  Эта функция также написана самостоятельно и содержится в библиотеке **data_valid_fns.php.**

- Проверить идентичность обоих вариантов пароля, введенных пользователем:

  ```
  if ($passwd != $passwd2)
  ```

- Проверить допустимость длины пароля:

  ```
  if (strlen($passwd) < 6 || strlen($passwd) > 16)
  ```

  В нашем примере длина пароля должна составлять не менее шести символов, чтобы его было сложнее угадать, но менее 16 символов, чтобы он помещался в базе данных.

Использованные здесь функции проверки допустимости данных **filled_out()** и **valid_email()** показаны, соответственно, в листингах 24.7 и 24.8.

Листинг 24.7 Функция filled_out() библиотеки data_valid_fns.php — эта функция проверяет, заполнена ли форма

```
function filled_out($form_vars)
{
  // проверить, что каждая переменная имеет значение
  foreach ($form_vars as $key => $value)
  {
      if (!isset($key) || ($value == ""))
          return false;
  }
  return true;
}
```

Листинг 24.8 Функция valid_email() библиотеки data_valid_fns.php — эта функция проверяет допустимость адреса электронной почты

```
function valid_email ($address)
{
  // проверка допустимости адреса электронной почты
  if (ereg("^[a-zA-Z0-9_]+@[a-zA-Z0-9\-]+\.[a-zA-Z0-9\-\.]+$", $address))
    return true;
  else
    return false;
}
```

Функция **filled_out()** ожидает передачи массива переменных — **$HTTP_POST_VARS** или **$HTTP_GET_VARS**. Если массив заполнен, она возвращает значение **true,** в противном случае — **false.**

В функции **valid_email()** применяется регулярное выражение, разработанное в главе 4 для проверки адресов электронной почты. Если адрес допустим, возвращается значение **true,** в противном случае — **false.**

После проверки введенных данных можно предпринять попытку регистрации пользователя. Как видно из листинга 24.6, это выполняется следующим образом:

```
$reg_result = register($username, $email, $passwd);
if ($reg_result == "true")
{
  // регистрация переменной сеанса
  $valid_user = $username;
  session_register("valid_user");
```

```
    // предоставление ссылки на страницу зарегистрированных пользователей
    do_html_header("Registration successful");
    echo "Your registration was successful. Go to the members page "
         ."to start setting up your bookmarks!";
    do_HTML_URL("member.php", "Go to members page");
}
```

В вызываемую функцию **register()** передаются имя пользователя, адрес электронной почты и пароль, введенные в форме регистрации. В случае успешного исхода имя пользователя регистрируется в качестве переменной сеанса и предоставляется ссылка на главную страницу зарегистрированных пользователей. Вывод сценария показан на рис. 24.5.



РИСУНОК 24.5
Регистрация прошла успешно — пользователь может перейти на страницу зарегистрированных членов.

Функция **register()** включена в библиотеку **user_auth_fns.php** и показана в листинге 24.9.

Листинг 24.9 Функция register() библиотеки user_auth_fns.php — предпринимает попытку ввода информации о новом пользователе в базу данных

```
function register($username, $email, $password)
// регистрация нового пользователя в базе данных
// возвращает значение true либо сообщение об ошибке
{
  // подключение к базе данных
  $conn = db_connect();
  if (!$conn)
     return "Could not connect to database server - please try later.";

  // проверить имя пользователя на уникальность
  $result = mysql_query("select * from user where username='$username'");
  if (!$result)
      return "Could not execute query";
  if (mysql_num_rows($result)>0)
      return "That username is taken - go back and choose another one.";

  // если имя уникально, данные заносятся в базу данных
  $result = mysql_query("insert into user values
                        ('$username', password('$password'), '$email')");
```

```
    if (!$result)
     return "Could not register you  in database - please try again later.";
    return true;
}
```

Эта функция не содержит ничего нового — она осуществляет подключение к созданной ранее базе данных. Если выбранное имя пользователя уже задействовано либо база данных не может быть обновлена, функция возвращает значение false. В противном случае база данных обновляется и возвращается значение true.

Заметим, что подключение к базе данных реализуется через написанную ранее функцию **db_connect()**. Она просто предоставляет единственную область хранения имени пользователя и пароля для подключения к базе данных. Таким образом, для изменения пароля базы данных достаточно модернизировать один файл приложения. Функция показана в листинге 24.10.

Листинг 24.10 Функция db_connect() библиотеки db_fns.php - выполняет подключение к базе данных MySQL

```
function db_connect()
{
    $result = mysql_pconnect("localhost", "bm_user", "password");
    if (!$result)
        return false;
    if (!mysql_select_db("bookmarks"))
        return false;

    return $result;
}
```

Зарегистрированные пользователи могут входить и выходить из системы через обычные страницы, предназначенные для этих целей. Мы построим их в следующих разделах.

ВХОД *В СИСТЕМУ*

После того как пользователи внесут свои данные в форму (**login.php**, см. рис. 24.3) и отправят ее, выполнится сценарий **member.php**. Этот сценарий позволит им войти в систему. Кроме того, отобразятся связанные с пользователями закладки. Это центральное событие оставшейся части приложения. Данный сценарий показан в листинге 24.11.

Листинг **24.11** member.php — этот сценарий является основой приложения

```
<?

// подключаемые файлы функций для данного приложения
require_once("bookmark_fns.php");
session_start();

if (!$username && !$passwd)
// пользователь только что попытался войти в систему
<
    if (login($username, $passwd))
    <
        // если пользователь записан в базе данных,
        // зарегистрировать его идентификатор
        $valid_user = $username;
        session_register("valid_user");
    }
```

```
        else
        {
           // неуспешная попытка входа а систему
           do_html_header("Problem:");
           echo "You could not be logged in.
                 You must be logged in to view this page.";
           do_html_url("login.php","Login");
           do_html_footer();
           exit;
        }
}
do_html_header("Home");
check_valid_user();
// отображение закладок, сохраненных пользователем
if (Surl_array = get_user_urls($valid_user));
   display_user_urls($url_array);

// отображение меню
display_user_menu();

do_html_footer();

?>
```

Логика этого сценария узнаваема — в нем используются идеи из главы 20.

Во-первых, выполняется проверка, осуществил ли пользователь переход из титульной страницы. Другими словами, заполнил ли он форму входа в систему. Затем выполняется попытка зарегистрировать пользователя в системе:

```
if ($username && $passwd)
// пользователь только что попытался войти в систему
{
     if (login($username, $passwd))
     {
        // если пользователь записан в базе данных,
        // зарегистрировать его идентификатор
        $valid_user = $username;
        session_register("valid_user");
     }
```

Для входа в систему используется функция **login().** Она содержится в библиотеке **user_auth_fns.php** и рассматривается ниже.

Если попытка входа в систему окажется успешной, сеанс будет зарегистрирован как и ранее. При этом имя пользователя сохраняется в переменной сеанса **$valid_user**.

Если все идет нормально, отображается страница зарегистрированных пользователей:

```
do_html_header("Home");
check_valid_user();
// отображение закладок, сохраненных пользователем
if ($url_array = get_user_urls($valid_user));
   display_user_urls($url_array);

// отображение меню
display_user_menu();

do_html_footer();
```

Эта страница также формируется функциями вывода. Можно заметить, что в ней используется несколько новых функций — **check_valid_user()** из файла **user_auth_fns.php**, **get_user_urls()** из файла **url_fns.php и display_user_urls()** из файла **output_fns.php** Фун-

кция **check_valid_user()** проверяет, что с текущим пользователем связан зарегистрированный сеанс. Она предназначена для пользователей, которые открыли сеанс ранее, а не только что вошли в систему. Функция **get_user_urls()** извлекает закладки пользователя из базы данных, а **display_user_urls()** отображает закладки в браузере. Код функции **check_valid_user()** будет рассмотрен вскоре, а остальных двух функций — во время обзора процесса хранения и извлечения закладок.

Сценарий **member.php** завершает страницу отображением меню с использованием функции **display_user_menu()**.

Пример вывода сценария **member.php** показан на рис. 24.6.

Теперь более подробно рассмотрим функции **login()** и **check_valid_user()**. Функция **login()** показана в листинге 24.12.

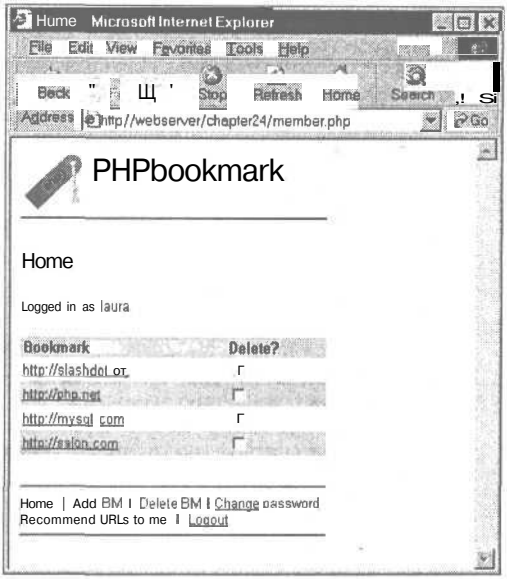


РИСУНОК 24.6 *Сценарий member.php проверяет, что пользователь вошел в систему, извлекает и отображает его закладки, а затем выводит меню.*

Листинг 24.12 Функция login() библиотеки user_auth_fns.php — проверяет сведения о пользователе в базе данных

```
function login($username, $password)
// проверка наличия имени пользователя и пароля в базе данных
// если они там содержатся, возвращается значение true,
// в противном случае – false.
{
   // подключение к базе данных
   $conn = db_connect();
   if (!$conn)
      return 0;

   // проверка уникальности имени пользователя
   $result = mysql_query("select * from user
                          where username='$username'
                          and passwd = password('$password')");
   if (!$result)
       return 0;

   if (mysql_num_rows($result)>0)
       return 1;
   else
       return 0;
}
```

Функция подключается к базе данных и проверяет в ней наличие комбинации имени и пароля для данного пользователя. Если эти записи присутствуют, возвращается значение **true,** в противном случае, либо когда данные пользователя не могут быть проверены, — **false**.

Функция **check_valid_user()** не выполняет повторного соединения с базой данных, однако проверяет, что с пользователем связан зарегистрированный сеанс. Другими словами, он вошел в систему ранее. Эта функция показана в листинге 24.13.

Листинг 24.13 Функция check_valid_user() библиотеки user_auth_fns.php — проверяет, что с пользователем связан сеанс

```
function check_valid_user()
// определить, вошел ли пользователь в систему и,
// если нет, вывести уведомление
{
   global $valid_user;
   if (session_is_registered("valid_user"))
   {
        echo "Logged in as $valid_user.";
        echo "<br>";
   }
   else
   {
       // пользователь не вошел в систему
       do_html_heading("Problem:");
       echo "You are not logged in.<br>";
       do_html_url("login.php", "Login");
       do_html_footer();
       exit;
   }
}
```

Если пользователь не вошел в систему, функция укажет ему, что это необходимо выполнить, чтобы данная страница отобразилась, и предоставит ему ссылку на страницу входа.

Выход из системы

Как видно из рис. 24.6, меню содержит ссылку "Logout". Щелчок на этой ссылке приводит к вызову сценария **logout.php.** Код сценария показан в листинге 24.14.

Листинг 24.14 logout.php — этот сценарий завершает сеанс пользователя

```
<?
// подключаемые файлы функций
require_once("bookmark_fns.php");
session_start();
$old_user = $valid_user; //сохранить на случай, если кто-то входил в систему
$result_unreg = session_unregister("valid_user");
$result_dest = session_destroy();

// запуск выводимого HTML-кода
do_html_header("Logging Out");

if(!empty($old_user))
{
   if ($result_unreg && $result_dest)
   {
     // пользователь вошел в систему и теперь выходит из нее
     echo "Logged out.<br>";
     do_html_url("login.php", "Login");
   }
   else
   {
    // пользователь вошел в систему и не может выйти из нее
     echo "Could not log you out.<br>";
   }
}
```

```
else
{
   // пользователь не вошел в систему,
   // но каким-то образом умудрился
   // открыть эту страницу
   echo "You were not logged in, and
    so have not been logged
    out.<br>";
   do_html_url("login.php", "Login");
}
do_html_footer();
?>
```



РИСУНОК 24.7 *Сценарии change_passwd_form.php предоставляет форму, в которой пользователи могут менять пароли.*

Этот код также может показаться знакомым. В нем использованы идеи, рассмотренные в главе 20.

Смена *паролей*

Если пользователь выберет опцию меню "Change Password", отобразится форма, показанная на рис. 24.7.

Эта форма сгенерирована сценарием **change_passwd_form.php**. Он довольно прост и использует лишь функции библиотеки вывода. Поэтому его код здесь не рассматривается.

После отправки формы запускается сценарий **change_passwd.php**, показанный в листинге 24.15.

Листинг 24.15 change_passwd.php — **этот сценарий выполняет попытку смены пароля**

```
<?
 require_once("bookmark_fns.php");
 session_start();
 do_html_header("Changing password");
 check_valid_user();
 if (!filled_out($HTTP_POST_VARS))
 {
   echo "You have not filled out the form completely.
         Please try again.";
   display_user_menu();
   do_html_footer();
   exit;
 }
 else
 {
    if ($new_passwd!=$new_passwd2)
      echo "Passwords entered were not the same. Not changed.";
    else if (strlen($new_passwd)>16 || strlen($new_passwd)<6)
      echo "New password must be between 6 and 16 characters.Try again.";
    else
    {
        // попытка обновления
        if (change_password($valid_user, $old_passwd, $new_passwd))
            echo "Password changed.";
        else
            echo "Password could not be changed.";
    }
 }
```

```
    display_user_menu();
    do_html_footer();
?>
```

Этот сценарий проверяет, что пользователь вошел в систему (функция **check_valid_user()**), заполнил форму ввода пароля (**filled_out()**), в обоих полях введены одинаковые пароли и длина их является допустимой. Если все правильно, вызывается функция **change_password()**:

```
if (change_password($valid_user, $old_passwd, $new_passwd) )
   echo "Password changed.";
else
   echo "Password could not be changed.";
```

Эта функция содержится в библиотеке **user_auth_fns.php** и показана в листинге 24.16.

Листинг 24.16 Функция change_password() библиотеки user_auth_fns.php — выполняет попытку обновления пароля в базе данных

```
function change_password($username, $old_password, $new_password)
// замена старого пароля новым
// возвращает значение true или false
{
   // если прежний пароль введен верно,
   // он заменяется новым и возвращается значение true,
   // в противном случае — false
   if (login($username, $old_password) )
   {
      if ( ! ($conn = db_connect() ) )
         return false;
      $result = mysql_query( "update user
                              set passwd = password ('$new_password')
                              where username= ' $username' " ) ;
      if (!$result)
         return false;   // пароль не изменен
      else
         return true;   // успешное изменение
   }
   else
      return false; // прежний пароль введен неверно
}
```

Эта функция проверяет правильность ввода прежнего пароля с помощью уже рассмотренной функции **login()**. Если пароль указан верно, функция соединяется с базой данных и обновляет пароль новым значением.

Переустановка забытых паролей

Помимо смены пароля необходимо предусмотреть часто возникающую ситуацию, когда пользователь попросту забывает пароль. Обратите внимание, что титульная страница, **login.php,** содержит ссылку "Forgot your password?", предназначенную для подобных случаев. Ссылка запускает сценарий **forgot_form.php**, который использует функции вывода для отображения формы (см. рис. 24.8).

Этот сценарий очень прост. В нем используются лишь функции вывода, поэтому код здесь не рассматривается. После отправки формы вызывается сценарий **forgot_passwd.php**, который заслуживает специального рассмотрения. Код сценария показан в листинге 24.17.



РИСУНОК 24.8
Сценарий forgot_form.php выводит форму, в которой пользователь может запросить переустановку пароля и отправку его по электронной почте.

Листинг 24.17 forgot_passwd.php — этот сценарий переустанавливает пароль, выбирая для него случайное значение, и отправляет новую версию пользователю по электронной почте

```
<?
  require_once("bookmark_fns.php");
  do_html_header("Resetting password");

  if ($password=reset_password($username))
  {
      if (notify_password($username, $password))
        echo "Your new password has been sent to your email address.";
      else
        echo "Your password could not be mailed to you."
             "." Try pressing refresh.";
  }
  else
     echo "Your password could not be reset - please try again later.";

   do_html_url("login.php", "Login");

  do_html_footer();
?>
```

В сценарии задействованы две основных функции: **reset_password()** и **notify_password()**. Рассмотрим их по очереди.

Функция **reset_password()** генерирует случайный пароль и помещает его в базу данных. Ее код приводится в листинге 24.18.

Листинг 24.18 Функция reset_password() библиотеки user_auth_fns.php — этот сценарий присваивает паролю случайное значение и отправляет его пользователю по электронной почте

```
function reset_password($username)
// установка случайного значения для пароля
// возвращается новый пароль либо значение false в случае ошибки
{
   // получение случайного слова длиной от 6 до 13 символов
   $new_password = get_random_word(6, 13);
```

```
    // добавление числа от 0 до 999
    // для небольшого усложнения пароля
    srand ((double) microtime() * 1000000);
    $rand_number = rand(0, 999);
    $new_password .= $rand_number;

    // изменение пароля в базе данных или возврат значения false
    if (!($conn = db_connect()))
        return false;
    $result = mysql_query( "update user
                            set passwd = password('$new_password')
                            where username = '$username'");
    if (!$result)
      return false;   // пароль не изменен
    else
      return $new_password;   // успешное изменение
}
```

Эта функция генерирует случайный пароль, получив случайное слово из словаря с помощью функции **get_random_word()** и добавив к нему случайное число от 0 до 999. Функция **get_random_word()** также содержится в библиотеке user_auth_fns.php. Она показана в листинге 24.19.

Листинг 24.19 Функция get_random_word() библиотеки user_auth_fns.php — получает случайное слово из словаря, используемое для генерации пароля

```
function get_random_word($min_length, $max_length)
// извлечение случайного слова из словаря в заданном
// диапазоне длины и возврат слова
{
   // генерация случайного слова
   $word = "";
   $dictionary = "/usr/dict/words";   // словарь ispell
   $fp = fopen($dictionary, "r");
   $size = filesize($dictionary);

   // переход к случайной позиции в словаре
   srand ((double) microtime() * 1000000);
   $rand_location = rand(0, $size);
   fseek($fp, $rand_location);

   // получение из файла следующего целого слова допустимой длины
   while (strlen($word)< $min_length || strlen($word)>$max_length)
   {
       if (feof($fp))
           fseek($fp, 0); // если достигнут конец файла, перейти на начало
       $word = fgets($fp, 80);   // пропустить первое слово, поскольку оно
                                 // может быть неполным
       $word = fgets($fp, 80); // потенциальный пароль
   };
   $word=trim($word); //усечение завершающих символов \n
                      //из результата функции fgets
   return $word;
}
```

Функция работает при наличии словаря. В системе UNIX встроенная программа проверки орфографии **ispell** укомплектована словарем, который обычно содержится в каталоге **/usr/dict/words**, как в нашем примере. Если используется другая операционная система либо нет желания устанавливать словарь, можно загрузить список слов, используемый программой **ispell** из следующего ресурса:

```
http://ficus-www.cs.ucla.edu/ficus-members/geoff/ispell-dictionaries.html
```

Этот сайт содержит словари на многих языках (в т.ч. и на английском). Поэтому для получения случайного слова, скажем, на норвежском языке или эсперанто, можно загрузить один из соответствующих словарей. В файлах словарей каждое слово содержится в отдельной строке, а разделителями служат символы перевода строки.

Чтобы получить случайное слово из файла, выбирается случайная позиция в диапазоне от 0 до значения размера файла, из которого выполняется чтение. В таком случае, вероятнее всего, будет прочитана часть слова, поэтому текущая строка пропускается и выбирается следующее слово путем двухкратного обращения к функции **fgets().**

Эта функция имеет две интересных особенности. Во-первых, если в процессе поиска слова достигается конец файла, выполняется переход на его начало:

```
if (feof($fp))
        fseek($fp, 0); // если достигнут конец файла, перейти на начало
```

Во-вторых, выполняется поиск слова определенной длины. Проверяется длина каждого слова, извлекаемого из словаря. Если она находится в диапазоне значений от **$min_length** до **$max_length**, поиск продолжается.

Вернемся к функции **reset_password()**. После того как будет сгенерирован новый пароль, произойдет обновление базы данных и возврат нового пароля в главный сценарий. Затем он передается в функцию **notify_password()**, которая отправит его пользователю по электронной почте.

Рассмотрим функцию ***notify_password()***, показанную в листинге 24.20.

Листинг 24.20 Функция notify_password() из библиотеки user_auth_fns.php — отправляет пользователю новый пароль по электронной почте

```
function notify_password($username, $password)
// уведомление пользователя о том, что его пароль был изменен
{
    if (!($conn = db_connect()))
      return false;
    $result = mysql_query("select email from user
                           where username='$username'");
    if (!$result)
      return false;   // пароль не изменен
    else if (mysql_num_rows($result)==0)
      return false; // имя пользователя в базе данных отсутствует
    else
    {
      $email = mysql_result($result, 0, "email");
      $from = "From: support@phpbookmark \r\n";
      $mesg = "Your PHPBookmark password has been changed to $password \r\n"
              ."Please change it next time you log in. \r\n";
      if (mail($email, "PHPBookmark login information", $mesg, $from))
        return true;
      else
        return false;
    }
}
```

Эта функция по имени пользователя и новому паролю выполняет поиск адреса электронной почты в базе данных и применяет PHP-функцию **mail()** для отправки пароля пользователю.

Безопаснее предоставить пользователю действительно случайный пароль, составленный из комбинации букв верхнего и нижнего регистра, чисел и знаков пунктуации,

вместо случайного слова и числа. Однако пароль вроде "zigzag487" пользователю будет проще читать и печатать, чем случайный набор символов. В таком наборе часто трудно различить 0 и O (нуль от прописного "O"), либо 1 и l (единицу от строчной "L").

В нашей системе файл словаря содержит приблизительно 45000 слов. Если даже взломщик знает способ создания пароля и имя пользователя, то ему и тогда придется угадать, в среднем, один вариант из 22500000. Этот уровень защищенности достаточен для приложений данного типа, даже если пользователи оставляют без внимания сообщения электронной почты с предложением смены пароля.

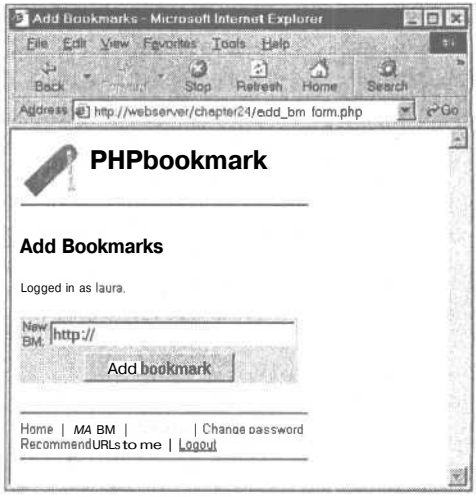


РИСУНОК 24.9 *Сценарий add_bm_form.php предоставляет форму, с помощью которой пользователи могут добавлять закладки к своим страницам.*

Хранение и извлечение закладок

Пришло время познакомиться с методами хранения, извлечения и удаления закладок.

Добавление закладок

Для добавления закладок можно щелкнуть на ссылке Add BM в меню пользователя. В результате отображается форма, показанная на рис. 24.9.

Этот сценарий также прост и использует лишь функции вывода. Поэтому его код здесь не рассматривается. После отправки формы вызывается сценарий **add_bms.php**, показанный в листинге 24.21.

Листинг 24.21 add_bms.php — этот сценарий добавляет новые закладки в личную страницу пользователя

```
<?
  require_once("bookmark_fns.php");
  session_start();
  do_html_header("Adding bookmarks");
  check_valid_user();
  if (!filled_out($HTTP_POST_VARS))
  {
    echo "You have not filled out the form completely.
          Please try again.";
    display_user_menu();
    do_html_footer();
    exit;
  }
  else
  {
    // проверка формата URL-адреса
    if (strstr($new_url, "http://")===false)
        $new_url = "http://".$new_url;

    // проверка допустимости URL-адреса
    if (@fopen($new_url, "r"))
    {
       // попытка добавления закладки
```

```
        if (add_bm($new_url))
          echo "Bookmark added.";
        else
          echo "Could not add bookmark.";
      }
      else
        echo "Not a valid URL.";
  }
    // отображение закладок, сохраненных данным пользователем
   if ($url_array = get_user_urls($valid_user));
     display_user_urls($url_array);

    display_user_menu();
    do_html_footer();
?>
```

Этот сценарий также осуществляет проверку допустимости данных, запись в базу данных и вывод информации.

Для проверки допустимости данных сначала с помощью функции **filled_out()** определяется, заполнил ли пользователь форму.

Затем выполняются две проверки URL-адреса. Вначале с помощью функции **strstr()** определяется, начинается ли адрес с последовательности http://. Если нет, она добавляется в начало адреса. После этого можно проверить, что данный адрес действительно существует. Как упоминалось в главе 17, функция **fopen()** позволяет открыть URL-адрес, начинающийся с последовательности **http://**. Если открыть файл удается, предполагается, что URL-адрес допустим, и вызывается функция **add_bm()** для его включения в базу данных.

Обратите внимание, что функция **fopen()** способна открывать файлы лишь в случае, когда сервер имеет прямой доступ к Internet. Если необходимо обращение к другим HTTP-серверам через прокси-сервер, функция **fopen()** не действует.

Эта и другие функции, связанные с закладками, содержатся в библиотеке **url_fns.php**. Код функции **add_bm()** приводится в листинге 24.22.

Листинг 24.22 Функция add_bm() из библиотеки url_fns.php — заносит в базу данных новые закладки

```
function add_bm($new_url)
{
  // добавление в баау данных новых закладок

  echo "Attempting to add " .htmlspecialchars($new_url)."<BR>";
  global $valid_user;
  if (!($conn = db_connect()))
    return false;

  // проверка, что закладка не дублируется
  $result = mysql_query("select * from bookmark
                         where username='$valid_user'
                         andbm_URL='$new_url'");
  if ($result && (mysql_num_rows($result)>0))
    return false;

  // вставка новой закладки
  if (!mysql_query( "insert into bookmark values
                       ('$valid_user','$new_url')"))
    return false;
   return true;
}
```

Эта функция довольно проста. Она проверяет, что данная закладка еще не содержится в базе данных. (Хотя маловероятно, что закладка будет вводиться дважды, вполне возможен случай, когда пользователь обновляет страницу.) Если закладка новая, она вводится в базу данных.

Вернемся к сценарию **add_bm.php.** Как и в сценарии **member.php**, его последние операции — вызов функций **get_user_urls() и display_user_urls()**. **Они** будут рассматриваться ниже.

Отображение закладок

В сценарии **member.php** и функции **add_bm()** применялись функции **get_user_urls() и display_user_urls()**. **Они** осуществляют, соответственно, извлечение закладок из базы данных и их отображение. Функция **get_user_urls()** содержится в библиотеке **url_fns.php, а display_user_urls()** — в библиотеке **output_fns.php**.

Функция **get_user_urls()** показана в листинге 24.23.

Листинг 24.23 Функция get_user_urls() из библиотеки url_fns.php — извлекает закладки пользователя из базы данных

```
function get_user_urls ($username)
{
  // извлечение из базы данных всех сохраненных пользователем URL-адресов
  if (!($conn = db_connect() ) )
    return false;
  $result = mysql_query( "select bm_URL
                          from bookmark
                          where username = '$username'");
  if  (!$result)
    return  false;

  // создание массива URL-адресов
  $url_array = array();
  for ($count = 1; $row = mysql_fetch_row ($result) ; ++$count)
  {
    $url_array[$count] = $row[0];
  }
  return $url_array;
};
```

Рассмотрим кратко эту функцию. Она принимает имя пользователя в качестве параметра и извлекает для него закладки из базы данных. Функция возвращает массив URL-адресов либо значение **false,** если закладки не могут быть извлечены.

Этот массив может передаваться из функции **get_user_urls()** в функцию **display_user_urls().** Это простая функция вывода HTML-содержимого, выполняющая печать URL-адресов в привлекательном табличном формате. Она здесь не рассматривается. Пример вывода показан на рис. 24.6. Функция помещает URL-адреса в форму. Рядом с каждым URL-адресом находится флажок, который позволяет пометить закладку для удаления.

Удаление закладок

Когда пользователь помечает некоторые закладки для удаления и выбирает из меню опцию Delete BM, передается форма, содержащая URL-адреса. Каждый флажок генерируется с помощью следующего кода функции **display_user_urls()**:

```
echo  "<td><input  type=checkbox  name=\"del_me[]\"
                  value=\"$url\"></td>";
```

Имя каждого флажка — **delme[]**. Это означает, что, если форма запускает PHP-сценарий, будет предоставлен доступ к массиву **$del_me**, который содержит все удаляемые закладки.

Опция Delete BM запускает сценарий **delete_bms.php**, показанный в листинге 24.12.

Листинг 2424 **delete_bms.php** — этот сценарий удаляет закладки из базы данных

```
<?
 require_once("bookmark_fns.php");
 session_start();
 do_html_header("Deletingbookmarks");
 check_valid_user();
 if (!filled_out($HTTP_POST_VARS))
 {
   echo "You have not chosen any bookmarks to delete.
         Please try again.";
   display_user_menu();
   do_html_footer();
   exit;
 }
 else
 {
    if (count($del_me) >0)
    {
       foreach ($del_me as $url)
       {
          if (delete_bm($valid user, $url))
            •cho "Deleted ".htmlspecialchars($url)."<br>";
          else
            echo "Could not delete ".htmlspecialchars($url)."<br>";
       }
    }
    else
       •cho "No bookmarks selected for deletion";
 }
 // отображение закладок, сохраненных данным пользователем
 if ($url_array = get_user_urls($valid_user));
   display_user_urls($url_array);
 display_user_menu();
 do_html_footer();
?>
```

Сценарий начинается с обычной проверки допустимости данных. Когда выясняется, что пользователь выбрал несколько закладок для удаления, их удаление выполняется в следующем цикле:

```
foreach($del_me as $url)
{
    if (delete_bm($valid_user, $url))
      echo "Deleted ".htmlspecialchars($url)."<br>";
    else
      echo "Could not delete ".htmlspecialchars($url)."<br>";
}
```

Можно видеть, что функция **delete_bm()** осуществляет удаление закладки из базы данных. Она приводится в листинге 24.25.

Листинг 24.25 Функция delete_bm() из библиотеки url_fns.php — удаляет одну закладку из списка пользователя

```
function delete_bm($user, $url)
{
   // удаление одного URL-адреса из базы данных
   if (!($conn = db_connect()))
      return false;

    // удаление закладки
   if (!mysql_query( "delete from bookmark
                      where username='$user' and bm_url='$url' "))
      return false;
   return true;
}
```

Эта функция также очень проста. Она предпринимает попытку удаления из базы данных закладки определенного пользователя. Отметим, что необходимо удалить определенную пару "имя пользователя — закладка". Для остальных пользователей данная закладка может сохраняться.

Пример вывода сценария удаления показан на рис. 24.10.

Как и в сценарии **add_bms.php**, после внесения изменений в базу данных отображается новый список закладок с помощью функций **get_user_urls()** и **display_user_urls()**.

Выработка рекомендаций

Наконец, переходим к сценарию рекомендации ссылок, **recommend.php**.

Существует множество различных способов выработки рекомендаций. Мы решили применить принцип "подобия интересов". Другими словами, выполняется поиск остальных пользователей, у которых хотя бы одна закладка совпадает с закладкой данного пользователя. Предполагается, что их остальные закладки также могут представлять интерес для пользователя.

Простейший метод реализации этого подхода в SQL-запросе — применение подзапроса. Сначала через подзапрос извлекается список пользователей со сходными интересами, а затем просматриваются их закладки с помощью внешнего запроса.

Однако, как уже говорилось, MySQL не поддерживает механизм подзапросов. Придется выполнить два различных запроса и передать вывод первого из них во второй запрос. Этого можно достичь путем создания временной таблицы результатов первого запроса либо их обработкой средствами PHP.

Выберем второй подход. Использование временной таблицы, вероятно, приведет к небольшому увеличению быстродействия, но обработка результатов средствами PHP упростит проверку и модификацию кода.

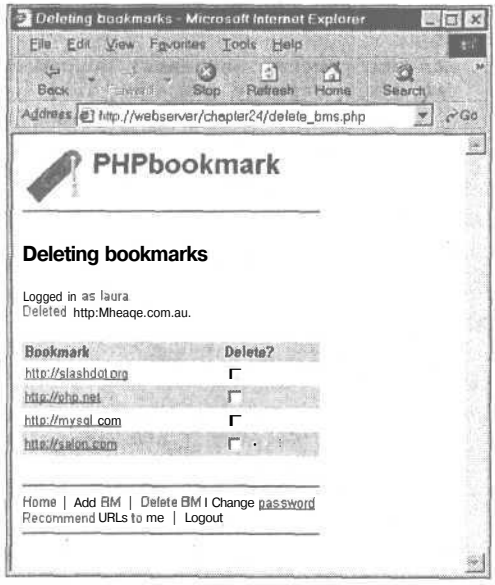


РИСУНОК 24.10 *Сценарий удаления уведомляет пользователя о том, какие закладки удалены, и отображает оставшиеся.*

Начнем с выполнения следующего запроса:

```
select distinct(b2.username)
from bookmark b1, bookmark b2
where b1.username='$valid_user'
and b1.username != b2.username
and b1.bm_URL = b2.bm_URL
```

В этом запросе используются псевдонимы для соединения таблицы закладок базы данных с собой же — странная, но иногда полезная концепция. Предположим, что действительно существуют две таблицы закладок — b1 и b2. В таблице b1 выбираются данные текущего пользователя и его закладок. В другой таблице просматриваются закладки всех остальных пользователей. Выполняется поиск других пользователей (**b2.username**), имеющих закладку (URL-адрес), совпадающий с закладкой текущего пользователя (**b1.bm_URL = b2.bm_URL**). Их имена не должны совпадать с именем текущего пользователя **(bl.username != b2.username**).

Этот запрос предоставляет список пользователей, интересы которых совпадают с интересами текущего пользователя. Воспользовавшись этим списком, можно выполнять поиск остальных закладок пользователей из списка с помощью следующего запроса:

```
select bm_URL
from bookmark
where username in $sim_users
and bm_URL not in $user_urls
group by bm_URL
having count(bm_URL)>$popularity
```

Переменная **$sim_users** содержит список пользователей со сходными интересами. Список закладок текущего пользователя содержится в переменной **$user_urls** — если таблица b1 уже содержит некоторую закладку, не имеет смысла ее рекомендовать. Наконец, переменная **$popularity** добавляет некоторый элемент фильтрации — не следует рекомендовать "слишком личные" закладки. Выбираются лишь те URL-адреса, которые сохранены определенным числом других пользователей.

Если ожидается регистрация в системе большого количества посетителей, можно увеличить значение переменной **$popularity**, чтобы рекомендовать лишь URL-адреса, сохраненные большим числом пользователей. Эти адреса будут наиболее интересными и отвечающими более широкому спектру интересов по сравнению с обычными Web-страницами.

Полный вариант сценария выработки рекомендаций показан в листингах 24.26 и 24.27. Основной сценарий называется **recommend.php** (см. листинг 24.26). Он вызывает функцию **recommend_urls()** из файла **url_fns.php** (см. листинг 24.27).

Листинг 24.26 recommend.php — предлагает пользователю ссылки, которые могут заинтересовать пользователя

```
<?
 require_once("bookmark_fns.php");
 session_start();
 do_html_header("Recommending URLs");
 check_valid_user();
 $urls = recommend_urls($valid_user);
 display_recommended_urls($urls);

 display_user_menu();
 do_html_footer();
?>
```

Листинг 24.27 Функция recommend_urls() из библиотеки **url_fns.php** — этот сценарий вырабатывает рекомендации

```
function recommend_urls ($valid_user, $popularity = 1)
{
  // Пользователям предоставляются практически разумные
  // рекомендации. Если у них есть URL-адрес,
  // совпадающий с закладками других пользователей,
  // остальные адреса этих пользователей также могут
  // представлять интерес
  if (!($conn = db_connect()))
    return false;

  // поиск других пользователей, закладки которых
  // совпадают с закладкой текущего пользователя

  if (!($result = mysql_query("
                        select distinct(b2.username)
                        from bookmark b1, bookmark b2
                        where b1.username='$valid_user'
                        and b1.username != b2.username
                        and b1.bm_URL = b2.bm_URL
                       ")))
    return false;
  if (mysql_num_rows($result)==0)
    return false;

  // создание набора пользователей с общими
  // URL-адресами для применения в операторе IN
  $row = mysql_fetch_object($result);
  $sim_users = "('".($row->username)."'";
  while ($row = mysql_fetch_object($result))
  {
      $sim_users .= ", '".($row->username)."'";
  }
  $sim_users .= ")";

  // создание списка URL-адресов пользователя во
  // избежание повторения тех, что ему уже известны
  if (!($result = mysql_query("
                        select bm_URL
                        from bookmark
                        where username='$valid_user'")))
    return false;

  // создание набора URL-адресов пользователя
  // для применения в операторе IN
  $row = mysql_fetch_object($result);
  $user_urls = " ('".($row->bm_URL)."'";
  while ($row = mysql_fetch_object($result))
  {
      $user_urls .= " , '".($row->bm_URL)."'";
  }
  $user_urls .= ")";

  // в качестве простого способа исключения личных
  // страниц и повышения вероятности рекомендовать
  // представляющие интерес URL-адреса с использованием
  // минимального уровня популярности
  // если $popularity = 1, могут рекомендоваться лишь
  // адреса, сохраненные более чем одним пользователем
```

```
    // поиск максимально возможного количества URL-адресов
    if (!($result = mysql_query("
                          select bm_URL
                          from bookmark
                          where username in $sim_users
                          and bm_URL not in $user_urls
                          group by bm_URL
                          having count(bm_URL)>$popularity
                        ")))
        return false;
    if (!($num_urls=mysql_num_rows($result)))
       return false;

    $urls = array() ;
    // построение массива подходящих URL-адресов
    for ($count=0; $row = mysql_fetch_object($result); $count++)
    {
        $urls[$count] = $row->bm_URL;
    }
    return $urls;
}
```

Пример вывода сценария **recommend.php** показан на рис. 24.11.

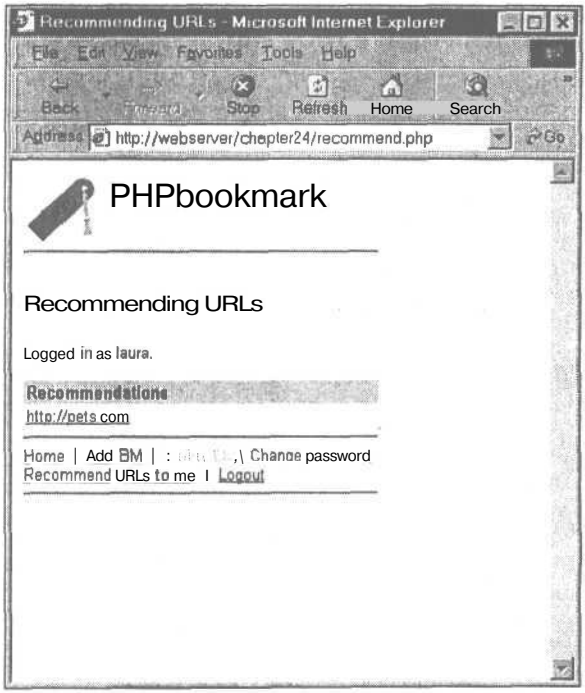


РИСУНОК 24.11
Сценарий, рекомендующий пользователю сайт pets.com. Этот адрес сохранен двумя другими пользователями из сайта slashdot.org.

Заключение и возможные расширения

Созданы базовые функциональные возможности приложения PHPBookmark. Ниже приводится список возможных расширений программы:

- Группирование закладок по темам
- Ссылка на рекомендации "Add this to my bookmarks" (добавить к сохраненным закладкам)
- Рекомендации, основанные на наиболее популярных URL-адресах базы данных либо на определенной теме
- Интерфейс администрирования пользователей и тем
- Методы повышения "интеллектуальности" закладок либо их быстродействия
- Автоматический выход пользователей из системы по истечении определенного времени
- Дополнительная проверка ошибок ввода пользователей

Экспериментируйте! Это наилучший метод изучения!

Что дальше

Следующий проект посвящен созданию покупательской тележки, которая даст пользователям возможность просматривать сайт, добавлять покупки с последующим подсчетом суммы и осуществлением электронного платежа.

Глава 25

Создание покупательской тележки

В этой главе изучаются основные методы создания покупательской тележки (shopping cart). Программа будет Строиться на основе базы данных Book-O-Rama, реализованной в части 2. Будет рассмотрена и другая возможность — установка и применение существующей покупательской тележки с открытым исходным кодом на PHP.

Термин *покупательская тележка* (иногда используется другое название, *покупательская корзинка, shopping basket)* описывает специальный интерактивный механизм осуществления покупок. В процессе просмотра интерактивного каталога можно добавлять в свою тележку отдельные позиции (наименования товаров). После завершения просмотра пользователь рассчитывается с интерактивным магазином — другими словами, приобретает товар, находящийся в тележке.

Для построения тележки будут реализованы следующие функциональные возможности:

- База данных продукции, которая будет продаваться в интерактивном магазине
- Интерактивный каталог товара с разбивкой по категориям
- Покупательская тележка, позволяющая отслеживать товар, выбираемый пользователем с целью приобретения
- Сценарий расчетов, который обрабатывает элементы платежа и доставки товара
- Интерфейс администрирования

Задача

Вероятно, вы помните базу данных Book-O-Rama, которая разрабатывалась во второй части. В этом проекте мы найдем ей применение. Система должна отвечать следующим требованиям:

- Необходимо отыскать способ подключения базы данных к браузеру пользователя. Пользователи должны иметь возможность просматривать позиции каталога, разбитые по категориям.
- Пользователи должны иметь возможность выбирать позиции из каталога с целью дальнейшего приобретения. Выбираемые позиции необходимо отслеживать.
- После завершения покупок выполняется подсчет суммы заказа, прием сведений для доставки и обработка платежа.
- Необходимо создать интерфейс администрирования сайта Book-O-Rama. Администратор должен иметь возможность добавления и редактирования информации о книгах и категориях сайта.

Компоненты решения

Рассмотрим методы реализации каждого из перечисленных требований.

Построение интерактивного каталога

Для каталога Book-O-Rama база данных уже существует. Однако для данного приложения необходимы некоторые изменения и добавления. Одно из них — добавление категорий книг, как указано в требованиях.

Кроме того, необходимо иметь возможность добавления в существующую базу данных информации об адресах доставки, условиях платежа и т.п.

Мы уже знаем, как создавать интерфейс с базой данных MySQL средствами PHP, потому эта часть решения не должна вызывать каких-либо затруднений.

Отслеживание выбираемого товара

Существует два основных метода отслеживания выбираемого посетителем товара. Один из них состоит в помещении выбираемых элементов в базу данных, а второй — в использовании переменной сеанса.

Использование переменной сеанса для отслеживания выбираемых элементов в процессе переходов между страницами проще в реализации, поскольку не требует постоянных запросов базы данных. Кроме того, этот метод позволяет избежать загромождения базы данных ненужными данными, поступаемыми от посетителей, которые только просматривают каталог и часто меняют свои решения.

В данной связи потребуется разработать переменную сеанса или набор переменных для хранения выбранных пользователем элементов. Когда пользователь завершает посещение магазина и выполняет расчет, эта информация помещается в базу данных в качестве регистрации транзакции.

Кроме того, эти данные могут использоваться для отображения в углу страницы текущего состояния тележки, чтобы посетитель в любой момент видел предстоящую сумму расходов.

Платежи

В этом проекте осуществляется прием заказа посетителя и сведений, касающихся доставки. Реальная обработка платежей здесь не рассматривается. Существует большое разнообразие систем платежей. Каждая из них имеет свою реализацию. Мы напишем фиктивную функцию, которую можно будет заменить интерфейсом любой выбранной системы.

Системы платежей обычно призваны действовать в определенных географических районах. Функционирование различных интерфейсов обработки платежей в реальном времени, в основном, подобно. Потребуется организовать учетные записи в банке для карточек, которые будут приниматься к оплате. Провайдер системы платежей укажет, какие параметры потребуется в нее передавать.

Система платежей будет передавать данные в банк и возвращать код успешного действия либо один из множества различных типов кодов ошибок. В обмен на передачу данных система платежей будет взымать сбор за установку либо годовой сбор, а также сбор, основанный на количестве или сумме транзакций. Некоторые провайдеры назначают плату даже за отклоненные транзакции.

Системе платежей необходима информация о клиенте (например, номер кредитной карточки), идентифицирующая информация от владельца магазина (чтобы указать, какая учетная запись будет применяться для оплаты), а также общая сумма транзакции.

Сумму заказа можно извлечь из переменной сеанса покупательской тележки. Окончательная информация заказа будет занесена в базу данных, а переменная сеанса будет удалена.

Интерфейс администрирования

Помимо сказанного выше, будет создан интерфейс администрирования, который позволяет добавлять, удалять и редактировать информацию о книгах и категориях в базе данных.

Один из часто используемых элементов редактирования — изменение цены товара (например, для специальных предложений). Это означает, что сохранение заказа клиента предусматривает и хранение цены, уплаченной за товар. Если записи отражают лишь позиции, заказанные каждым клиентом, и текущую цену каждого наименования, это существенно усложняет систему учетных записей. Кроме того, это означает, что когда клиент возвращает или обменивает товар, ему будут возвращаться лишние денежные средства.

Мы не будем создавать интерфейс отслеживания заказов и реализации для данного примера. Его можно будет добавить в систему по мере возникновения потребностей.

Обзор решения

Подведем итоги.

Существуют два основных представления системы: пользовательское и администраторское. С учетом необходимых функциональных возможностей созданы две блок-схемы системы — по одной для каждого представления. Они показаны на рис. 25.1 и. 25.2.

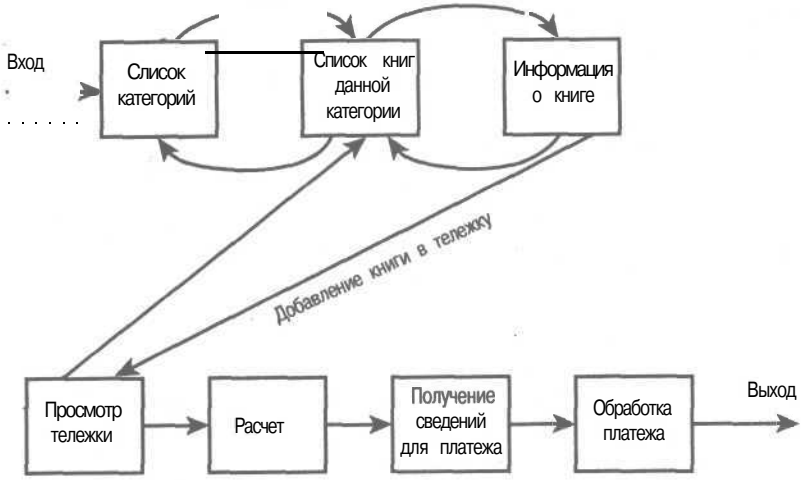


РИС. 25.1 *Система Book-O-Rama в представлении пользователя дает возможность просматривать книги по категориям и сведения о них, добавлять книги в тележку и приобретать их.*

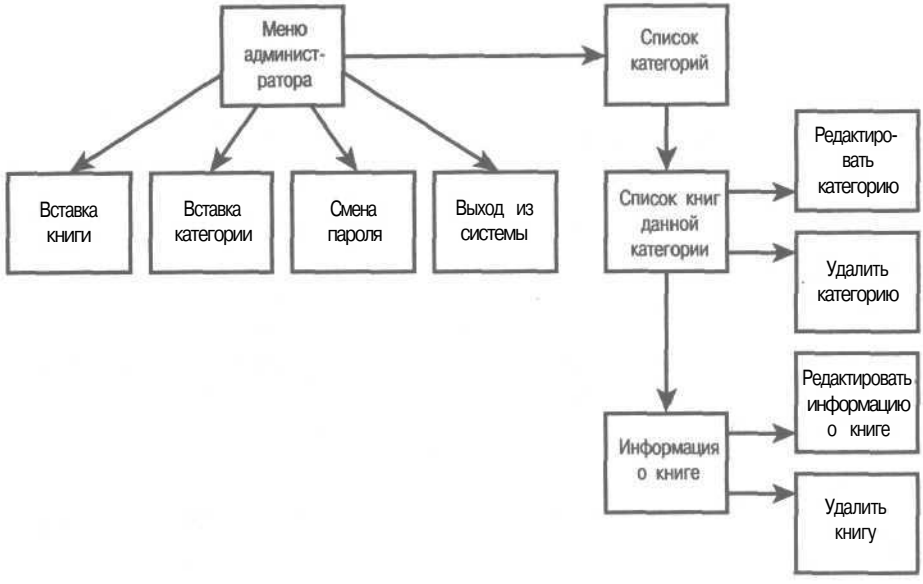


РИСУНОК **25.2** *Система Book-O-Rama в представлении администратора позволяет выполнять добавление, редактирование и удаление книг и категорий.*

На рис. 25.1 показаны главные ссылки между сценариями в пользовательской части сайта. Клиент сначала открывает главную страницу, в которой перечислены все категории книг сайта. Отсюда можно перейти к определенной категории книг, а затем к информации по отдельной книге.

Пользователю предоставляется ссылка для добавления выбранной книги в тележку. На этапе просмотра тележки можно рассчитаться и покинуть магазин.

На рис. 25.2 показан интерфейс администратора. В нем больше сценариев, но не особенно много нового кода. Эти сценарии позволяют администратору входить в систему и добавлять новые книги и категории.

Простейший способ реализовать редактирование и удаление книг и категорий — отобразить для администратора несколько отличную версию интерфейса пользователя сайта. Администратор по-прежнему будет иметь возможность просматривать категории и книги, но вместо доступа к покупательской тележке он может переходить к определенной книге или категории, а затем редактировать или удалять ее. Создание сценариев, одновременно пригодных для обычных пользователей и администраторов, позволяет экономить время и трудозатраты.

Ниже перечислены три основных кодовых модуля для приложения:

- Каталог.
- Покупательская тележка и обработка заказов (эти функции здесь объединены, поскольку тесно взаимосвязаны.)
- Администрирование.

Как и в предыдущем проекте, будет создаваться набор библиотек функций. В этом проекте применяется API-интерфейс функций, подобный использованному в предыдущем проекте. Попытаемся объединить фрагменты кода, отвечающие за вывод HTML-содержимого, в одну библиотеку. Это соответствует принципу разделения логики и содержимого и, что еще важнее, упрощает чтение и сопровождение кода.

Кроме того, потребуется внести небольшие изменения в базу данных Book-O-Rama. База данных **book_sc** (Shopping Cart) переименована, чтобы отличать базу данных покупательской тележки от той, что построена в части 2.

Весь код данного проекта содержится на сопровождающем CD-ROM. Перечень файлов приложения приведен в табл. 25.1.

Таблица 25.1 Файлы приложения Shopping Cart

Имя	*Модуль*	*Описание*
index.php	Каталог	Титульная страница сайта. Отображает список категорий системы.
show_cat.php	Каталог	Отображает все книги определенной категории.
show_book.php	Каталог	Отображает данные по определенной книге.
show_cart.php	Покупательская тележка	Отображает содержимое покупательской тележки. Кроме того, используется для добавления элементов в тележку.
checkout.php	Покупательская тележка	Представляет пользователю все данные заказа. Принимает информацию по доставке.
purchase.php	Покупательская тележка	Принимает информацию по платежу от пользователя.
process.php	Покупательская тележка	Обрабатывает данные платежа и добавляет заказ в базу данных.
login.php	Администрирование	Позволяет администратору входить в систему для внесения изменений.

Имя	Модуль	Описание
logout.php	Администрирование	Реализует выход администратора из системы.
admin.php	Администрирование	Главное меню администрирования.
change_password_form.php	Администрирование	Форма, позволяющая администратору изменять свой пароль.
change_password.php	Администрирование	Изменяет пароль администратора.
insert_category_form.php	Администрирование	Форма, позволяющая администратору добавлять в базу данных новую категорию.
insert_category.php	Администрирование	Вставляет новую категорию в базу данных.
insert_book_form.php	Администрирование	Форма, позволяющая администратору добавлять в систему новую книгу.
insert_book.php	Администрирование	Добавляет новую книгу в базу данных.
edit_category_form.php	Администрирование	Форма, позволяющая администратору редактировать категорию.
edit_category.php	Администрирование	Обновляет категорию в базе данных.
edit_book_form.php	Администрирование	Форма, позволяющая администратору редактировать информацию о книге.
edit_book.php	Администрирование	Обновление информации о книге в базе данных.
delete_category.php	Администрирование	Удаляет категорию из базы данных.
delete_book.php	Администрирование	Удаляет книгу из базы данных.
book_sc_fns.php	Функции	Набор подключаемых файлов.
admin_fns.php	Функции	Набор функций, используемых сценариями администрирования.
book_fns.php	Функции	Набор функций хранения и извлечения данных о книгах.
order_fns.php	Функции	Набор функций хранения и извлечения данных заказа.
output_fns.php	Функции	Набор функций вывода HTML-содержимого.
data_valid_fns.php	Функции	Набор функций проверки допустимости вводимых данных.
db_fns.php	Функции	Набор функций для подключения к базе данных book_sc.
user_auth_fns.php	Функции	Набор функций аутентификации администраторов.
book_sc.sql	SQL	SQL-код создания базы данных book_sc.
populate.sql	SQL	SQL-код для вставки данных в базу данных book_sc.

Теперь рассмотрим реализацию каждого модуля.

ПРИМЕЧАНИЕ

Это приложение содержит большой объем кода. Значительная часть его реализует функциональные возможности, которые были изучены ранее (особенно в предыдущей главе) и к которым относится хранение данных и извлечение информации из базы данных, а также аутентификация администратора. Этот код рассматривается кратко, а основная часть времени уделяется функциям работы с покупательской тележкой.

Чтобы код проекта работал как задумано, необходимо задействовать режим magic quotes. Если это не выполнено, потребуется применять функцию **addslashes()** ко вводимой в базу данных MySQL информации, а к извлекаемым из нее данным — функцию **stripslashes()**. Упомянутый режим служит как удобное сокращение.

Режим magic quotes можно задействовать для отдельных каталогов с помощью следующих директив файла **.htaccess:**

php_value `magic_quotes_gpc` on (для PHP 4)

либо

`php3_magic_quotes_gpc` on (для PHP 3)

Реализация базы данных

Как упоминалось ранее, в базу данных Book-O-Rama, представленную в части 2, вносятся небольшие изменения.

SQL-код создания базы данных **book_sc** приведен в листинге 25.1.

Листинг 25.1 book_sc.sql — SQL-код создания базы данных book_sc

```
create database book_sc;

use book_sc;

create table customers
 (
   customerid int unsigned not null auto_increment primary key,
   name char(40) not null,
   address char(40) not null,
   city char(20) not null,
   state char(20),
   zip char(10),
   country char(20) not null
);

create table orders
 (
   orderid int unsigned not null auto_increment primary key,
   customerid int unsigned not null,
   amount float(6,2),
   date date not null,
   order_status char(10),
   ship_name char(40) not null,
   ship_address char(40) not null,
   ship_city char(20) not null,
   ship_state char(20),
   ship_zip char(10),
   ship_country char(20) not null
);
```

```
create table books
(
    isbn char(13) not null primary key,
    author char(30),
    title char(60),
    catid int unsigned,
    price float(4,2) not null,
    description varchar(255)
);
create table categories
(
  catid int unsigned not null auto_increment primary key,
  catname char(40) not null
);
create table order_items
(
  orderid int unsigned not null,
  isbn char(13) not null,
  item_price float(4,2) not null,
  quantity tinyint unsigned not null,
  primary key (orderid, isbn)
);
create table admin
(
  username char(16) not null primary key,
  password char(16) not null
);
grant select, insert, update, delete
on book_sc.*
to book_sc@localhost identified by 'password';
```

Исходный интерфейс Book-O-Rama вполне подходит, однако возникает ряд дополнительных требований, поскольку база данных должна быть доступной в интерактивном режиме.

Ниже перечислены изменения, внесенные в исходную базу данных:

- Добавлены поля адресов для клиентов — это особенно важно сейчас, когда создается более реалистичное приложение.
- Добавлен адрес доставки заказа. Контактный адрес клиента не всегда совпадает с адресом доставки, особенно когда приобретаются подарки.
- Добавлена таблица категорий, а также идентификатор категорий в таблицу книг. Сортировка книг по категориям упрощает просмотр сайта.
- В таблицу **order_items** добавлен элемент **item_price** (цена товара), дабы учесть возможность изменения цены товара. Необходимо знать цену товара на момент, когда клиент его заказывает.
- Добавлена таблица admin для хранения входного имени и пароля администратора.
- Удалена таблица рецензий — их можно реализовать в качестве расширения проекта. Вместо этого для каждой книги существует поле описания, которое содержит краткую аннотацию.

Чтобы создать в системе базу данных, в среде MySQL потребуется выполнить сценарий **book_sc.sql,** имея права привилегированного пользователя (root):

```
mysql -u root -p < book_sc.sql
```

(Следует ввести пароль привилегированного пользователя.)

Предварительно необходимо изменить пароль пользователя **book_sc** (по умолчанию он имеет значение **password).**

Кроме того, подключен файл тестовых данных с именем **populate.sql**. Тестовые данные можно поместить в базу данных, запустив этот сценарий в среде MySQL.

Реализация интерактивного каталога

Это приложение содержит три сценария, связанных с каталогами: главной страницы, страницы категорий и страницы информации о книге.

Титульная страница сайта генерируется через сценарий **index.php** Вывод этого сценария показан на рис. 25.3.

Можно заметить, что кроме списка категорий окно содержит ссылку на покупательскую тележку (в правом верхнем углу) и итоговые данные по содержимому тележки. Эти элементы содержатся на каждой странице, открываемой в процессе просмотра и выбора товара.

Когда пользователь выполняет щелчок на одной из категорий, открывается страница категорий, генерируемая сценарием **show_cat.php.** Страница категорий для книг, посвященных Internet, показана на рис. 25.4.

Все книги категории Internet перечислены в виде ссылок. Когда пользователь выполняет щелчок на одной из них, открывается страница информации о соответствующей книге (см. рис. 25.5).

На этой странице помимо View Cart (просмотр тележки) находится ссылка Add to Cart (добавить в тележку), позволяющая выбирать товар. Мы вернемся к ней позже, когда будем создавать собственно покупательскую тележку.

Рассмотрим каждый из трех сценариев.

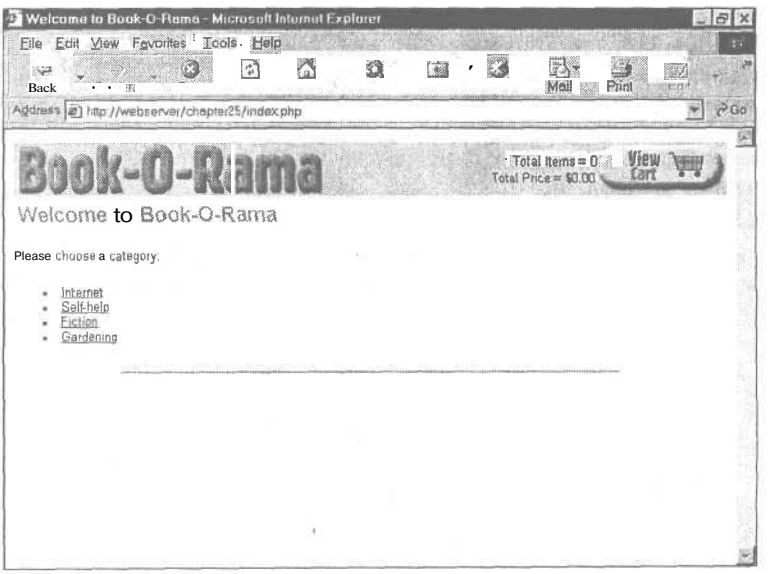


РИСУНОК 25.3
Титульная страница сайта содержит список категорий книг, доступных для приобретения.

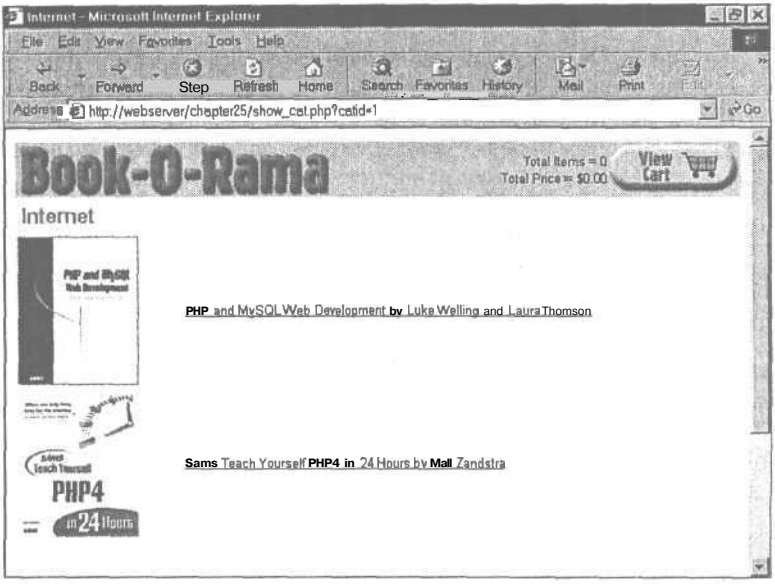


РИСУНОК 25.4 *Каждая книга категории сопровождается изображением обложки.*

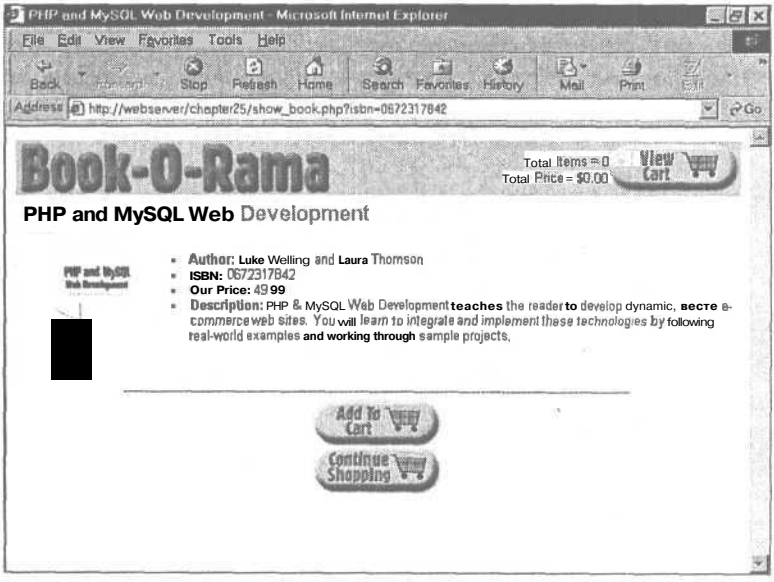


РИСУНОК 25.5. *С каждой книгой связана информационная страница, содержащая краткое описание.*

Список категорий

Первый сценарий, **index.php,** выводит все категории из базы данных (см. листинг 25.2).

Листинг 25.2 index.php — сценарий вывода титульной страницы сайта

```
<?
   include('book_sc_fns.php');
   // Для покупательской тележки требуется запуск сеанса
   session_start();
   do_html_header("Welcome to Book-O-Rama");

   echo "<p>Please choose a category:</p>";

   // Извлечение категорий из базы данных
   $cat_array = get_categories();

   // Отображение ссылок на страницы категорий
   display_categories($cat_array);

   // Если пользователь вошел в систему с правами
   // администратора, отобразить ссылки на добавление,
   // удаление и редактирование категорий
   if(session_is_registered("adminuser"))
   {
      display_button("admin.php", "admin-menu", "Admin Menu");
   }
   do_html_footer();
?>
```

Сценарий начинается с подключения файла **book_sc_fns.php**, который содержит все библиотеки функций для данного приложения.

После этого необходимо запустить сеанс, который необходим для корректного функционирования покупательской тележки. Сеанс используется каждой страницей сайта.

Сценарий включает вызовы функций вывода HTML-содержимого, таких как **do_html_header() и do_html_footer()** (обе находятся *в* файле **output_fns.php).**

Кроме того, предусмотрен код проверки, что пользователь вошел в систему с правами администратора. В этом случае ему предоставляются другие средства навигации. Мы вернемся к этому в разделе, посвященном функциям администрирования.

Ниже показана наиболее важная часть сценария:

```
// Извлечение категорий из базы данных
$cat_array = get_categories();

// Отображение ссылок на страницы категорий
display_categories($cat_array);
```

Функции **get_categories() и display_categories()** содержатся, соответственно, в библиотеках **book_fns.php и output_fns.php.** Функция **get_categories()** возвращает массив категорий системы, который затем передается в функцию **display_categories()**. Код функции **get_categories()** приводится в листинге 25.3.

Листинг 25.3 функция get_categories() из библиотеки output_fns.php — извлекает из базы данных список категорий

```
function get_categories()
{
    // запрос выбора списка категорий в базе данных
    $conn = db_connect();
    $query = "select catid, catname
                from categories";
    $result = @mysql_query($query);
    if (!$result)
       return false;
    $num_cats = @mysql_num_rows($result);
    if (num_cats =0)
       return false;
    $result = db_result_to_array($result);
    return $result;
}
```

Функция осуществляет соединение с базой данных и извлечение списка всех идентификаторов и имен категорий. Здесь используется ранее созданная функция **db_result_to_array()** из **файла db_fns.php** (см. листинг 25.4). Функция принимает идентификатор результата от MySQL и возвращает массив строк с числовой индексацией, где каждая строка представляет собой ассоциативный массив.

Листинг 25.4 Функция db_result_to_array() библиотеки db_fns.php — преобразует идентификатор результата MySQL в массив результатов

```
function db_result_to_array($result)
{
    $res_array = array();
    for ($count=0; $row = @mysql_fetch_array($result); $count++)
       $res_array[$count] = $row;

    return $res_array;
}
```

В нашем случае этот массив возвращается в сценарий **index.php**, где передается в **display_categories()** из библиотеки **output_fns.php**. Эта функция отображает каждую категорию в виде ссылки на страницу, содержащую книги данной категории. Код функции показан в листинге 25.5.

Листинг 25.5 Функция display_categories() из библиотеки outputjns.php - отображает массив категорий в виде списка ссылок на категории

```
function display_categories($cat_array)
{
   if (!is_array($cat_array))
   {
       echo "No categories currently available<br>";
       return;
   }
   echo "<ul>";
   foreach ($cat_array as $row)
   {
      $url = "show_cat.php?catid=".($row["catid"]);
      $title = $row["catname"];
      echo "<li>";
```

```
    do_html_url($url, $title);
  }
  echo "</ul>";
  echo "<hr>";
}
```

Функция преобразует каждую категорию базы данных в ссылку. Все ссылки передаются в следующий сценарий, **show.cat.php,** но каждая из них имеет свой параметр, идентификатор категории (catid). (Это уникальное число, сгенерированное MySQL и применяемое для идентификации категории.)

Упомянутый параметр определяет, какую категорию будет отображать следующий сценарий.

Список книг в категории

Процесс вывода рписка книг определенной категории аналогичен рассмотренному выше. Вывод осуществляет сценарий **show_cat.php**, показанный в листинге 25.6.

Листинг 25.6 show_cat.php — этот сценарий отображает книги определенной категории

```
<?
  include('book_sc_fns.php');
  // Покупательская корзина должна иметь запущенный сценарий
  session_start();
  $name = get_category_name($catid);

  do_html_header($name);

  // получение информации о книге из базы данных
  $book_array = get_books($catid);

  display_books($book_array);

  // если пользователь вошел в систему с правами администратора,
  // отобразить ссылки на добавление и удаление книг
  if(session_is_registered("admin_user"))
  {
    display_button("index.php", "continue", "Continue Shopping");
    display_button("admin.php", "admin-menu", "Admin Menu");
    display_button("edit_category_form.php?catid=$catid", "edit-category",
                   "Edit Category");
  }
  else
    display_button("index.php", "continue-shopping", "Continue
  Shopping");

  do_html_footer();
?>
```

Структура этого сценария во многом подобна структуре сценария вывода титульной страницы с той лишь разницей, что вместо категорий извлекаются книги.

Сначала стандартным способом запускается сеанс с помощью функции session_start(), а затем с использованием функции **get_category_name()** передаваемый идентификатор категории преобразуется в имя категории:

```
$name = get_category_name($catid);
```

Эта функция выполняет поиск имени категории в базе данных. Код функции показан в листинге 25.7.

Листинг 25.7 Функция get_category_name() библиотеки book_fns.php — преобразует идентификатор категории в имя категории

```
function get_category_name ($catid)
{
    // запрос в базе данных имени категории для данного идентификатора
    $conn = db_connect() ;
    $query = "select catname
                from categories
                where catid = $catid";
    $result = @mysql_query ($query);
    if (!$result)
       return false;
    $num_cats = @mysql_num_rows($result) ;
    if  ($num_cats =0)
        return false;
    $result = mysql_result ($result, 0, "catname"),-
    return $result;
}
```

После извлечения имени категории можно отобразить HTML-заголовок и перейти к извлечению из базы данных списка книг, относящихся к выбранной категории:

```
$book_array = get_books ($catid);
display_books ($book_array);
```

Функции **get_books() и display_books()** во многом подобны функциям **get_categories() и display_categories(),** поэтому они здесь не рассматриваются. Единственное отличие состоит в том, что информация извлекается из таблицы книг, а не таблицы категорий.

Функция **display_books()** предоставляет ссылку на каждую книгу категории при помощи сценария **show_book.php.** Опять-таки, каждая ссылка сопровождается параметром в виде суффикса. На этот раз он определяет ISBN конкретной книги.

В заключительном фрагменте сценария **show_cat.php** содержится код для отображения дополнительных опций в случае, когда в систему входит администратор. Этот код рассматривается в разделе, посвященном функциям администрирования.

Отображение информации о книгах

Сценарий **show_book.php** принимает ISBN в качестве параметра, а затем извлекает и отображает сведения о данной книге. Код сценария показан в листинге 25.8.

Листинг 25.8 show_book.php — этот сценарий отображает данные по определенной книге

```
<?
   include ('book_sc_fns.php');
   // Покупательская тележка требует наличия запущенного сеанса
   session_start();

   // извлечение книги из базы данных
   $book = get_book_details($isbn);
   do_html_header($book["title"]);
   display_book_details($book);

   // установка URL-адреса для кнопки "continue"
   $target = "index.php";
   if($book["catid"])
   {
      $target = "show_cat.php?catid=".$book["catid"];
   }
```

```
    // если пользователь вошел в систему с правами
    // администратора, отобразить ссылки на редактирование книг
    if( check_admin_user() )
    {
      display button ( "edit_book_form.php?isbn=$isbn", "edit-item", "Edit
    Item");
      display_button("admin.php", "admin-menu", "Admin Menu");
      display_button($target, "continue", "Continue");
    }
    else
    {
      display_button("show_cart.php?new=$isbn", "add-to-cart",
                     "Add ".$book["title"]." To My Shopping Cart");
      display_button($target, "continue-shopping", "Continue Shopping");
    }
    do html_footer();
?>
```

Этот сценарий также очень похож на сценарии вывода двух предыдущих страниц. Сначала запускается сеанс, а затем с помощью строки:

```
$book = get_book_details($isbn);
```

из базы данных извлекается информация о книге. Для вывода данных в формате HTML используется следующий код:

```
display_book_details($book);
```

Заметим, что функция **display_book_details()** выполняет поиск файла изображения для книги, который выглядит как **images/$isbn.jpg**. Если файл не существует, изображение не выводится.

Оставшаяся часть сценария устанавливает средства навигации. Обычному пользователю предоставляется кнопка "Continue Shopping" (продолжить покупку), возвращающая на страницу категорий, и кнопка Add to Cart для добавления книг в покупательскую тележку. Если пользователь вошел в систему с правами администратора, ему предлагаются другие опции.

На этом обзор системы каталогов завершается. Перейдем к коду реализации покупательской тележки.

Реализация покупательской тележки

Функции покупательской тележки тесно связаны с переменной сеанса **$cart**. Это ассоциативный массив, в котором номера ISBN служат ключами, а объемы заказа – значениями. Например, если поместить в тележку один экземпляр данной книги, массив будет содержать следующую запись:

```
0672317842 => 1
```

Когда элементы помещаются в тележку, происходит их добавление в массив. Во время просмотра содержимого тележки массив **$cart** используется для поиска полной информации о товарах в базе данных.

Кроме того, используются две других переменных сеанса для управления отображением в заголовке данных Total Items (количество элементов) и Total Price (сумма заказа) — соответственно, **$items** и **$total_price**.

Использование сценария show_cart.php

Обзор реализации покупательской тележки начнем со сценария **show_cart.php**. Он отображает страницу, которая открывается после щелчка на ссылках View Cart либо Add to Cart. Если сценарий **show_cart.php** вызывается без параметров, отображается содержимое тележки. Если в качестве параметра указывается номер ISBN, соответствующая этому коду позиция добавляется в тележку.

Для полного понимания сначала обратимся к рис. 25.6.

В данном случае щелчок на ссылке View Cart был выполнен, когда тележка пуста. Другими словами, для покупки не выбрана еще ни одна позиция.

На рис. 25.7 показана тележка, когда для покупки выбраны две книги. В данном случае страница открыта после щелчка на ссылке Add to Cart в странице, сгенерированной сценарием **show_book.php** для книги, которую вы читаете (имеется в виду ее англоязычный вариант). Если внимательно посмотреть на строку URL-адреса, можно заметить, что на этот раз сценарий вызывается с параметром. Параметр называется new и имеет значение **0672317842** — номер ISBN книги, которая только что помещена в тележку.

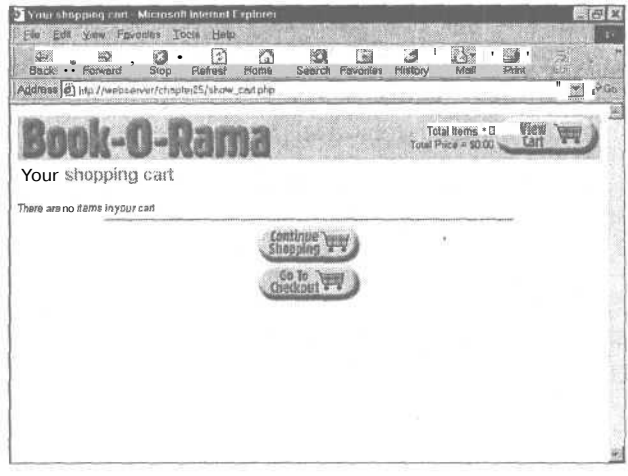


РИСУНОК 25.6
Сценарий show_cart.php, вызванный без параметров, просто отображает содержимое тележки.

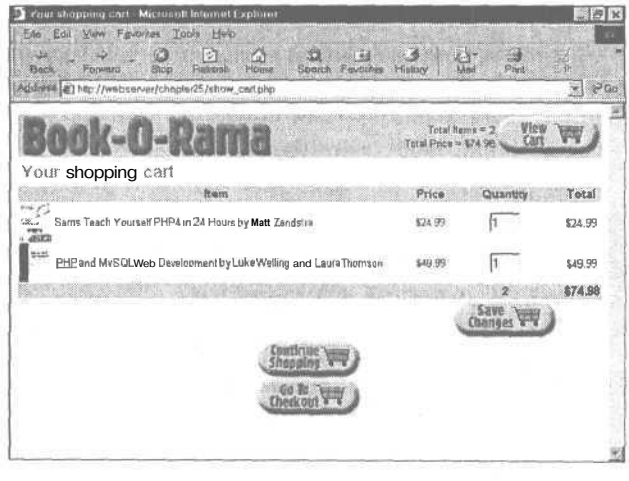


РИСУНОК 25.7
Сценарий show_cart.php с параметром new помещает в тележку новый элемент.

На этой странице предоставляются еще две опции. Это кнопки Save Changes (сохранить изменения) и Go To Checkout (произвести расчет). Первую из них можно использовать для изменения количества элементов в тележке. Для этого следует непосредственно изменить количество экземпляров в полях Quantity и щелкнуть на кнопке Save Changes. В сущности, это кнопка отправки формы, которая обеспечивает возврат в сценарий **show_cart.php** с целью обновления содержимого тележки.

Вторая кнопка выбирается пользователем, когда он готов покинуть магазин. Мы вернемся к ней несколько позже.

Пока рассмотрим код сценария **show_cart.php**, показанный в листинге 25.9.

Листинг 25.9 show_cart.php — этот сценарий управляет покупательской тележкой

```
<?
  include ('book_sc_fns.php');
  // Покупательская тележка требует наличия запущенного сеанса
  session_start();

  if ($new)
  {
    //выбран новый элемент
    if (!session_is_registered("cart"))
    {
      $cart = array();
      session_register("cart");
      $items = 0;
      session_register("items");
      $total_price = "0.00";
      session_register("total_price");
    }
    if($cart[$new])
      $cart[$new]++;
    else
      $cart[$new] = 1;
    $total_price = calculate_price($cart);
    $items = calculate_items($cart);

  }
  if($save)
  {
    foreach ($cart as $isbn => $qty)
    {
      if($$isbn=="0")
        unset($cart[$isbn]);
      else
        $cart[$isbn] = $$isbn;
    }
    $total_price = calculate_price($cart);
    $items = calculate_items($cart);
  }
  do_html_header("Your shopping cart");

  if($cart&&array_count_values($cart))
    display_cart($cart);
  else
  {
    echo "<p>There are no items in your cart";
    echo "<hr>";
  }
  $target = "index.php";
```

```
    // если в тележку только что добавлен новый элемент,
    // продолжаем выбор книг данной категории
    if($new)
    {
      $details = get_book_details($new);
      if($details["catid"])
        $target = "show_cat.php?catid=".$details["catid"];
    }
    display_button($target, "continue-shopping", "Continue Shopping");
    $path = $PHP_SELF;
    $path = str_replace("show_cart.php", "", $path);
    display_button("https://".$SERVER_NAME.$path."checkout.php",
                   "go-to-checkout", "Go To Checkout");
    do_html_footer();
?>
```

Этот сценарий содержит три главных части: отображение содержимого тележки, добавление в нее элементов и сохранение изменений. Все части рассматриваются в трех последующих разделах.

Просмотр содержимого тележки

Содержимое тележки будет отображаться независимо от страницы, на которой был произведен щелчок на ссылке. В общем случае после щелчка на ссылке View Cart выполняется лишь следующая часть кода:

```
if($cart&&array_count_values($cart))
  display_cart($cart);
else
{
  echo "<p>There are no items in your cart";
  echo "<hr>";
}
```

Очевидно, что если тележка имеет непустое содержимое, будет вызываться функция **display_cart()**. Когда тележка пуста, пользователю выводится соответствующее сообщение.

Функция **display_cart()** лишь печатает содержимое тележки в формате HTML, как показано на рис. 25.6 и 25.7. Код функции содержится в файле **outputjns.php** и представлен в листинге 25.10. Хотя это функция отображения, она довольно сложна и заслуживает отдельного рассмотрения.

Листинг 25.10 Функция display_cart() библиотеки outputjns.php — печатает содержимое покупательской тележки

```
function display_cart($cart, $change = true, $images = 1)
{
  // отображение элементов в покупательской тележке
  // возможно внесение изменений (true или false)
  // возможно включение изображений (1 — да, 0 — нет)

  global $items;
  global $total_price;

  echo "<table border = 0 width = 100% cellspacing = 0>
        <form action = show_cart.php method = post>
        <trXth colspan = ". (1+$images) ." bgcolor=\"#cccccc\">Item</th>
        <thbgcolor=\"#cccccc\">Price</th><thbgcolor=\"#cccccc\">Quantity</th>
        <th bgcolor=\"#cccccc\">Total</th></tr>";
```

```
// отображение каждого элемента в виде строки таблицы
foreach ($cart as $isbn => $qty)
{
   $book = get_book_details($isbn);
   echo "<tr>";
   if($images ==true)
   {
      echo "<td align = left>";
      if (file_exists("images/$isbn.jpg"))
      {
          $size = GetImageSize("images/".$isbn.".jpg");
          if ($size[0]>0 && $size[1]>0)
          {
             echo "<img src=\"images/".$isbn.".jpg\" border=0 ";
             echo "width = ". $size[0]/3 ." height = " .$size[1]/3 . ">";
          }
      }
      else
          echo " ";
      echo "</td>";
   }
   echo "<td align = left>";
   echo "<a href = \"show_book.php?isbn=".$isbn."\">"
                    .$book["title"]."</a>by " .$book["author"];
   echo "</td><td align = center>$".number_format($book["price"], 2) ;
   echo "</td><td align = center>";

   // если изменения допускаются, количества указываются
   // в текстовых полях
   if ($change == true)
      echo "<input type = text name = \"$isbn\" value = $qty size = 3>";
   else
      echo $qty;
   echo "</td><td align = center>$".number_format($book["price"]*$qty,2)
        ."</td></tr>\n";
}
// отображение итоговой строки
echo "<tr>
         <th colspan = ". (2+$images) ." bgcolor=\"#cccccc\"> </td>
         <th align = center bgcolor=\"#cccccc\">
            $items
         </th>
         <th align = center bgcolor=\"#cccccc\">
            \$".number_format($total_price, 2).
         "</th>
      </tr>";

// отображение кнопки сохранения изменений
if($change == true)
{
   echo "<tr>
            <td colspan = ". (2+$images) ."> </td>
            <td align = center>
               <input type = hidden name = save value = true>
               <input type = image src = \"images/save-changes.gif\"
                     border = 0 alt = \"Save Changes\">
            </td>
            <td> </td>
      </tr>";
}
echo "</form></table>";
}
```

Ниже описывается базовый алгоритм, реализованный в функции:

1. Циклический обход каждого элемента тележки и передача его номера TSBN в функцию **get_book_details(),** что позволяет получить сводную информацию по каждой книге.
2. Для каждой книги предоставляется изображение, если оно существует. Здесь используются дескрипторы высоты и ширины изображения, чтобы несколько уменьшить его размеры. В результате изображения немного искажаются, но они достаточно малы, чтобы это не составляло серьезной проблемы. (Если это вас беспокоит, можно всегда изменить размеры изображений с помощью библиотеки **gd,** рассмотренной в главе 19, либо вручную создать изображения другого размера для каждого продукта.)
3. Преобразовать каждую запись тележки в ссылку на соответствующую книгу. Другими словами — на сценарий **show_book.php** с передачей номера ISBN в качестве параметра.
4. Если функция вызывается, когда для параметра **$chande** установлено значение true (либо вообще не установлено — значение **true** принимается по умолчанию), отображаются текстовые поля ввода количества. Они составляют форму, завершаемую кнопкой Save Changes. (При повторном использовании этой функции после осуществления расчета нельзя допускать, чтобы пользователь мог изменить свой заказ.)

Эта функция не содержит ничего особо сложного, но выполняет множество операций. Поэтому стоит внимательно изучить ее код.

Добавление элементов в тележку

Когда пользователь открывает страницу **show_cart.php** щелчком на кнопке Add To Cart, перед отображением содержимого тележки необходимо выполнить некоторые операции. В частности, следует поместить в тележку выбранный элемент.

Во-первых, если пользователь еще не помещал в тележку элементы, у него нет тележки, поэтому ее требуется создать:

```
if(!session_is_registered("cart"))
{
   $cart = array 0;
   session_register("cart");
   $items = 0;
   session_register("items");
   $total_price = "0.00";
   session_register("total_price");
}
```

Сначала тележка пуста.

Во-вторых, когда известно, что тележка создана, в нее можно добавлять элементы:

```
if ($cart[$new])
   $cart[$new]++;
else
   $cart[$new] = 1;
```

Здесь проверяется, не содержится ли уже элемент в тележке. Если да, количество данных книг увеличивается на единицу. В противном случае в тележку помещается новый элемент.

В-третьих, необходимо определить общую сумму заказа и количество элементов в тележке. Для этого применяются функции **calculate_price() и calculate_items()**:

```
$total_price = calculate_price ($cart);
$items = calculate_items ($cart);
```

Эти функции содержатся в библиотеке **book_fns.php**. Их код приводится в, соответственно, листингах 25.11 и 25.12.

Листинг 25.11 Функция **calculate_price()** библиотеки book_fns.php — вычисляет и возвращает общую стоимость содержимого тележки для покупок

```
function calculate_price ($cart)
{
  // вычисление общей стоимости всех элементов тележки
  $price = 0.0;
  if (is_array ($cart))
  {
    $conn = db_connect() ;
    foreach ($cart as $isbn => $qty)
    {
      $query = "select price from books where isbn= ' $isbn ' " ;
      $result = mysql_query ($query);
      if ($result)
      {
        $item_price = mysql_result ($result, 0, "price");
        $price +=$item_price*$qty;
      }
    }
  }
  return $price;
}
```

Можно видеть, что функция **calculate_price()** выполняет поиск в базе данных цены каждого элемента, помещенного в тележку. Этот процесс требует времени, поэтому, чтобы не повторять его чаще, чем необходимо, цена (и общее количество элементов) сохраняются в переменных сеанса. Повторные вычисления выполняются только в случаях, когда изменяется содержимое тележки.

Листинг 25.12 Функция **calculate_items()** библиотеки book_fns.php — вычисляет и возвращает общее количество элементов в тележке

```
function calculate_items ($cart)
{
  // подсчет общего количества элементов в тележке
  $items = 0;
  if (is_array ($cart))
  {
    foreach ($cart as $isbn => $qty)
    {
      $items += $qty;
    }
  }
  return $items;
```

Функция **calculate_items()** проще — она лишь складывает количества всех элементов тележки для получения итогового значения.

Сохранение изменений содержимого тележки

Когда сценарий **show_cart.php** вызывается щелчком на кнопке Save Changes, процесс несколько изменяется. В данном случае осуществляется передача формы. При внимательном рассмотрении кода можно заметить, что кнопка Save Changes является кнопкой отправки формы. Эта форма содержит скрытую переменную save. Если для переменной установлено значение, можно заключить, что сценарий вызван кнопкой Save Changes. Это означает, что пользователь мог изменить количества элементов, и эти изменения требуется сохранить.

Имена текстовых полей формы содержат номера ISBN книг, которые они представляют:

```
echo "<input type = text name = \"$isbn\" value = $qty size = 3>";
```

Теперь рассмотрим часть сценария, которая сохраняет изменения:

```
if($save)
   {
      foreach ($cart as $isbn => $qty)
      {
         if ($$isbn=="0")
            unset($cart[$isbn]);
         else
            $cart[$isbn] = $$isbn;
      }
      $total_price = calculate_price($cart);
      $items = calculate_items($cart);
}
```

Перебираются все элементы тележки и для каждой переменной **$isbn** проверяется значение переменной с соответствующим именем. Для этого используется обозначение **$$isbn** (переменная переменной).

(Переменные переменных рассматривались в главе 1.) Напомним, что ссылка на **$$isbn** означает ссылку на переменную, имя которой хранится в переменной **$isbn**. Речь идет о полях формы Save Changes.

Если в любом поле устанавливается значение 0, соответствующий элемент удаляется из тележки с использованием функции unset(). В противном случае содержимое тележки обновляется в соответствии со значениями полей формы:

```
if ($$isbn=="0")
   unset($cart[$isbn]);
else
   $cart[$isbn] = $$isbn
```

После обновления содержимого тележки повторно вызываются функции **calculate_price()** и **calculate_items()** для определения новых значений переменных сеанса **$total_price** и **$items**.

Печать итоговых данных в строке заголовка

Несложно заметить, что в строке заголовка каждой страницы сайта выводятся итоговые данные состояния покупательской тележки. Это осуществляется путем печати значений переменных сеанса **$total_price** и **$items** функцией **do_html_header()**.

Эти переменные регистрируются, когда пользователь впервые посещает страницу **show_cart.php**. Кроме того, необходимо реализовать логику для случаев, когда пользователь еще не открывал данную страницу. Эта логика также содержится в функции **do_html_header()**:

```
if(!$items) $items = "0";
if(!$total_price) $total_price = "0.00";
```

Выполнение расчета

Когда пользователь щелкает на кнопке Go to Checkout (перейти к окончательной оплате), запускается сценарий **cheekout.php.** Доступ к странице расчетов и последующим страницам должен осуществляться через SSL, но наше демонстрационное приложение этого не требует. (Дополнительные сведения о SSL содержатся в главе 15.)

Страница расчетов показана на рис. 25.8.

Данный сценарий требует, чтобы клиент ввел свой адрес (и адрес доставки, если они отличаются). Сценарий довольно прост, в чем легко убедиться, просмотрев код листинга 25.13.

Листинг 25.13 cheekout.php — этот сценарий принимает данные пользователя

```
<?
   // подключение набора функций
   include ('book_sc_fns.php');

   // запуск сеанса
   session_start();

   do_html_header("Checkout");

   if($cart&&array_count_values($cart))
   {
      display_cart($cart, false, 0);
      display_checkout_form($HTTP_POST_VARS);
   }
   else
      echo "<p>There are no items in your cart";

   display_button("show_cart.php", "continue-shopping", "Continue
    Shopping");

   do_html_footer();
?>
```

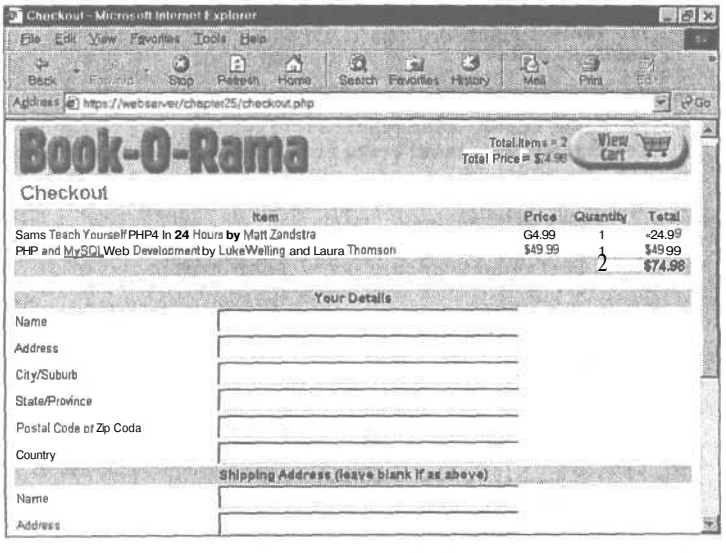


РИСУНОК 25.8
Сценарий cheekout.php принимает данные пользователя.

Сценарий не содержит ничего особо примечательного. Если тележка пуста, об этом выводится уведомление. В противном случае отображается форма, показанная на рис. 25.8.

Когда пользователь продолжает сеанс щелчком на кнопке Purchase в нижней части формы, запускается сценарий **purchase.php.** Вывод сценария показан на рис. 25.9.

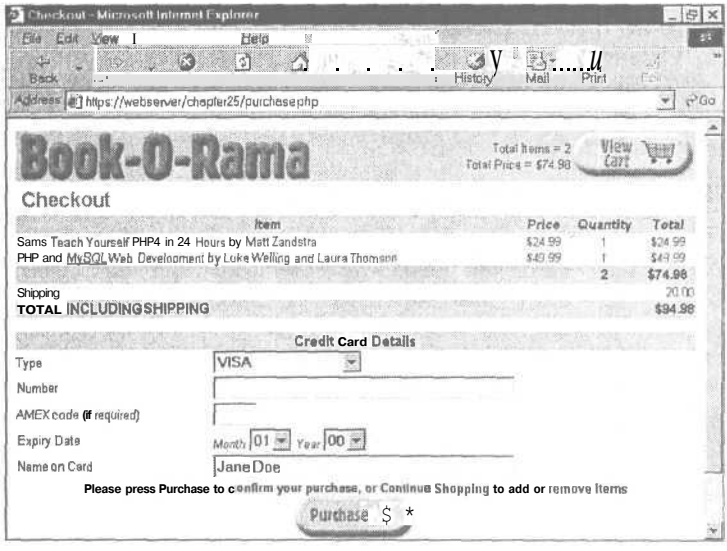


РИСУНОК 25.9
Сценарий purchase.php вычисляет сумму окончательного заказа и расходов на доставку, а также принимает данные платежа.

По сравнению с **checkout.php** этот сценарий немного сложнее. Его код показан *в* листинге 25.14.

Листинг 25.14 purchase.php — этот сценарий сохраняет заказ в базе данных и принимает данные платежа

```
<?
   include ('book_sc_fns.php');
   // запуск сеанса
   session_start();

   do_html_header("Checkout");

   // если форма заполнена
   if($from=='process'||$cart&&$name&&$address&&$city&&$zip&&$country)
   {
      // возможность вставки в базу данных
      if($from!='process')
      {
         if(!insert_order($HTTP_POST_VARS))
         {
            echo "Could not store data, please try again.";
            display_button("checkout.php", "back", "Back");
            do_html_footer();
            exit;
         }
      }
      // отображение содержимого тележки без изображений,
      // изменения не допускаются
      display_cart($cart, false, 0);
      display_shipping(calculate_shipping_cost());
```

```
    //получение данных кредитной карточки
    display_card_form($HTTP_POST_VARS);

    display_button("show_cart.php", "continue-shopping",
                   "Continue Shopping");
  }
  else
  {
    echo "You did not fill in all the fields, please try again.<hr>";
    echo "<form action = 'checkout.php' method = post>";

    // обратная передача данных, принятых этой страницей, чтобы
    // пользователю не пришлось повторно входить в систему
    foreach($HTTP_POST_VARS as $name => $value)
      echo "<input type = hidden name = $name value = '$value'>\n";
                                                    ft
    display_form_button("back", "Back") ;
    echo "</form>";
  )
  do_html_footer() ;
?>
```

Логика сценария проста: выполняется проверка, что пользователь заполнил форму, и вводится информация в базу данных с помощью функции **insert_order()**. Это простая функция, вставляющая сведения о клиенте в базу данных. Ее код показан в листинге 25.15.

Листинг 25.15 Функция insert_order() библиотеки order_fns.php — вставляет информацию о заказе в базу данных

```
function insert_order($order_details)
{
  global $total_price;
  global $cart;
  //извлечение данных заказа в виде переменных
  extract($order_details);

  //установка адреса доставки, совпадающего с адресом клиента

  if(!$ship_name&&!$ship_address&&!$ship_city&&
!$ship_state&&!$ship_  zip&&!$ship_
country)
  {
    $ship_name = $name;
    $ship_address = $address;
    $ship_city = $city;
    $ship_state = $state;
    $ship_zip = $zip;
    $ship_country = $country;
  !

  $conn = db_connect() ;

  //вставка адреса клиента
  $query = "select customerid from customers where
            name = '$name' and address = '$address'
            and city = '$city' and state = '$state'
            and zip = '$zip' and country = '$country'";
  $result = mysql_query($query);
  if(mysql_numrows($result)>0)
  {
    $customer_id = mysql_result($result, 0, "customerid");
  }
```

```
    else
    {
      $query = "insert into customers values
                C ',
     '$name','$address','$city','$state','$zip','$country')";
      $result = mysql_query($query);
      if (!$result)
          return false;
    }
    $query = "select customerid from customers where
                name = '$name' and address = '$address'
                and city = '$city' and state = '$state'
                and zip = ' $zip' and country = ' $country'";
    $result = mysql_query($query);
    if (mysql_numrows($result) >0)
      $customerid = mysql_result($result, 0, "customerid");
    else
      return false;
    $date = date("Y-m-d");
    $query = "insert into orders values
                ('', $customerid, $total_price, '$date', 'PARTIAL',
     '$ship_name',
                 '$ship_address','$ship_city','$ship_state','$ship_zip',
                  '$ship_country')";
    $result = mysql_query($query);
    if (!$result)
      return false;

    $query = "select orderid from orders where
                    customerid = $customerid and
                    amount > $total_price-.001 and
                    amount < $total_price+.001 and
                    date = '$date' and
                    order_status = 'PARTIAL' and
                    ship_name = '$ship_name' and
                    ship_address = '$ship_address' and
                    ship_city = '$ship_city' and
                    ship_state = '$ship_state' and
                    ship_zip = '$ship_zip' and
                    ship country = '$ship_country'";
    $result = mysql_query($query);
    if(mysql_numrows($result)>0)
      $orderid = mysql_result($result, 0, "orderid");
    else
      return false;

    // вставка каждой книги
    foreach($cart as $isbn => $quantity)
    {
      $detail = get_book_details($isbn);
      $query = "delete from order_items where
                   orderid = $orderid and isbn =  '$isbn'";
      $result = mysql_query($query);
      $query = "insert into order_items values
                   ('$orderid', '$isbn', ".$detail["price"].", $quantity) ";
      $result = mysql_query($query);
      if (!$result)
        return false;
    }
    return $orderid;
}
?>
```

Код функции довольно длинный, поскольку необходимо выполнить вставку данных пользователя, заказа, а также информации о каждой приобретаемой книге.

Затем определяется стоимость доставки по адресу клиента и выводится на экран с помощью следующей строки кода:

```
display_shipping(calculate_shipping_cost());
```

Код вычисления стоимости доставки (**calculate_shipping_cost()**) всегда возвращает значение $20. В реальном торговом сайте потребуется выбирать способ доставки, определять затраты на нее для различных адресов и соответствующим образом вычислять стоимость.

Затем отображается форма, чтобы пользователь ввел в нее данные кредитной карточки с помощью функции **display_card_form()** библиотеки **output_fns.php**.

Реализация платежа

Когда пользователь выполняет щелчок на кнопке Purchase, обрабатываются данные платежа с помощью сценария **process.php**. Результаты успешной платежной операции показаны на рис. 25.10.

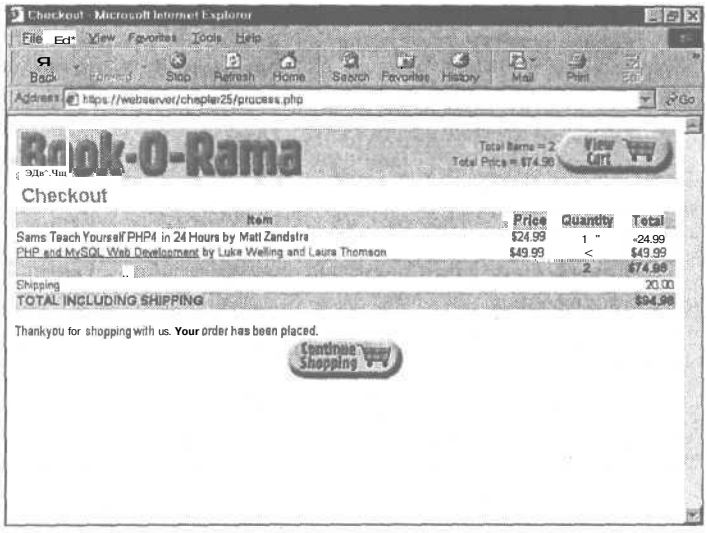


РИСУНОК 25.10
Транзакция была успешной и товар будет доставлен.

Код сценария **process.php** приводится **в листинге** 25.16.

Листинг 25.16 process.php — этот сценарий обрабатывает платеж и выводит его результаты

```
<?
  include ('book_sc_fns.php');
  // запуск сеанса
  session_start();

  do_html_header("Checkout");

  if($cart&&$card_type&&$card_number&&$card_month&&$card_year&&$card_name )
  {
    //отображение содержимого тележки без изображений,
    //изменения не допускаются
    display_cart($cart, false, 0);
```

```
    display_shipping(calculate_shipping_cost());

    if(process_card($HTTP_POST_VARS))
    {
      //удаление содержимого тележки
      session_destroy();
      echo "Thank you for shopping with us.  Your order has been placed.";
      display_button("index.php", "continue-shopping", "Continue Shopping");
    }
    else
    {
      echo "Could not process your card, please contact the card issuer or
            try again.";

      display_button("purchase.php", "back", "Back");
    }
  }
  else
  {
    echo "You did not fill in all the fields, please try again.<hr>";
    echo "<form action = 'purchase.php' method = post>";
    echo "<input type = hidden name = from value = process>\n";

    // обратная передача принятых страницей данных, чтобы пользователю не
    // пришлось повторно входить в систему
    foreach($HTTP_POST_VARS as $name => $value)
      echo "<input type = hidden name = $name value = '$value'>\n";

    display_form_button("back", "Back");
    echo "</form>";
  }
  do_html_footer();
?>
```

Настраиваемая часть сценария, касаемая обработки карточки, находится в следующих строках кода:

```
if(process_card($HTTP_POST_VARS))
{
  //удаление содержимого тележки
  session_destroy();
  echo "Than kyou for shopping with us.  Your order has been placed.";
  display_button("index.php", "continue-shopping", "Continue Shopping");
}
```

Как и в других фрагментах, где существует прямая ссылка на **$HTTP_POST_VARS**, необходимо задействовать отслеживание переменных (**track_vars**). Обрабатываются данные карточки клиента. В случае успешной транзакции сеанс прекращается.

В нашем упрощенном варианте функция обработки данных карточки просто возвращает значение **true.**

В реальном сайте потребуется принять решение, какой механизм транзакций будет использоваться. Существуют следующие возможности:

- Заключить договор с поставщиком транзакций. Здесь существует множество альтернатив, зависящих от района проживания. Некоторые из них предлагают банковские операции в реальном времени. Необходимость в таких операциях зависит от предлагаемой службы. Для предоставления интерактивных служб они, скорее всего, необходимы. Для доставки товара это менее важно. В любом случае поставщик освободит вас от ответственности хранения номеров кредитных карточек.

- Отправлять номера кредитных карточек зашифрованными сообщениями электронной почты, например, с помощью PGP или GPG, как упоминалось в главе 15. После получения и расшифровки электронной почты эти транзакции можно обрабатывать вручную.
- Хранить номера кредитных карточек в своей базе данных. Мы не рекомендуем этот метод, если вы должным образом не позаботились о защищенности системы. Дополнительные сведения приводятся в главе 15.

На этом завершается обзор покупательской тележки и модулей платежей.

Реализация интерфейса администрирования

Реализованный в этом проекте интерфейс администрирования достаточно прост. Он сводится к построению Web-интерфейса взаимодействия с базой данных, применяющем входную аутентификацию. Во многом повторяется код, используемый в главе 24. Код с краткими комментариями приводится ниже.

Интерфейс администрирования требует, чтобы пользователь вошел в систему через файл **login.php**, который затем выводит меню администратора, **admin.php**. Страница входа в систему показана на рис. 25.11. (Здесь для краткости опущен файл **login.php** — он почти точно соответствует файлу из главы 24. Его можно найти на CD-ROM.) Меню администрирования показано на рис. 25.12.

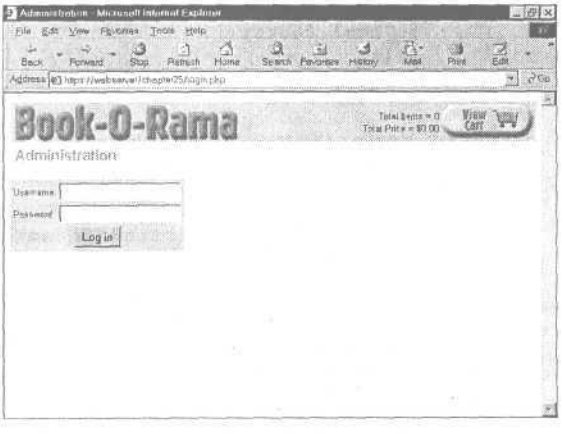


РИСУНОК 25.11
Доступ к функциям администрирования лежит через страницу входа в систему.

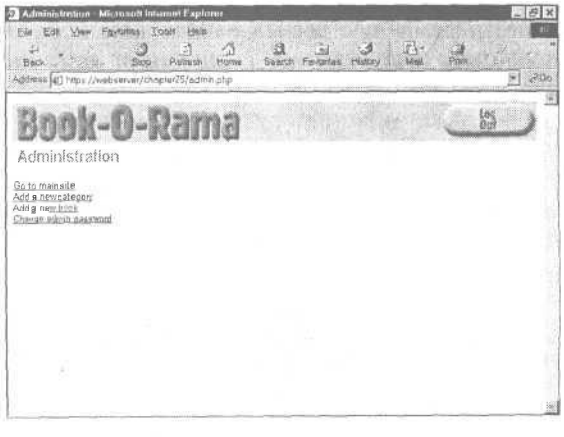


РИСУНОК 25.12
Доступ к функциям администрирования осуществляется через меню.

Код меню администрирования показан в листинге 25.17.

Листинг 25.17 admin.php — этот сценарий выполняет аутентификацию администратора и предоставляет ему доступ к функциям администрирования

```
<?
// подключение файлов функций
require_once("book_sc_fns.php");
session_start();

if ($username && $passwd)
// пользователь только что попытался войти в систему
{
     if (login($username, $passwd))
     {
        // зарегистрировать идентификатор пользователя,
        // если он содержится в базе данных
        $admin_user = $username;
        session_register("admin_user");
     }
     else
     {
        // неуспешная попытка входа в систему
        do_html_header("Problem:");
        echo "You could not be logged in.
              You must be logged in to view this page.<br>";
        do_html_url("login.php", "Login");
        do_html_footer();
        exit;
     }
}
do_html_header("Administration");
if (check_admin_user())
  display admin_menu();
else
  echo "You are not authorized to enter the administration area.";
do_html_footer();
?>
```

Этот код напоминает сценарий из главы 24. На этом этапе администратор может изменять свой пароль либо выходить из системы — данный код идентичен коду из главы 24 и здесь не рассматривается.

Идентификация пользователя-администратора после входа в систему осуществляется через переменную сеанса **$admin_user** и функцию **check_admin_user().** Эта и другие функции, используемые сценариями администрирования, содержатся в библиотеке **admin_fns.php**.

Когда администратору требуется добавить новую категорию или книгу, вызывается сценарий **insert_category_form.php** или **insert_book_form.php**. Каждый сценарий предоставляет администратору форму для заполнения. Формы обрабатываются соответствующими сценариями (**insert_category.php и insert_book.php**), которые проверяют, что форма заполнена, и вводят новую информацию в базу данных. Рассмотрим лишь сценарии, связанные с добавлением книги, поскольку обе пары мало чем отличаются друг от друга.

Вывод сценария **insert_book_form.php** показан на рис. 25.13.

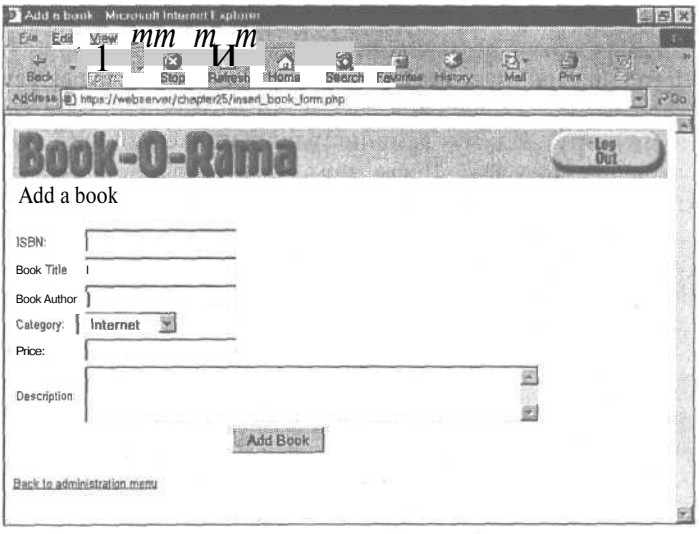


РИСУНОК 25.13
Эта форма позволяет администратору вводить новые книги в интерактивный каталог.

Можно заметить, что поле Category представляет собой HTML-элемент **SELECT.** Опции этого элемента генерируются за счет вызова ранее рассмотренной функции **get_categories()**.

После щелчка на кнопке Add Book (добавить книгу) активизируется сценарий **insert_book.php**. Код сценария показан в листинге 25.18.

Листинг 25.18 insert_book.php - этот сценарий проверяет допустимость данных новой книги и вводит их в базу данных

```
<?
// подключение файлов функций
require_once("book_sc_fns.php");
session_start();

do_html_header("Adding a book");
if (check_admin_user())
{
  if (filled_out($HTTP_POST_VARS))
  {
    if (insert_book($isbn, $title, $author, $catid, $price, $description))
      echo "Book '$title' was added to the database.<br>";
    else
      echo "Book '$title' could not be added to the database.<br>";
  }
  else
    echo "You have not filled out the form.  Please try again.";
  do_html_url("admin.php", "Back to administration menu");
}
else
  echo "You are not authorised to view this page.";

do_html_footer();
?>
```

Легко заметить, что этот сценарий вызывает функцию **insert_book()**. Эта и другие функции, используемые сценариями администрирования, содержатся в библиотеке **admin_fns.php**.

Помимо добавления новых категорий и книг, пользователь с правами администрирования может редактировать и удалять эти элементы. При реализации упомянутых функциональных возможностей повторно использован максимально возможный объем кода. Когда администратор выполняет щелчок на ссылке **Go to main site** (переход к главному сайту) в меню администрирования, выводится индекс категорий (с помощью сценария **index.php**). После этого администратор может выполнять навигацию по сайту так же, как и обычный пользователь. При этом используются те же сценарии.

Однако навигация администратора имеет некоторые отличия. Для него выводятся особые опции, связанные с тем, что зарегистрирована переменная сеанса **$admin_user**. Например, в рассмотренной ранее странице **show_book.php** можно будет заметить отличия в меню опций. Обратимся к рис. 25.14.

На этой странице администратору предоставляются две дополнительные опции: Edit Item (редактировать элемент) и Admin Menu (меню администрирования). Кроме того, в правом верхнем углу вместо ссылки на тележку находится кнопка Log Out (выход из системы).

Это реализуется следующим фрагментом кода (из листинга 25.8):

```
if( check_admin_user() )
  {
    display_button("edit_book_form.php?isbn=$isbn",
                   "edit-item", "Edit Item");
    display_button("admin.php", "admin-menu", "Admin Menu");
    display_button($target, "continue", "Continue");
  }
```

Если вернуться к сценарию **show_cat.php**, можно заметить, что в него также встроены данные опции.

Когда администратор выполняет щелчок на кнопке Edit Item, запускается сценарий **edit_book_form.php**. Его вывод показан на рис. 25.15.

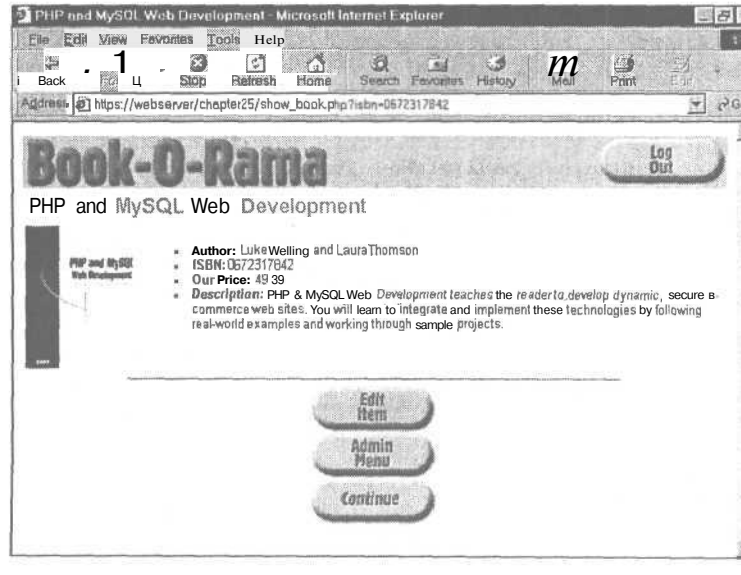


РИСУНОК 25.14
Вывод сценария show_book.php для администратора имеет отличия.

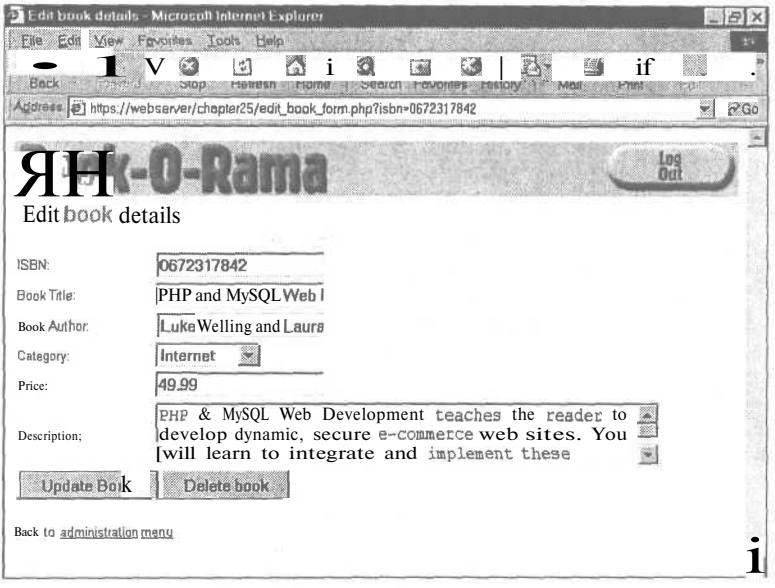


РИСУНОК 25.15 *Сценарий edit_book_form.php позволяет администратору редактировать информацию о книге, либо удалить книгу.*

Фактически, это та же форма, что ранее использовалась для извлечения данных книги. В нее встроена опция передачи и отображения существующих сведений о книге. То же самое выполнено в отношении формы просмотра категорий. Сказанное иллюстрирует листинг 25.19.

Листинг 25.19 Функция display_book_form() библиотеки admin_fns.php — эта форма выполняет двойное назначение вставки и редактирования данных

```
function display_book_form($book = "")
// Отображает форму для книги.
// Она во многом подобна форме для категорий.
// Эта форма может применяться для вставки и
// редактирования информации о книгах.
// Для вставки не нужна передача параметров.
// Переменная $edit получит значение false
// и форма вызовет сценарий insert_book.php.
// Для обновления данных следует передать массив,
// содержащий данные книги. Будет отображена форма с
// прежними данными и запущен сценарий
// update_book.php.
// Кроме того, добавляется кнопка "Delete book".
{
  // если передается существующая книга, переход к
  // режиму редактирования
  $edit = is_array($book);

  // большую часть формы составляет простой HTML-код
  // с небольшими вставками PHP-кода
?>
  <form method=post
        action="<?=$edit?"edit_book.php":"insert_book.php";?>">
  <table border=0>
  <tr>
    <td>ISBN:</td>
    <td><input type=text name=isbn
```

```
        value="<?=$edit?$book["isbn"]:""; ?>"></td>
</tr>
<tr>
  <td>Book Title:</td>
  <td><input type=text name=title
        value="<?=$edit?$book["title"]:""; ?>"></td>
</tr>
<tr>
  <td>Book Author:</td>
  <td><input type=text name=author
        value="<?=$edit?$book["author"]:""; ?>"></td>
 </tr>
 <tr>
    <td>Category:</td>
    <td><select name=catid>
    <?
        // список существующих категорий поступает из базы данных
        $cat_array=get_categories();
        foreach ($cat_array as $thiscat)
        {
              echo "<option value=\"";
              echo $thiscat["catid"];
              echo "\"";
              // если книга существует, она помещается
              // в текущую категорию
              if ($edit ss $thiscat["catid"] == $book["catid"])
                  echo " selected";
              echo ">";
              echo $thiscat["catname"];
              echo "\n";
        }
        ?>
        </select>
      </td>
 </tr>
 <tr>
  <td>Price:</td>
  <tdXinput type=text name=price
            value="<?=$edit?$book["price"]:""; ?>"></td>
 </tr>
 <tr>
   <td>Description:</td>
   <tdxtextarea rows=3 cols=50
        name=description>
        <?=$edit?$book["description"]:""; ?>
        </textarea></td>
 </tr>
 <tr>
    <td <? if (!$edit) echo "colspan=2"; ?> align=center>
       <?
          if ($edit)
           // если обновляется номер ISBN, для поиска книги в базе
           // данных потребуется прежний номер ISBN
           echo "<input type=hidden name=oldisbn
                    value=\"".$book["isbn"]."\">";
       ?>
       <input type=submit
              value="<?=$edit?"Update":"Add"; ?> Book">
       </form></td>
       <?
          if ($edit)
          {
```

```
                echo "<td>";
                echo "<form method=post action=\"delete_book.php\">";
                echo "<input type=hidden name=isbn
                        value=\"". $book["isbn"] "\">".
                echo "<input type=submit
                        value=\"Delete book\">";
                echo "</form></td>";
              }
            ?>
          </td>
        </tr>
    </table>
    </form>
<?
}
```

Если передается массив, содержащий данные книги, форма переводится в режим редактирования, а поля заполняются существующими данными:

```
<input type=text name=price
       value="<?=$edit?$book["price"]:""; ?>">
```

Используется даже другая кнопка отправки формы. Фактически, в форме редактирования их две — одна для обновления книги, а вторая для ее удаления. Они вызывают сценарии **edit_book.php и delete_book.php**, которые соответствующим образом обновляют базу данных.

Версии этих сценариев, связанные с категориями, работают в основном идентично, за исключением одного момента. Когда администратор пытается удалить категорию, этого не произойдет, пока категория содержит книги. (Это проверяется путем запроса базы данных.) В таком случае устраняются все возможности ошибочного удаления, что обсуждалось в главе 7. В данном случае, если удалить категорию, содержащую книги, эти книги станут "зависшими". Навигация к ним будет невозможна, поскольку с данными книгами не связана ни одна категория.

На этом завершается обзор интерфейса администрирования. Для получения дополнительных сведений можно обратиться к коду, который содержится на CD-ROM.

Расширение проекта

Была создана довольно простая система покупательской тележки. Возможны многочисленные расширения и усовершенствования проекта:

- В реальном интерактивном магазине необходимо предусмотреть отслеживание заказов и систему выполнения. В данный момент не существует возможности просматривать размещенные заказы.
- Клиентам нужна возможность проверять обработку своих заказов без необходимости связываться с владельцем магазина. Мы считаем важным, чтобы клиенту не приходилось входить в систему для просмотра заказов. Однако предоставление существующим клиентам способа аутентификации дает им возможность просматривать прежние заказы, а администратору — объединять поведения в профили.
- В настоящее время необходимо пересылать по FTP изображения книг в каталог изображений и присваивать им надлежащие имена. Чтобы упростить этот процесс, можно добавить на страницу вставки книг функцию загрузки файлов.

- Можно добавить входное имя пользователя, персонализацию, рекомендации книг, интерактивные обзоры, сопутствующие программы, проверку запасов и т.п. Возможности практически безграничны.

Использование существующей системы

Чтобы получить программу покупательской тележки с богатыми функциональными возможностями и высоким быстродействием, можно воспользоваться какой-либо существующей системой. Одна из широко известных систем тележек с открытым исходным кодом реализована на языке PHP и называется FishCartSQL. Ее можно найти по адресу:

```
http://www.fishcart.org/
```

Она располагает множеством расширенных функций, таких как отслеживание клиентов, ограниченные по времени продажи, поддержка нескольких языков, обработка кредитных карточек и поддержка нескольких интерактивных магазинов на одном сервере. Текущая версия (на момент написания данной книги) создана на языке PHP 3 и для управления сеансами применяет PHPLib.

Конечно, в существующей системе всегда что-то может показаться лишним, а чего-то может недоставать. Преимущество продукта с открытым исходным кодом состоит в возможности вносить изменения в программу.

Что дальше

В следующей главе рассматривается построение интерактивной системы управления содержимым. Она приспособлена для управления интеллектуальной собственностью в электронном виде, что может оказаться полезным для сайта, основанного на содержимом.

Глава 26

Построение системы управления содержимым

В этой главе рассматривается система управления содержимым для хранения, индексации, а также поиска текста и мультимедийного содержимого.

Системы управления содержимым очень эффективны для Web-сайтов, где содержимое поддерживается не только одним автором, либо сопровождение осуществляет не технический персонал, либо содержимое и графическое оформление разрабатывается различными людьми или отделами.

Мы создадим приложение, помогающее уполномоченным пользователям управлять интеллектуальной собственностью организации в электронном виде.

Будет изучено:

- Представление Web-страниц с помощью набора шаблонов
- Построение механизма поиска, который индексирует документы в соответствии с метаданными

Задача

Предположим, что группа Web-разработчиков компании SuperFastOnlineNews включает хорошего оформителя и несколько авторов-лауреатов. Сайт содержит регулярно обновляемые страницы новостей, спорта и погоды. Главная страница отображает новейшие анонсы по каждой из трех категорий страниц.

В компании SuperFastOnlineNews оформители обеспечивают привлекательный вид содержимого. Это у них получается лучше всего. Авторы, с другой стороны, пишут замечательные статьи, но не умеют рисовать и создавать Web-сайты,

Необходимо позволить каждому сконцентрироваться на своей работе и объединить результаты усилий, чтобы получилась сверхоперативная служба новостей, подобающая названию компании.

Требования к проекту

Необходимо создать систему, которая:

- Увеличивает продуктивность работы, позволяя авторам сконцентрироваться на статьях, а дизайнерам — на оформлении
- Позволяет редактору просматривать статьи и выбирать их для публикации
- Создает единообразное восприятие сайта с помощью шаблонов страниц
- Предоставляет авторам доступ только к предназначенным для них областям сайта
- Позволяет легко изменять оформление любого раздела либо всего сайта
- Предотвращает изменение нового содержимого

Редактирование содержимого

Во-первых, необходимо продумать способ ввода содержимого в систему, а также методы его хранения и редактирования.

Ввод содержимого в систему

Требуется выбрать метод передачи компонентов статей и оформления. Существует три возможности.

FTP

Авторам и дизайнерам можно предоставить FTP-доступ к областям Web-сервера. Это позволит им загружать на сервер файлы со своих локальных компьютеров. Для загружаемых файлов потребуется выработать строгий стандарт именования (для идентификации принадлежности изображений к статьям). Вместо этого можно применить основанную на Web систему, которая будет осуществлять идентификацию независимо от FTP-загрузки.

Использование FTP создает проблему допусков. Необходимая для данного примера гибкость не позволяет использовать FTP для предоставления пользователям возможности загружать файлы.

Метод загрузки файлов

Как упоминалось в главе 16, HTTP-протокол предоставляет метод загрузки файлов при помощи Web-браузера. Язык PHP позволяет решать эту задачу очень просто.

Кроме того, метод загрузки файлов дает возможность хранить текст в базе данных вместо файлов. Для этого выполняется чтение во временный файл и сохранение его содержимого в базе данных, а не копирование в другую область файловой системы. Метод загрузки файлов в этом проекте не используется.

Преимущество базы данных перед файловой системой будет обсуждаться ниже.

Интерактивное редактирование

Пользователи могут создавать и редактировать документы без участия FTP либо другого метода загрузки файлов. Вместо этого авторам можно предоставить в окне большое текстовое поле, в котором они смогут редактировать содержимое статей.

Этот метод прост, но часто эффективен. Web-браузер не предоставляет каких-либо возможностей редактирования текста, кроме функций копирования и вставки, присущих операционной системе. Однако, когда требуется внести лишь небольшое изменение, например, исправить орфографическую ошибку, это можно осуществить очень быстро.

Как и для метода загрузки файлов, данные формы можно записать в файл либо сохранить в базе данных.

Преимущество баз данных перед файлами для хранения содержимого

На начальном этапе необходимо принять важное решение относительно метода хранения содержимого после его загрузки в систему.

Поскольку совместно с текстом хранятся метаданные, мы решили поместить текстовую часть содержимого в базу данных. Хотя MySQL способен хранить мультимедийные данные, принято решение хранить загружаемые изображения в файловой системе. Как обсуждалось во второй части, использование большого двоичного объекта (BLOB) в базе данных MySQL может снизить быстродействие.

В базе данных будут храниться лишь имена файлов изображений. Дескриптор **<IMG SRC>** может прямо ссылаться на каталог графических файлов обычным образом.

Когда объем данных велик, важно оптимизировать их хранение. Подобно тому, как эффективность базы данных зависит от правильной индексации, файловая система существенно выигрывает от хорошо продуманной системы каталогов.

В качестве примера рассмотрим рис. 26.1.



РИСУНОК 26.1
Эта структура каталогов сформирована для загрузки файлов.

В данном случае файловая система содержит каталоги, представляющие первую букву каждого имени файла. Таким образом, 10000 файлов распределены по 26 каталогам. При этом каждый каталог содержит приблизительно 400 файлов. Это существенно ускоряет доступ по сравнению с ситуацией, когда все 10000 файлов хранятся в одном каталоге. Важно отметить, что имена файлов должны генерироваться сценарием обработки загрузки таким образом, чтобы гарантировалась их уникальность.

В примере на рис. 26.1 предполагается, что все имена файлов начинаются с букв нижнего регистра.

Структура документов

В качестве примеров статей будет использоваться краткий текст новостей, включающий один-два абзаца и единственное изображение. Он предназначен для тех, кто спешит. Подобные документы с заголовком, абзацами и иллюстрацией можно считать структурированными.

Чем выше степень структурирования документа, тем проще его разбить на составляющие, хранимые в базе данных. Преимущество такого подхода состоит в возможности единообразного структурированного представления документов.

В качестве примера возьмем статью новостей. Заголовок будет храниться в своем поле отдельно от текста. Изображение по своей природе является отдельным компонентом документа.

Поскольку заголовок является отдельным элементом, для его отображения можно задать стандартный шрифт и стиль, а также легко отделить заголовок от остальной части статьи, сформировав главную страницу заголовков.

Для крупных документов можно применить к отдельным абзацам отношение один ко многим. Другими словами, каждый абзац будет храниться в отдельной строке базы данных и иметь связь с идентификатором главного документа. Этот вид динамической структуры документа позволит представлять страницу содержания для каждого документа и отображать каждый раздел независимо либо отображать документ целиком.

Использование метаданных

Уже решено, что запись для каждой статьи содержит заголовок, текст и изображение. Однако ничто не мешает хранить в той же записи другие данные.

Система автоматически вставляет значения, описывающие авторство статьи и время ее последней модификации. Они могут автоматически отображаться в нижней части статьи и служить подписью и меткой времени, избавляя автора от необходимости добавления этой информации.

Кроме того, иногда полезно добавлять неотображаемые данные, называемые *метаданными*. Хорошим примером служит хранение ключевых слов, которые будут использоваться в качестве индексов поискового механизма.

Поисковый механизм будет извлекать ключевое слово для каждой статьи и на его основе определять соответствие критериям поиска. Это устраняет необходимость сканирования всего текста каждой статьи. Администратор сайта сможет управлять соответствием ключевых слов и фраз определенным документам.

В нашем примере допускается связывать со статьей любое количество ключевых слов, а также присваивать каждому ключевому слову "весовое" значение, указывающее степень его значимости в диапазоне от 1 до 10.

Затем можно разработать алгоритм поискового механизма, который располагает соответствия статьям согласно человеческой шкале значимости. Это заменит сложный алгоритм, который интерпретирует английский текст и принимает решения в зависимости от своего ограниченного понимания, а также управляется фиксированными правилами.

Само собой разумеется, метаданные должны храниться в базе данных. Ничто не мешает использовать HTML-дескриптор **<META>** либо даже применять XML для создания документов. Однако лучше при любой возможности воспользоваться преимуществами базы данных для управления документами.

Форматирование вывода

В нашем примере сайта новостей страницы отображаются в простом, но структурированном формате. Каждая страница содержит ряд статей, сформатированных одинаково. Прежде всего, заголовок выводится крупным шрифтом, внизу слева отображается фотография, а справа — текст статьи. Страница целиком содержится в стандартном шаблоне страниц, что создает последовательность в оформлении сайта.

На рис. 26.2 показана логическая структура страниц, используемая в примере

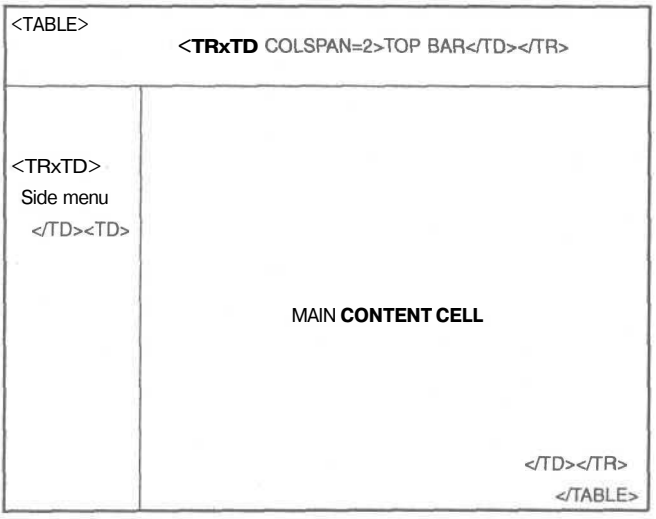


РИСУНОК 26.2
Логическая структура страницы.

Этот вид компоновки крайне популярен — сколько сайтов вы регулярно посещаете, где панель меню расположена слева, а ссылки быстрого доступа вверху? При этом средства навигации остаются неизменными независимо от просматриваемой страницы.

Реализация структуры шаблонов, подобной той, что используется для оформления страниц, очень проста. В HTML-коде можно найти дескриптор **<TD>,** начинающий ячейку главного содержимого. В этом месте HTML-код расщепляется на два файла. Первая половина составляет файл заголовка, а вторая — нижний колонтитул. Когда отображается страница, сначала выводится заголовок, затем текст и, наконец, нижний колонтитул.

Реализация сайта с шаблоном заголовка и нижнего колонтитула позволяет модифицировать оформление сайта, изменяя файлы шаблонов.

PHP предоставляет две опции конфигурации, которые удобны в данной ситуации. Можно определить для каждого каталога директивы **auto_prepend_file** и **auto_append_file**, которые указывают файлы, подключаемые до или после выполнения любого сценария.

Однако существуют некоторые ограничения. Если существуют сценарии, которые не генерируют вывод и отправляют заголовок переадресации, такой как

```
<? header("Location: destination.php"); ?>
```

файлы заголовка и нижнего колонтитула будут по-прежнему отображаться, а переадресация не произойдет, поскольку вывод уже отправлен в Web-браузер. Кроме того, это создает проблемы с cookie-наборами и встроенными функциями управления сеансами PHP 4, поскольку заголовок cookie не будет правильно функционировать после отправки вывода Web-браузеру.

Управление изображениями

Авторы могут дополнять статьи своими фотографиями. Нам необходимо единообразие оформления, но что произойдет, когда один автор загрузит крупное изображение высокого качества, а другой — свернутую в пиктограмму картинку?

Исходя из предположения, что изображения в основном представляют собой фотографии, можно ограничиться лишь форматом JPEG и воспользоваться PHP-функциями для управления графикой. Эта тема подробно рассматривается в главе 19.

Мы написали небольшой сценарий **resize_image.php**, который на лету изменяет размер изображений, в результате чего они могут отображаться с помощью дескриптора **<IMG SRC>**. Сценарий приводится в листинге 26.1.

Листинг 26.1 Сценарий resize_image.php изменяет размеры JPEG-изображения "на лету"

```
<?
if (!$max_width)
   $max_width = 150;
if (!$max_height)
   $max_height = 100;

$size = GetImageSize($image);
$width = $size[0];
$height = $size[1];

$x_ratio = $max width / $width;
$y_ratio = $max_height / $height;

if ( ($width <= $max_width) && ($height <= $max_height) ) {
   $tn_width = $width;
   $tn_height = $height;
}
else if ( ($x_ratio * $height) < $max_height) {
   $tn_height = ceil ($x_ratio * $height);
   $tn_width = $max_width;
}
else {
   $tn_width = ceil($y_ratio * $width);
   $tn_height = $max_height;
}

$src = ImageCreateFromJpeg($image);
$dst = ImageCreate($tn_width,$tn_height);
ImageCopyResized($dst, $src, 0, 0, 0, 0,
     $tn_width,$tn_height,$width,$height);
header("Content-type: image/jpeg");
ImageJpeg($dst, null, -1);
ImageDestroy($src);
ImageDestroy($dst);
?>
```

Этот сценарий принимает три параметра — имя файла изображения, максимальную ширину и высоту в точках. Не стоит полагать, что если указан максимальный размер 200 x 200, изображение будет масштабировано в соответствии с этими значениями. Его масштаб будет уменьшен пропорционально таким образом, чтобы указанные максимальные размеры не превышались. Например, изображение размером 400 x 300 будет уменьшено до размера 200x150. Таким образом достигается сохранение пропорций изображения.

Изменение размера изображения на сервере предпочтительнее, чем простое задание атрибутов **HEIGHT и WIDTH** в дескрипторе **<IMG SRO.** Размер большого изображения с высоким разрешением может составлять несколько мегабайт. Если же картинку уменьшить до приемлемых размеров, ее размер может не превышать 100 Кб. Нет надобности загружать файл большого размера, а затем указывать браузеру изменить размер изображения.

Функции управления изображениями подробно рассматриваются в главе 19. Здесь используется функция **ImageCopyResized()** для масштабирования изображений на лету.

Суть операции изменения размера состоит в вычислении новых параметров ширины и высоты. Определяется соотношение между реальными и максимальными размерами. Параметры **$max_width и $max_height** могут быть переданы в одной строке запроса. Иначе будут использованы стандартные значения, указанные в верхней части листинга.

```
$x_ratio = $max_width / $width;
$y_ratio = $max_height / $height;
```

Если изображение уже меньше указанных максимальных размеров, его ширина и высота остаются неизменными. В противном случае будет использован коэффициент X или Y для равного масштабирования обоих размеров, чтобы изображение уменьшенного размера не оказалось растянутым либо сплющенным:

```
if ( ($width <= $max_width) && ($height <= $max_height) ) {
   $tn_width = $width;
   $tn_height = $height;
}
else if (($x_ratio * $height) < $max_height) {
   $tn_height = ceil($x_ratio * $height);
   $tn_width = $max_width;
}
else {
   $tn_width = ceil($y_ratio * $width);
   $tn_height = $max_height;
}
```

Обзор проекта

В табл. 26.1 содержится перечень файлов данного приложения.

Таблица 26.1 файлы приложения управления содержимым

Имя	*Тип*	*Описание*
create_database.sql	SQL	SQL-запрос для создания базы данных и образцовых данных
include_fns.php	Функции	Набор подключаемых файлов для данного приложения
do_fns.php	Функции	Набор функций для подключения к базе данных содержимого
select_fns.php	Функции	Набор вспомогательных функций для создания списков SELECT
user_auth_fns.php	Функции	Набор функций для аутентификации пользователей
header.php	Шаблон	Отображается в верхней части каждой страницы содержимого
footer.php	Шаблон	Отображается в нижней части каждой страницы содержимого
logo.gif	Изображение	Файл логотипа, отображаемого сценарием header.php
headlines.php	Приложение	Отображает новейший заголовок каждой страницы сайта
page.php	Приложение	Выводит список заголовков и текст для определенной страницы

Имя	Тип	Описание
resize_image.php	Приложение	Изменяет на лету размеры изображений для сценария **headlines.php**
search_form.php	Приложение	Форма ввода ключевых слов для ведения поиска в содержимом сайта
search.php	Приложение	Отображает заголовки содержимого, соответствующего ключевым словам
login.php	Приложение	Выполняет аутентификацию пароля пользователя и вводит его в систему
togout.php	Приложение	Выводит пользователя из системы
stories.php	Приложение	Перечисляет статьи, написанные вошедшим в систему пользователем. Ему предоставляются опции добавления, модификации и удаления статей
story.php	Приложение	Окно информации о статьях для редактирования, либо добавления новых статей
story_submit.php	Приложение	Добавляет новую статью либо передает изменения на основе данных, введенных в окно **story.php**
delete_story.php	Приложение	Обрабатывает запрос на удаление статьи, переданный из сценария **stories.php**
keywords.php	Приложение	Выводит список ключевых слов для статьи с опцией их добавления либо удаления
keyword_add.php	Приложение	Обрабатывает запрос на добавление ключевого слова, переданный из сценария **keywords.php**
keyword_delete.php	Приложение	Обрабатывает запрос на удаление ключевого слова, переданный из сценария **keywords.php**
publish.php	Приложение	Список статей для редактора с указанием, какие из них опубликованы, а также опцией переключения статуса каждой из них
publish_story.php	Приложение	Обрабатывает запрос на публикацию, переданный из сценария **publish.php**
unpublish_story.php	Приложение	Обрабатывает запрос на отмену публикации, переданный из сценария **publish.php**

Создание базы данных

Листинг 26.2 отображает SQL-запросы, используемые для создания базы данных содержимого системы. Этот листинг демонстрирует часть файла **create_database.sql**. Файл на сопровождающем CD-ROM кроме этого содержит запросы для заполнения базы данных примерами пользователей и статей.

Листинг 26.2 Фрагмент create_database.sql — SQL-файла для создания содержимого базы данных

```
drop database if exists content;

create database content;

use content;

drop table if exists writers;
```

```
create table writers (
  username     varchar(16) primary key,
  password     varchar(16) not null,
  full_name    text
);

drop table if exists stories;

create table stories (
   id    int primary key auto_increment,
   writer      varchar(16) not null,        # внешний ключ writers.username
   page        varchar(16) not null,        # внешний ключ pages.code
   headline    text,
   story_text  text,
   picture     text,
   created     int,
   modified    int,
   published   int
);

drop table if exists pages;

create table pages (
   code          varchar(16) primary key,
   description text
);

drop table if exists writer_permissions;

create table writer_permissions (
   writer      varchar(16) not null,        # внешний ключ writers.username
   page        varchar(16) not null         # внешний ключ pages.code
);

drop table if exists keywords;

create table keywords (
   story       int not null,                # внешний ключ stories.id
   keyword     varchar(32) not null,
   weight      int not null
);

grant select, insert, update, delete
on content.*
to content@localhost identified by 'password';
```

Необходимо сохранить краткие сведения о каждом авторе, включая входное имя и пароль, в таблице **writers.** Полные имена авторов будут храниться для вывода после каждой статьи, а также приветствия авторов после входа в систему.

Таблица **pages** содержит заголовки каждой страницы, в которых будут отображаться статьи. Таблица **writer_permissions** реализует отношение многие ко многим с указанием, для каких страниц автор может передавать статьи.

Таблица **stories** содержит отдельные поля для компонентов **headline** (заголовок), **story_text** (текст статьи) и **picture** (изображение), о чем речь шла выше. Поля **created** (создана), **modified** (модифицирована) и **published** (опубликована) имеют целочисленный тип данных и хранят метки времени UNIX соответствующих событий.

Для создания базы данных необходимо выполнить следующую команду:

```
mysql -u root < create_database.sql
```

Реализация

Теперь, когда база данных и функция изменения размеров созданы, приступим к построению основной части системы.

Интерфейсная часть

Начнем с обзора сценария **headlines.php**, показанного в листинге 26.3. Он выводит первую страницу, которая будет отображаться посетителям сайта. Необходимо отобразить заголовки последних статей из каждой страницы.

Листинг 26.3 Сценарий headlines.php отображает новейшие заголовки каждой страницы

```
<?
include ( " include_fns .php");
include ("header.php");

$conn = db_connect ();

$pages_sql = "select * from pages order by code" ;
$pages_result = mysql_query ($pages_sql, $conn) ;

while ($pages = mysql_fetch_array ($pages_result)) {

   $story_sql = "select * from stories
                    where page = ' $pages [code]'
                    and published is not null
                    order by published desc";
   $story_result = mysql_query ($story_sql, $conn) ;
   if (mysql_num_rows($story_result)) {
      $story = mysql_fetch_array ($story_result) ;
      echo "<TABLE BORDER=0 WIDTH=400>" ;
      echo "<TR>";
      echo "<TD ROWSPAN=2 WIDTH=100>" ;
      if ($story[picture])
         echo "<IMG SRC=\"resize_image.php?image=$story[picture]\">" ;
      echo "</TD>";
      echo "<TD>";
      echo "<H3>$pages[description]</H3>" ;
      echo $story[headline];
      echo "</TD>";
      echo "</TR>";
      echo "<TR><TD ALIGN=RIGHT>" ;
      echo "<A HREF=\"page .php?page=$pages[code]\">" ;
      echo "<FONT SIZE=1>Read more $pages[code] . . .</FONT>";
      echo "</A>";
      echo "</TABLE>";
   }
}
include ("footer.php");
?>
```

Этот сценарий, как и все общедоступные сценарии, подключает файл **header.php** в начале и файл **footer.php** в конце. В результате любой вывод, генерируемый сценарием, отображается в ячейке основного содержимого страницы.

Основную нагрузку несут два запроса базы данных. Первый

```
select * from pages order by code
```

извлекает список страниц, содержащихся в базе данных. Затем выполняется тело цикла

```
select * from stories
where page = '$pages[code]'
and published is not null
order by published desc
```

для поиска статей данной страницы в обратном порядке дат публикации.

На рис. 26.3 показан вывод сценария **headline.php** с применением образцовых данных.



РИСУНОК 26.3
Отображение заголовков каждой страницы сайта.

Рядом с каждым заголовком генерируется ссылка в следующей форме:

```
<A HREF="page.php?page=1"><FONT SIZE=l>Read more news...</FONT></A>
```

Рассмотренный выше цикл обеспечивает вывод значения строки запроса **$page** и имени страницы рядом с соответствующим заголовком. В результате щелчка на ссылке выводится страница, генерируемая сценарием **page.php.** Она содержит полный список статей определенной страницы. Код сценария **page.php** приводится в листинге 26.4.

Листинг 26.4 Сценарий page.php отображает на странице все опубликованные статьи

```
<?
include ("include_fns.php");
include ("header.php");

$conn = db_connect();

if (!$page) {
  header("Location: headlines.php");
  exit;
}

$sql = "select * from stories
        where page = '$page'
        and published is not null
        order by published desc";
$result = mysql_query($sql, $conn);
```

```
while ($story = mysql_fetch_array($result))
{
   echo "<H2>".$story[headline]."</H2>";
   if ($story[picture]) {
      $size = getImageSize($story[picture]);
      $width = $size[0];
      $height = $size[1];
      echo "<IMG SRC=\"$story[picture]\" HEIGHT=$height WIDTH=$width
   ALIGN=LEFT>";
   }
   echo $story[story_text];
   $w = get_writer_record($story[writer]);
   echo "<br><FONT SIZE=1>";
   echo $w[full_name].", ";
   echo date("M d, H:i", $story[modified]);
   echo "</FONT>";
}
include ("footer.php");
?>
```

Обратите внимание, что сценарий **page.php** требует наличия значения у переменной **$page**, чтобы правильно сформировать первый запрос. Когда сценарий **page.php** вызывается непосредственно, без строки запроса, первый условный оператор

```
if (!$page) {
   header("Location: headlines.php");
   exit;
}
```

отсылает посетителя обратно к странице заголовков, чтобы отсутствие значения **$page** не привело к ошибке.

Первый запрос

```
select * from stories
where page = '$page'
and published is not null
order by published desc
```

выбирает все статьи, опубликованные на указанной странице, располагая их в обратном хронологическом порядке (самые свежие статьи следуют в начале). Внутри каждого цикла загруженное изображение и текст отображаются на экране, затем следует имя автора и дата последнего изменения.

На рис. 26.4 показана страница **page.php** в действии. В ней выводятся все элементы страницы новостей, которые служат образцовыми данными приложения.

Прикладная часть

Теперь рассмотрим метод добавления статей в систему. Сначала для авторов запускается сценарий **stories.php**. Этот сценарий после аутентификации автора отображает список написанных им статей и дату публикации и либо опции добавления новой статьи, редактирования или удаления существующей, либо установки ключевых слов поиска. Пример показан на рис. 26.5.

Эти окна не форматируются внутри файлов заголовка и нижнего колонтитула, хотя такое и возможно. Поскольку данные сценарии применяют только авторы и редактор, в нашем примере использован лишь необходимый объем форматирования для создания работоспособной системы. Код сценария **stories.php** показан в листинге 26.5.

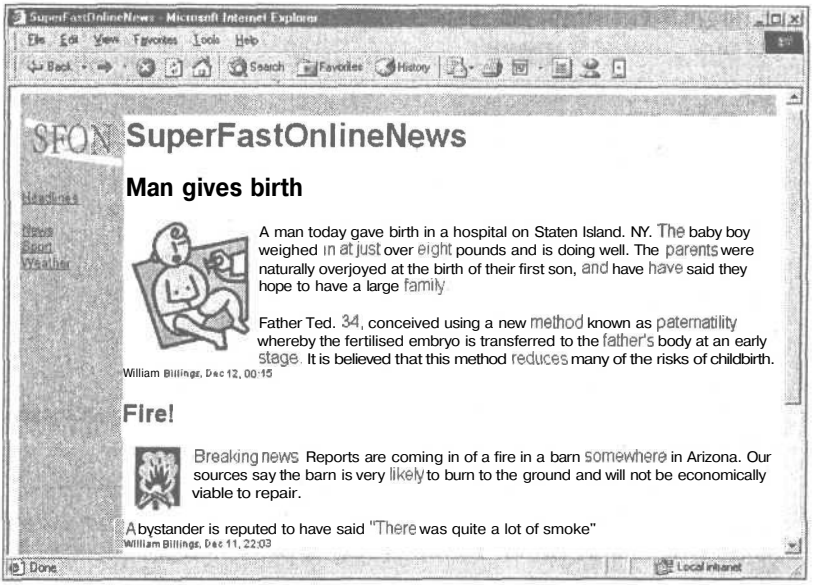


РИСУНОК 26.4
Отображение всех опубликованных статей в странице новостей.

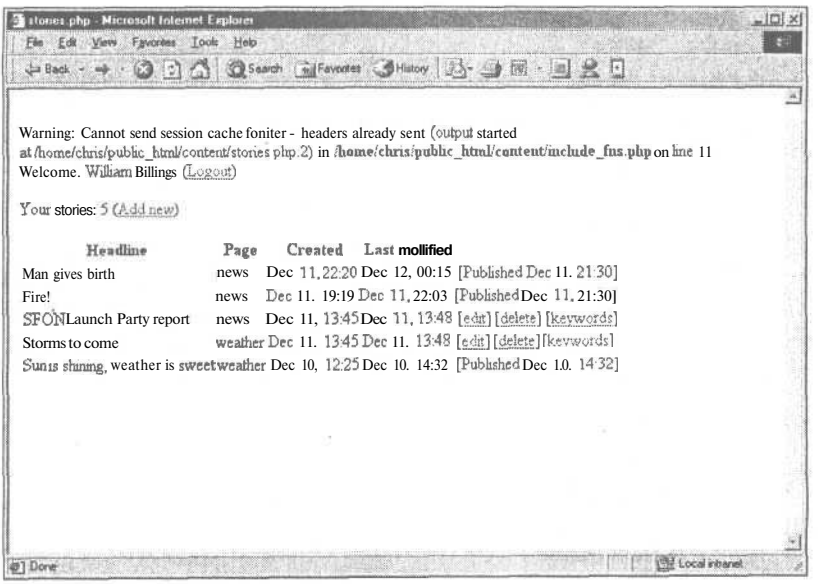


РИСУНОК 26.5
Страница управления статьями для авторов.

Листинг 26.5 Сценарий stories.php служит интерфейсом, который позволяет авторам управлять своими статьями

```
<?
include ("include_fns.php");
session_register("auth_user");

if (!check_auth_user()) {
?>
<FORM ACTION="login.php" METHOD=POST>
<TABLE BORDER=0>
<TR>
   <TD>Username</TD>
   <TD><INPUT SIZE=16 NAME="username"></TD>
</TR>
<TR>
   <TD>Password</TD>
   <TD><INPUT SIZE=16 TYPE="PASSWORD" NAME="password"></TD>
</TR>
</TABLE>
<INPUT TYPE=SUBMIT VALUE="Log in">
</FORM>
<?
}
else {
   $conn = db_connect();

   $w = get_writer_record($auth_user);

   echo "Welcome, ".$w[full_name];
   echo "  (<A HREF=\"logout.php\">Logout</A>)";
   echo "<p>";

   $sql = "select * from stories where writer = '$auth_user' ".
          "order by created desc";
   $result = mysql_query($sql, $conn);

   echo "Your stories: " ;
   echo mysql_num_rows($result);
   echo "  (<A HREF=\"story.php\">Add new</A>)";
   echo "<br><br>";

   if (mysql_num_rows($result)) {
      echo "<TABLE>";
      echo "<TR><TH>Headline</TH><TH>Page</TH>";
      echo "<TH>Created</TH><TH>Last modified</TH></TR>";
      while ($qry = mysql_fetch_array($result)) {
         echo "<TR>";
         echo "<TD>";
         echo $qry[headline];
         echo "</TD>";
         echo "<TD>";
         echo $qry[page];
         echo "</TD>";
         echo "<TD>";
         echo date("M d, H:i", $qry[created]);
         echo "</TD>";
         echo "<TD>";
         echo date("M d, H:i", $qry[modified]);
         echo "</TD>";
         echo "<TD>";
         if ($qry[published])
            echo "[Published ".date("M d, H:i", $qry[published])."]";
         else {
```

```
            echo "[<A HREF=\"story.php?story=".$qry[id]."\">edit</A>] ";
            echo "[<A HREF=\"delete_story.php?story=".$qry[id]."\">delete
                                                        </A>] ";
            echo "[<A HREF=\"keywords.php?story=".$qry[id]."\">keywords</A>]";
        }
        echo "</TD>";
        echo "</TR>";
    }
    echo "</TABLE>";
  }

}
?>
```

На первом этапе проверяется, выполнена ли аутентификация пользователя. Если нет, отображается только форма входной регистрации.

После входа автора в систему переменной сеанса **$auth_user** присваивается значение. Используемая здесь аутентификация не очень надежна. В реальной ситуации необходимо обеспечить, чтобы аутентификация пользователей выполнялась должным образом. Об этом подробно говорится в главе 14.

Форма входной регистрации передает сценарий **login.php**, который сравнивает имя пользователя и пароль с соответствующими значениями базы данных. В случае успешности входной регистрации пользователю возвращается предыдущая страница с помощью значения **$HTTP_REFERER**. Это означает, что сценарий регистрации может вызываться из любой страницы системы.

Затем автор приветствуется по имени и предоставляется возможность выхода из системы. Эта ссылка всегда отображается в верхней части страницы **stories.php**, что позволяет легко выйти из системы в любой момент.

```
$w = get_writer_record($auth_user);

echo "Welcome, ".$w[full_name];
echo "  (<A HREF=\"logout.php\">Logout</A>)";
```

Функция **get_writer_record()** описана в библиотеке **db_fns.php** и возвращает массив полей таблицы автора на основе переданного имени пользователя. Сценарий **logout.php** просто сбрасывает значение переменной **$auth_user.**

Следующий SQL-запрос выбирает все статьи автора, начиная с добавленных в последнее время:

```
select * from stories where writer = '$auth_user'
order by created desc
```

Для каждой записи, связанной со статьей, хранятся метки времени добавления, модификации и публикации. Когда добавляется новая статья, меткам создания и модификации присваивается текущее системное время. Каждое последующее изменение статьи будет вызывать обновление лишь метки модификации.

Вся эта информация выводится в окне статей сначала с помощью кода:

```
echo date("M d, H:i", $qry[created]);
```

затем:

```
echo date("M d, H:i", $qry[modified]);
```

и, наконец:

```
if ($qry[published])
   echo "[Published ".date("M d, H:i", $qry[published])."]";
else {
   echo "[<A HREF=\"story.php?story=".$qry[id]."\">edit</A>] ";
   echo "[<A HREF=\"delete_story.php?story=".$qry[id]."\">delete</A>] ";
   echo "[<A HREF=\"keywords.php?story=".$qry[id]."\">keywords</A>]";
}
```

Последний фрагмент отображает лишь дату публикации, если она имеет смысл. В противном случае выводятся ссылки для редактирования или удаления статьи, а также установки ключевых слов поискового механизма.

Сценарий ввода новой статьи или редактирования существующей содержится в файле **story.php**. На рис. 26.6 показан вывод сценария в режиме редактирования статей образцовой базы данных.

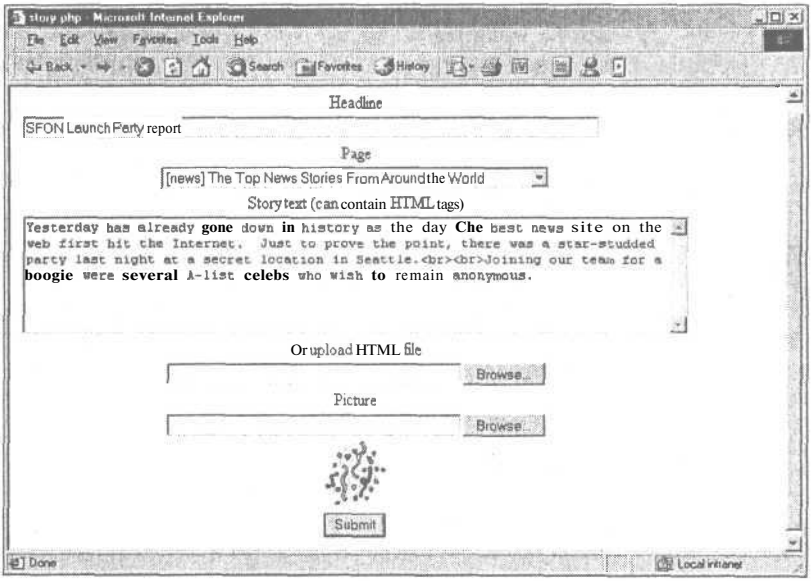


РИСУНОК 26.6 *Редактирование статьи.*

Полный код сценария story.php содержится в листинге 26.6.

Листинг 26.6 Сценарий story.php служит для создания или редактирования статьи

```
<?
include ("include_fns.php");

if (isset($story))
   $s = get_story_record($story);
?>

<FORM ACTION="story_submit.php" METHOD=POST ENCTYPE="multipart/form-data">
<INPUT TYPE=HIDDEN NAME="story" VALUE="<? echo $story;?>">
<INPUT TYPE=HIDDEN NAME="destination" VALUE="<? echo $HTTP_REFERER;?>">
<TABLE>

<TR>
   <TD ALIGN=CENTER>Headline<TD>
</TR>
<TR>
   <TD><INPUT SIZE=80 NAME="headline"
                  VALUE="<? echo $s[headline];?>"></TD>
</TR>
```

```
<TR>
   <TD ALIGN=CENTER>Page<TD>
</TR>
<TR>
   <TD ALIGN=CENTER><? echo query_select ("page",
                "select p.code, p.description
                 from pages p, writer_permissions w
                 where p.code = w.page
                 andw.writer= ' $auth_user ' " ,$s[page]) ;?></TD>
</TR>

<TR>
   <TD ALIGN=CENTER>Story text (can contain HTML tags)</TD>
</TR>
<TR>
   <TDXTEXTAREA COLS=80 ROWS=7 NAME="story_text"
             WRAP=VIRTUAL><? echo $s[story_text] ;?></TEXTAREA>
   </TD>
</TR>

<TR>
   <TD ALIGN=CENTER>Or upload HTML file</TD>
</TR>
<TR>
   <TD ALIGN=CENTER><INPUT TYPE=FILE NAME="html" SIZE=40></TD>
</TR>

<TR>
   <TD ALIGN=CENTER>Picture</TD>
</TR>
<TR>
   <TD ALIGN=CENTER><INPUT TYPE=FILE NAME="picture" SIZE=40></TD>
</TR>

<?
if ($s[picture]) {
   $size   = getImageSize($s[picture]) ;
   $width  = $size[0] ;
   $height = $size[1] ;
?>
<TR>
   <TD ALIGN=CENTER>
      <IMG SRC="<? echo $s[picture];?>"
           WIDTH=<? echo $width;?> HEIGHT=<? echo $height;?>>
   </TD>
</TR>
<? } ?>

<TR>
   <TD ALIGN=CENTER><INPUT TYPE=SUBMIT VALUE="Submit"></TD>
</TR>

</TABLE>
</FORM>
```

Для добавления статей и их редактирования может использоваться один и тот же сценарий. Выполняемое действие зависит от того, установлено ли значение переменной **$story** при вызове сценария.

```
if (isset($story))
   $s = get_story_record($story);
```

Функция **get_story_record()** описана в библиотеке **db_fns.php.** Она возвращает массив всех полей таблицы статей для указанного идентификатора статьи. Если идентифи-

катор не передан, переменная **$story** имеет значение **NULL,** а переменная $s не содержит элементов массива.

```
<INPUT SIZE=80 NAME="headline" VALUE="<? echo $s[headline];?>">
```

Если значение переменной **$story** не установлено, предыдущий фрагмент кода не создаст значения из оператора PHP, поэтому поле ввода заголовка будет пустым. Когда значение переменной **$story** установлено, она содержит текст заголовка для редактируемой статьи.

```
echo query_select("page",
                  "select p.code, p.description
                   from pages p, writer_permissions w
                   where p.code = w.page
                   andw.writer= ' $auth_user ' " ,$s[page]) ;
```

Функция **query_select()** описана в библиотеке select_fns.php. Она возвращает HTML-код вывода списка SELECT на основе данного SQL-запроса. Первый параметр представляет собой атрибут NAME для оператора SELECT. SQL-запрос из второго параметра выбирает два столбца, где первый является составляющей VALUE каждой опции, а второй следует после дескриптора OPTION и представляет собой текст, отображаемый в списке. Третий параметр необязателен. Он добавляет атрибут SELECTED к опции, значение которой совпадает с указанным.

```
<INPUT TYPE=HIDDEN NAME="story" VALUE="<?echo $story;?>">
```

Здесь создается переменная-заполнитель путем установки нового значения для статьи из переданного в переменной **$story**. После передачи формы сценарий story_submit.php проверяет, существует ли значение переменной **$story**, и генерирует в соответствии с этим SQL-оператор UPDATE либо INSERT.

Код сценария **story_submit.php** показан в листинге 26.7.

Листинг 26.7 Сценарий **story_submit.php** служит для вставки или обновления статей в базе данных

```
<?
// story_action.php
// добавление/модификация записи для статьи
include("include_fns.php");
$conn = db_connect();
$time = time();
if ( ($html) && (dirname($html_type) == "text") ) {
   $fp = fopen($html, "r");
   $story_text = addslashes(fread($fp, filesize($html)));
   fclose($fp);
}
if ($story) {    // необходимо обновление
   $sql = "update stories
           set headline = '$headline',
               story_text = '$story_text',
               page = '$page',
               modified = $time
           where id = $story";
>
else {           // необходимо вставить новую статью
```

```
    $sql = "insert into stories
                (headline, story_text, page, writer, created, modified)
             values
                ('$headline', '$story_text', '$page', '$auth_user', $time,
    $time)";
}
$result = mysql_query($sql, $conn);
if (!$result) {
   echo "There was a database error when executing <PRE>$sql</PRE>";
   echo mysql_error();
   exit;
}
if ( ($picture) && ($picture != "none") ) {
   if (!$story)
      $story = mysql_insert_id();
   $type = basename($picture_type);
   switch ($type) {
      case "jpeg":
      case "pjpeg":    $filename = "pictures/$story.jpg";
                       copy ($picture, $filename);
                       $sql = "update stories
                                 set picture = '$filename'
                                 where id = $story";
                       $result = mysql_query($sql, $conn);
                       break;
      default:         echo "Invalid picture format: $picture_type";
   }
}
header("Location: $destination");
?>
```

Ссылка удаления статьи вызывает сценарий **delete_story.php.** Он просто реализует оператор **DELETE** и возвращает автора к вызывающей странице. Код сценария **delete_story.php** показан в листинге 26.8.

Листинг 26.8 Сценарий delete_story.php удаляет статью из базы данных

```
<?
// delete_story.php
include ("include_fns.php");
$conn = db_connect();
$now = time();
$sql = "delete from stories where id = $story";
$result = mysql_query($sql, $conn);
header("Location: $HTTP_REFERER");
?>
```

Поиск статей

В результате щелчка на ссылке ключевых слов в списке статей вызывается новая форма для ввода ключевых слов, связанных со статьей. Количество ключевых слов не ограничено. Каждому из них присваивается значение весомости. Более высокие значения соответствуют более важным ключевым словам.

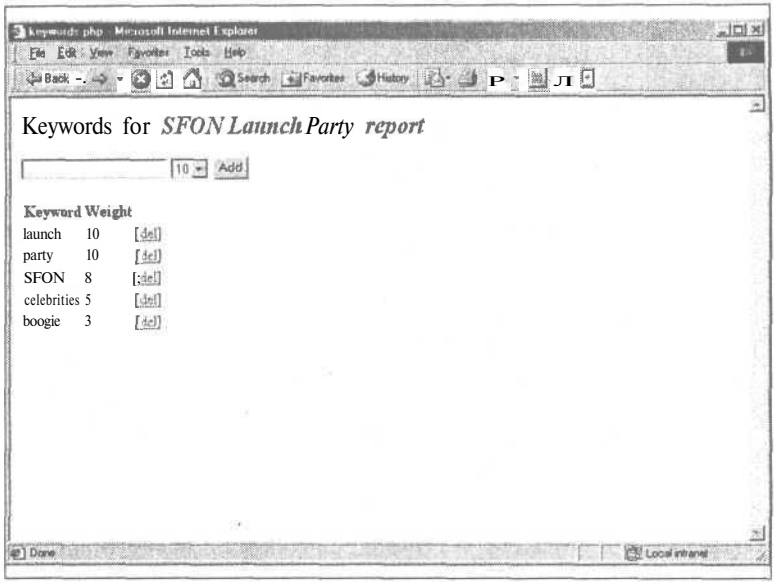


РИСУНОК 26.7
Установка ключевых слов для статьи.

На рис. 26.7 показано окно, используемое для назначения ключевых слов определенной статье.

Сценарий **keywords.php** довольно прост, поэтому мы не будем его подробно рассматривать. Он содержится на сопровождающем CD-ROM.

Этот сценарий запускает сценарии **keyword_add.php** и **keyword_delete.php**. Они также просты и здесь не рассматриваются.

Для ввода новых ключевых слов в базу данных сценарий **keyword_add.php** использует следующий запрос:

```
insert into keywords (story, keyword, weight)
values ($story, '$keyword', $weight)
```

Подобным же образом, сценарий **keyword_delete.php** для удаления ключевого слова использует следующий запрос:

```
delete from keywords where story = $story and keyword = '$keyword'
```

Интерес представляет способ использования показателей весомости для вычисления процентного отношения значимости в процессе поиска.

Форма поиска, генерируемая сценарием **search_form.php**, содержит единственное поле для ключевых слов и передается в сценарий **search.php**, который запрашивает базу данных статей с целью поиска соответствующего содержимого. Код сценария **search.php** показан в листинге 26.9.

Листинг 26.9 Сценарий search.php выполняет поиск статей по ключевым словам и вычисляет процентное отношение значимости соответствия

```
<?
include ("include_fns.php");
include ("header.php");

$conn = db_connect();
```

```
if ($keyword) {
   $k = split(" ", $keyword);
   $num_keywords = count($k);
   for ($i=0; $i<$num_keywords; $i++) {
      if <$i)
         $k_string .= "or k.keyword = '".$k[$i]."' ";
      else
         $k_string .= "k.keyword = '".$k[$i]."' ";
   }
   $and .= "and ($k_string) ";
}
$sql = "select s.id,
                  s.headline,
                  10 * sum(k.weight) / $num_keywords as score
          from stories s, keywords k
          where s.id = k.story
          $and
          group by s.id, s.headline
          order by score desc, s.id desc";

$result = mysql_query($sql, $conn);
echo "<H2>Search results</H2>";
if (mysql_num_rows($result)) {
   echo "<TABLE>";
   while ($qry = mysql_fetch_array($result)) {
      echo "<TR><TD>";
      echo $qry[headline];
      echo "</TD><TD>";
      echo floor($qry[score])."%";
      echo "</TD></TR>";
   }
   echo "</TABLE>";
}
else {
   echo "No matching stories found";
}");
include ("footer.php");
?>
```

Сначала строка ключевого слова, передаваемая в сценарий, разбивается на отдельные слова для поиска. В этом примере не используются какие-либо расширенные методы поиска, такие как возможность применения операторов AND и OR, либо объединение слов в фразу.

```
if ($keyword) {
   $k = split(" ", $keyword);
   $num_keywords = count($k);
   for ($i=0; $i<$num_keywords; $i++) {
      if <$i)
         $k_string .= "or k.keyword = '".$k[$i]."' ";
      else
         $k_string .= "k.keyword = '".$k[$i]."' ";
   }
   $and .= "and ($k_string) ";
}
```

В этом коде используется PHP-функция **split()** для создания массива, содержащего каждое слово строки **$keyword**, разделенное пробелами. Если указано только одно слово, возвращается массив с единственным элементом, и последующий цикл ВЫПОЛ-

няется один раз. В результате условие, хранимое в переменной **$and**, будет иметь примерно следующий вид:

```
and (k.keyword = 'keyword1' or k.keyword = 'keyword2' or
   k.keyword = 'keyword3')
```

Ниже приводится запрос поиска, основанный на рассмотренном ранее коде:

```
select s.id,
       s.headline,
       10 * sum(k.weight) / $num_keywords as score
from stories s, keywords k
where s.id = k.story
and (k.keyword = 'keyword1'
   or k.keyword = 'keyword2'
   or k.keyword = 'keyword3')
group by s.id, s.headline
order by score desc, s.id desc
```

Общий показатель значимости определяется как сумма весовых значений всех соответствующих ключевых слов, деленная на количество искомых ключевых слов и умноженная на десять. Эта формула пригодна для поисков, когда все введенные ключевые слова соответствуют ключевым словам базы данных.

Поскольку показатели весомости находятся в диапазоне от 1 до 10, максимальное значение общего показателя значимости составляет 100. Эта величина для поиска по трем ключевым словам достигается, когда все три слова найдены и каждое из них имеет весовое значение 10.

Окно *редактора*

Осталась нерассмотренной одна составляющая проекта, отвечающая за публикацию статьи после ее написания. Это осуществляет сценарий **publish.php**, показанный в листинге 26.10.

Листинг 26.10 Сценарий publish.php выводит список всех документов, чтобы редактор мог выбрать из них те, которые будут отображаться на странице опубликованных статей

```
<?
include ("include_fns.php");

$conn = db_connect();

$sql = "select * from stories order by modified desc";
$result = mysql_query($sql, $conn);

echo "<H2>Editor admin</H2>";
echo "<TABLE>";
echo "<TR><TH>Headline</TH><TH>Last modified</TH></TR>";
while ($story = mysql_fetch_array($result)) {
  echo "<TR><TD>";
  echo $story[headline];
  echo "</TD><TD>";
  echo date("M d, H:i", $story[modified]);
  echo "</TD><TD>";
  if ($story[published]) {
    echo "[<A HREF=\"unpublish_story.php?story=$story[id]\">unpublish</A>]
";
  }
```

```
    else {
      echo " [<A HREF=\"publish_story.php?story=$story[id]\">publish</A>] ";
      echo "[<A HREF=\"delete_story.php?story=$story[id]\">delete</A>] ";
    }
    echo "[<A HREF=\"story.php?story=$story[id]\">edit</A>] ";

    echo "</TD></TR>";
}
echo "</TABLE>";
?>
```

Данный сценарий должен быть доступным только для лиц, уполномоченных публиковать статьи на сайте. В нашем приложении это редактор сайта. Однако в целях простоты здесь не реализовано управление доступом. В реальной ситуации сценарий должен быть защищен.

Он очень похож на сценарий **stories.php** с той разницей, что для редактора отображаются статьи всех авторов, а не только его собственные. Оператор **if** обеспечивает предоставление для каждой статьи необходимых опций. Публикация статей может отменяться, а неопубликованные статьи могут публиковаться либо удаляться.

Связанные с этим оператором три ссылки вызывают, соответственно, сценарии unpublish story.php, **publish_story.php** и **delete_story.php**.

Сценарий **publish_story.php** использует следующий SQL-запрос:

```
update stories set published = $now
          where id = $story
```

Он помечает статью как опубликованную и делает ее общедоступной для просмотра.

Подобно этому, сценарий **unpublish_story.php** использует следующий запрос, чтобы пометить статью как неопубликованную и прекратить ее отображение для всеобщего просмотра:

```
update stories set published = null
          where id = $story
```

Ссылка редактирования отображается независимо от факта публикации статьи, поэтому редактор всегда может внести в нее изменения. В этом состоит отличие от уровня доступа автора, который может изменять статью лишь до ее публикации.

Расширение проекта

Существует несколько путей расширения проекта, чтобы сделать систему управления содержимым более совершенной:

- Позволить группам пользователей совместно работать над статьями.
- Реализовать более гибкую компоновку страницы, чтобы редакторы могли помещать в нее текст и изображения.
- Построить библиотеку изображений, чтобы часто используемые картинки не дублировались. Кроме того, с изображениями, как и текстом, можно связать ключевые слова для целей поиска.
- Добавить функции проверки орфографии при редактировании содержимого. Для реализации проверки можно воспользоваться, например, библиотекой aspell.

Что дальше

В следующем проекте создается основанный на Web интерфейс, который позволит просматривать и отправлять электронную почту в среде Web по протоколу IMAP.

Глава 27

Построение почтовой службы, основанной на Web

В наше время сайты все чаще предоставляют пользователям услуги электронной почты, основанной на Web. В этой главе рассматривается реализация Web-интерфейса с существующим почтовым сервером при помощи PHP-библиотеки IMAP. Его можно использовать для просмотра содержимого собственного почтового ящика с помощью Web-страницы. Кроме того, проект можно расширить для создания многопользовательской системы основанной на Web почты, подобной Hotmail.

В этом проекте будет создан почтовый клиент Warm Mail, который предоставит пользователям следующие возможности:

- Подключаться к почтовым серверам POP3 или IMAP с использованием своих учетных записей
- Читать почтовые сообщения
- Отправлять почту
- Отвечать на почтовые сообщения
- Переадресовывать почтовые сообщения
- Удалять сообщения из своего почтового ящика

Задача

Чтобы пользователь мог читать почтовые сообщения, необходимо найти способ подключения к его почтовому серверу. Обычно он содержится на отдельной от Web-сервера машине.

Требуется наладить взаимодействие с почтовым ящиком, чтобы просматривать список принятых сообщений и индивидуально обрабатывать их.

Для чтения сообщений из почтовых ящиков пользователей почтовые серверы поддерживают два основных протокола: POP3 и IMAP.

Если возможно, в проекте следует реализовать поддержку обоих протоколов. Сокращение POP3 означает Post Office Protocol (почтовый протокол) версии 3, а ШАР — Internet Message Access Protocol (протокол доступа к сообщениям Internet).

Основное отличие между ними состоит в том, что POP3 предназначен и обычно используется пользователями, которые кратковременно подключаются к сети для загрузки и удаления сообщений на сервере. Протокол IMAP предназначен для использования в режиме постоянного подключения к сети для постоянного взаимодействия с почтовой службой, поддерживаемой на удаленном сервере. Протокол IMAP обладает некоторыми расширенными возможностями, которые в данном проекте не используются.

О различиях между протоколами можно узнать из документов RFC (RFC 1939 для POP3 и RFC 2060 для IMAP 4.1). Хорошую статью со сравнительным анализом протоколов можно найти по адресу:

```
http://www.imap.org/papers/imap.vs.pop.brief.html
```

Ни один из этих протоколов не предназначен для отправки почты — для этого необходимо использовать *SMTP* (Simple Mail Transfer Protocol — упрощенный протокол передачи электронной почты), который мы ранее применяли в PHP-коде при помощи функции **mail()**. Этот протокол описан в документе RFC 821.

Компоненты решения

PHP реализует широкую поддержку протоколов IMAP и POP3, но она предоставляется через библиотеку функций IMAP. Чтобы использовать код этой главы, необходимо установить библиотеку IMAP. Узнать, установлена ли она уже в систему, можно за счет просмотра вывода функции **phpinfo()**.

Если библиотека не установлена, это расширение следует загрузить. Последнюю версию можно загрузить через FTP из ресурса:

```
ftp://ftp.cac.washington.edu/imap/c-client.tar.Z
```

В системе UNIX следует загрузить ресурс и скомпилировать его в рамках операционной системы. После этого потребуется скопировать файлы **rfc822.h**, **mail.h** и **linkage.h** в каталог **/usr/local/include**, либо другой подключаемый каталог, выполнить сценарий конфигурирования, добавив директиву **--with-imap** к остальным параметрам, и повторно скомпилировать PHP.

Существует документация с описанием компилирования версии для Windows, но это сложнее, чем компилирование для UNIX. Для платформы Windows существует другая альтернатива. Можно загрузить версию PHP, предварительно скомпилированную с различными расширениями, включая IMAP-расширение, из сайта:

```
http://www.php4win.de
```

Важно отметить, что IMAP-функции, несмотря на название, одинаково хорошо взаимодействуют с протоколами POP3 и NNTP (Networks News Transfer Protocol — протоколом передачи сетевых новостей). Мы будем применять их для протоколов IMAP и POP3, но приложение Warm Mail можно легко расширить для использования NNTP.

Эта библиотека содержит очень много функций, но для данного приложения потребуются лишь несколько из них. Они будут рассмотрены в процессе использования. Если преследуются иные цели либо необходимо включить в приложение дополнительные функциональные возможности, можно обратиться к документации.

Можно построить довольно эффективное почтовое приложение, применив лишь часть встроенных функций. Это означает, что достаточно проработать лишь часть документации. Ниже перечислены функции IMAP, используемые в данной главе:

- `imap_open()`
- `imap_close()`
- `imap_headers()`
- `imap_header()`
- `imap_fetchheader()`
- `imap_body()`
- `imap_delete()`
- `imap_expunge()`

Чтобы пользователь мог читать почтовые сообщения, необходимо получить информацию о его сервере и учетной записи. Чтобы пользователю не приходилось вводить эти сведения каждый раз, для их хранения будет создана база данных имен пользователей и паролей.

Часто пользователи имеют более одной учетной записи электронной почты (например, одну для дома и одну для работы). Необходимо предоставить им возможность подключения с использованием любой учетной записи. Поэтому следует обеспечить хранение в базе данных нескольких наборов сведений об учетных записях для каждого пользователя.

Пользователи должны иметь возможность читать сообщения, отвечать на них, переадресовывать и удалять существующие сообщения, а также отправлять новые. Все функции чтения можно возложить на протоколы IMAP и POP3, а операции отправки — на протокол SMTP и функцию **mail()**.

Рассмотрим, как объединить все составляющие проекта.

Обзор проекта

Общая схема основанной на Web системы не особенно отличается от других почтовых клиентов. Блок-схема системы и ее модулей показана на рис. 27.1.

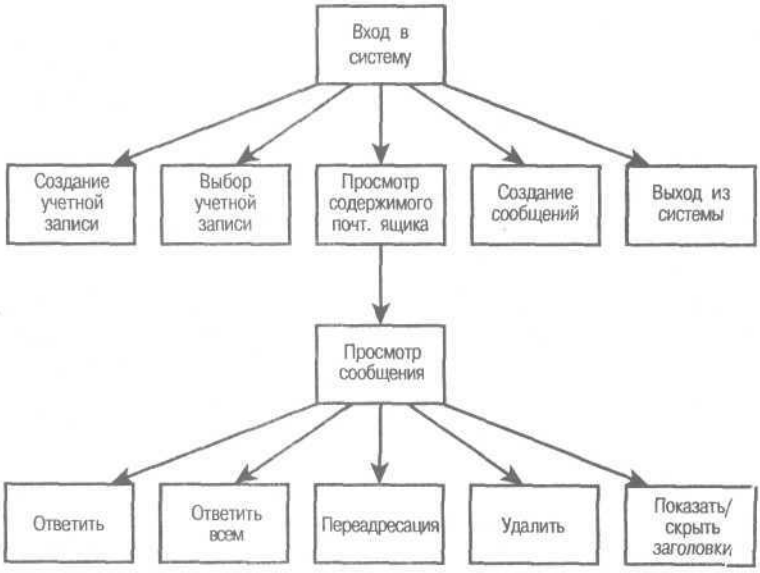


РИСУНОК 27.1 *Интерфейс приложения Warm Mail предоставляет пользователю функциональные возможности уровня почтового ящика и уровня сообщений.*

Как видно из диаграммы, сначала пользователь должен войти в систему, а затем ему предоставляется список опций. Он может создать новую учетную запись либо выбрать для использования одну из существующих. Кроме того, пользователь может просматривать входящие сообщения, отвечать на них, переадресовывать, удалять, а также отправлять новые сообщения.

Помимо этого пользователю предоставляется опция просмотра заголовков с дополнительной информацией для определенного сообщения. Просмотр всех заголовков позволяет многое узнать о сообщении. Можно определить, с какого компьютера почта отправлена — удобное средство отслеживания нежелательных сообщений ("спэмминга"). Можно узнать, какие машины пересылали сообщение и время его поступления на каждый хост — это удобно для предъявления претензий по поводу задержки сообщений. Кроме того, если приложение добавляет в заголовки выборочную информацию, можно определить, каким почтовым клиентом пользовался отправитель.

В этом проекте применена несколько отличная архитектура приложения. Вместо набора сценариев, по одному для каждого модуля, использован сценарий несколько большего размера, **index.php,** который играет роль цикла событий программы с графическим интерфейсом пользователя. Каждое действие в сайте, осуществляемое нажатием кнопки, возвращается в сценарий **index.php,** но со своим параметром. В зависимости от параметра вызываются различные функции, чтобы вывести необходимое содержимое пользователю. Как обычно, функции содержатся в библиотеках.

Эта архитектура пригодна для небольших приложений, подобных данному. Это могут быть управляемые событиями приложения, когда функции запускаются действиями пользователя. Использование единственного обработчика событий не очень удобно для крупных приложений либо проектов, разрабатываемых группой программистов.

Перечень файлов проекта Warm Mail показан в табл. 27.1.

Таблица 27.1 файлы **приложения Warm Mail**

Имя	***Тип***	***Описание***
index.php	Приложение	Основной сценарий, полностью реализующий приложение
include_fns.php	Функции	Набор подключаемых файлов приложения
data_valid_fns.php	Функции	Набор функций проверки допустимости вводимых данных
do_fns.php	Функции	Набор функций подключения к базе данных **mail**
mail_fns.php	Функции	Набор связанных с электронной почтой функций для открытия почтовых ящиков, чтения сообщений и т. п.
output_fns.php	Функции	Набор функций для вывода HTML-содержимого
user_auth_fns.php	Функции	Набор функций аутентификации пользователей
create_database.sql	SQL	SQL-код создания базы данных **book_sc** и регистрации пользователей

Перейдем к обзору приложения.

Создание базы данных

База данных для приложения Warm Mail довольно проста, поскольку в ней не будут храниться сообщения электронной почты.

Необходимо хранить сведения о пользователях системы. Для каждого пользователя должны существовать следующие поля:

- **username** — выбранное пользователем имя для приложения Warm Mail.
- **password** — выбранный пользователем пароль.
- **address** — указанный пользователем адрес электронной почты, который будет отображаться в поле From (отправитель) отправляемых из системы сообщений.
- **displayname** — "читабельное" имя, выбранное пользователем для отображения в отправляемых сообщениях.

Кроме того, требуется хранить следующие данные каждой учетной записи:

- **username** — имя пользователя Warm Mail, которому принадлежит учетная запись.
- **server** — машина, на которой размещена учетная запись. Например, **localhost** или **mail.tangledweb.com.au**.
- **port** — порт, к которому выполняется подключение с использованием данной учетной записи. Обычно для POP3-серверов применяется номер порта 110, а для IMAP-серверов — 143.
- **type** — протокол, используемый для подключения к данному серверу — **'POP3'** или **'IMAP'**.
- **remoteuser** — имя пользователя для подключения к почтовому серверу.
- **accountid** — уникальный ключ для идентификации учетных записей.

Для создания базы данных можно применить SQL-запрос, показанный в листинге 27.1.

Листинг 27.1 create_database.sql — SQL-запрос для создания почтовой базы данных mail

```
create database mail;

use mail;

create table users
(
  username char(16) not null primary key,
  password char(16) not null,
  address char(100) not null,
  displayname char(100) not null
);

create table accounts
(
  username char(16) not null,
  server char(100) not null,
  port int not null,
  type char(4) not null,
  remoteuser char(50) not null,
  remotepassword char(50) not null,
  accountid int unsigned not null auto_increment primary key
);

grant select, insert, update, delete
on mail.*
to mail@localhost identified by 'password';
```

Помните, что этот SQL-запрос можно выполнить, напечатав следующую строку:

```
mysql -u root -p < create_database.sql
```

Потребуется ввести свой пароль привилегированного пользователя (root). Перед выполнением запроса необходимо изменить пароль пользователя почтовой службы в файлах **create_database.sql и db_fns.php.**

В содержимое CD-ROM дополнительно включен файл **populate.sql**. В этом приложении не будет реализован процесс регистрации пользователей либо администрирования. Если планируется крупномасштабное использование приложения, это можно выполнить самостоятельно. Однако для личного применения достаточно ввести в базу данных свои данные. Сценарий **populate.sql** предоставляет для этого форму. Чтобы стать пользователем, можно ввести в нее свои данные и передать их.

Архитектура сценария

Как уже упоминалось, в этом приложении всем управляет один сценарий — **index.php** Его код показан в листинге 27.2. Сценарий довольно длинный, но мы последовательно рассмотрим его разделы.

Листинг 27.2 index.php - основа системы Warm Mail

```
<?
// Этот файл служит основой приложения Warm Mail.
// Он функционирует, главным образом, как автомат и
// генерирует для пользователей вывод в зависимости от
// выполняемых ими действий

//*************************************************************************
// // Этап 1: предварительная обработка
// Выполнение всех необходимых операций перед
// отправкой заголовка страницы и выбор информации,
// отображаемой в заголовках страницы
//*************************************************************************
  include ('include_fns.php');
  session_start();

  $buttons = array();

  // присоединить к этой строке, если выполнены какие-либо
  // операции перед выводом заголовка
  $status = '';

  // прежде всего, следует обработать запросы входа или
  // выхода из системы
  if($username||$password)
  {
    if(login($username, $passwd))
    {
      $status .= "<p>Logged in successfully.<br><br><br><br><br><br>";
      $auth_user = $username;
      session_register("auth_user");
    }
    else
    {
      $status .= "<p>Sorry, we could not log you in with that
                  username and password.<br><br><br><br><br><br>";
    }
  }
  if($action == 'log-out')
  {
    session_destroy();
    unset($action);
    unset($auth_user);
  }
```

```
// обработка выбора, удаления или сохранения учетной
// записи перед отображением заголовка
switch ( $action )
{
  case 'delete-account' :
  {
    delete_account($auth_user, $account);
    break;
  }
  case 'store-settings' :
  {
    store_account_settings($auth_user, $HTTP_POST_VARS);
    break;
  }
  case 'select-account' :
  {
    // если выбрана допустимая учетная запись,
    // сохранить ее в переменной сеанса
    if($account&&account_exists($auth_user, $account))
    {
      $selected_account = $account;
      session_register('selected_account');
    }
  }
}
// создание кнопок, выводимых в панели инструментов
$buttons[0] = 'view-mailbox';
$buttons[1] = 'new-message';
$buttons[2] = 'account-setup';
//кнопка log-out необходима только после успешного входа в систему
if (check auth_user())
{
  $buttons[4] = 'log-out';
}
//*****************************************************************************
// Этап 2 : заголовки
// Отправка HTML-заголовков и строки меню,
// соответствующих текущему действию
//*****************************************************************************
  if($action)
  {
    // отображение заголовка с именем приложения и
    // описание страницы либо действия
    do_html_header($auth_user, "Warm Mail - ".
                   format_action($action), $selected_account);
  }
  else
  {
    // отображение заголовка с одним лишь названием приложения
    do_html_header($auth_user, "Harm Mail", $selected_account);
  }
  display_toolbar($buttons);
//*****************************************************************************
// // Этап 3: тело страницы
// В зависимости от действия, отображать основное содержимое
//*****************************************************************************
  // отображать текст, сгенерированный функциями,
  // вызванными до отображения заголовка
  echo $status;
```

```
if(!check_auth_user())
{
   echo "<P>You need to log in";
   if($action&&$action!='log-out')
      echo " to go to ".format_action($action);
   echo ".<br><br>";
   display_login_form($action);
}
else
{
   switch ( $action )
   {
      // если выбрана опция создания новой учетной записи
      // либо только добавлена или удалена учетная запись,
      // отобразить страницу создания учетных записей
      case 'store-settings' :
      case 'account-setup' :
      case 'delete-account' :
      {
         display_account_setup($auth_user);
         break;
      }
      case 'send-message' :
      {
         if (send_message($to, $cc, $subject, $message) )
            echo "<p>Message sent .<br><br><br><br><br><br>";
         else
            echo "<p>Could not send message. <br><br><br><br><br><br>";
         break;
      }
      case 'delete' :
      {
          delete_message ($auth_user, $selected_account, $messageid) ;
          // оператор 'break' опущен умышленно — будет
          // выполнен переход к следующему оператору case
      >
      case 'select-account' :
      case 'view-mailbox' :
      {
         // если почтовый ящик выбран, отобразить его содержимое
         display_list ($auth_user, $selected_account) ;
         break;
      }
      case 'show-headers' :
      case 'hide-headers' :
      case 'view-message' :
      {
         // если только что выбрано сообщение из списка, либо
         // оно просматривается и выбрана опция сокрытия или
         // вывода заголовков, загрузить сообщение
         $fullheaders = ($action=='show-headers ');
         display_message ($auth_user, $selected_account, $messageid,
                          $fullheaders);
         break;
      (
      case 'reply-all' :
      {
        // присвоить переменной $cc значение из строки адреса
       // отправки копии (carbon copy)
         if(!$imap)
            $imap = open_mailbox($auth_user, $selected_account);
         if($imap)
```

```
        {
            $header = imap_header($imap, $messageid);
            if ($header->reply_toaddress)
               $to = $header->reply_toaddress;
            else
               $to = $header->fromaddress;
            $cc = $header->ccaddress;
            $subject = 'Re: ' . $header->subject;
            $body = add_quoting(stripslashes (imap_body($imap,
                                                $messageid)));
            imap_close($imap);
            display new_message_form($auth_user, $to, $cc, $subject, $body);
         }
         break;
      }
      case ' reply ' :
      {
         // установка адреса из поля reply-to (ответить), либо from
         //  (от) текущего сообщения
         if (!$imap)
            $imap = open_mailbox($auth user, $selected_account);
         if ($imap)
         {
            $header = imap_header($imap, $messageid);
            if($header->reply_toaddress)
               $to = $header->reply_toaddress;
            else
               $to = $header->fromaddress;
            $subject = 'Re: '.$header->subject;
            $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
            imap_close($imap);
            display_new_message form($auth_user, $to, $cc, $subject, $body);
         }
         break;
      }
      case 'forward' :
      {
         // вывод текущего сообщения в кавычках в окне текущего
         // сообщения
         if(!$imap)
            $imap = open_mailbox($auth_user, $selected_account);
         if($imap)
         {
            $header = imap_header($imap, $messageid);
            $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
            $subject = 'Fwd: ' . $header->subject;
            imap_close($imap);
            display new_message_form($auth_user, $to, $cc, $subject, $body);
         }
         break;
      }
      case 'new-message' :
      {
         display_new_message_form($auth_user, $to, $cc, $subject, $body);
         break;
      }
   }
}
```

```
//*************************************************************************
// Этап 4: нижний колонтитул
  do_html_footer();
//*************************************************************************
?>
```

В этом сценарии использован принцип обработки событий. Он реализует логическую последовательность выполнения функций для каждого события. В данном случае события запускаются пользователем, который выполняет щелчки на кнопках страницы. Большинство кнопок генерируется функцией **display_button(),** а функция **display_form_button()** используется для кнопок отправки формы (submit). Обе функции содержатся в библиотеке **output_fns.php.** Все они подчинены URL-адресам формы:

```
index.php?action=log-out
```

Когда вызывается сценарий **index.php**, значение переменной **$action** определяет, обработчик какого события требуется запустить.

Ниже перечислены четыре основных раздела сценария:

1. Выполняются некоторые операции, которые должны предшествовать отправке заголовка страницы браузеру. Сюда относятся запуск сеанса, выполнение предварительной обработки для выбранного пользователем действия и определение вида заголовков.
2. Обрабатываются и отправляются необходимые заголовки и строка меню для выбранного пользователем действия.
3. Выбирается фрагмент сценария для выполнения в зависимости от предпринятого действия. Различные действия запускают различные функции.
4. Отправляются нижние колонтитулы страницы.

Легко заметить, что эти четыре раздела отмечены комментариями.

Для полного понимания сценария последовательно рассмотрим все действия, реализуемые сайтом.

Вход и выход из системы

Когда пользователь загружает страницу **index.php**, генерируется вывод, показанный на стр. 27.2.

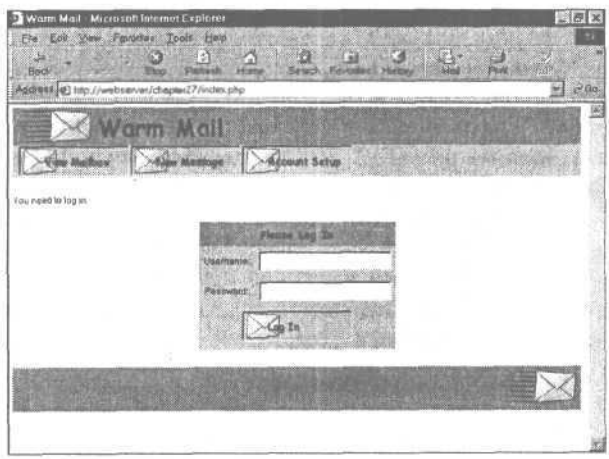


РИСУНОК 27.2
Окно входа в систему для приложения Warm Mail запрашивает имя пользователя и пароль.

Так реализуется стандартное поведение приложения. Когда ни одно действие не выбрано (значение переменной **$action** не установлено) и не предоставлены данные входа в систему, необходимо выполнить следующие фрагменты кода.

На этапе предварительных операций выполняется следующий код:

```
include ('include_fns.php');
session_start();
```

Эти строки запускают сеанс, который будет использоваться для отслеживания переменных сеанса **$auth_user** и **$selected_account()**. К упомянутым переменным мы вернемся позже.

Чтобы упростить настройку интерфейса пользователя, отображаемые в панели инструментов кнопки управляются массивом. Сначала объявляется пустой массив

```
$buttons = array();
```

а затем создаются кнопки, отображаемые в странице:

```
$buttons[0] = 'view-mailbox';
$buttons[1] = 'new-message';
$buttons[2] = 'account-setup';
```

На втором этапе создается простой заголовок:

```
do_html_header($auth_user, "Warm Mail", $selected_account);
...
display_toolbar($buttons);
```

Этот код печатает строку заголовка и панель инструментов с кнопками, как показано на рис. 27.2. Задействуемые функции содержатся в библиотеке **output_fns.php**. Поскольку результат действия функций хорошо виден на иллюстрации, мы их рассматривать не будем.

Переходим к основной части кода:

```
if(!check_auth_user())
{
  echo "<P>You need to log in";
  if($action&&$action!='log-out')
    echo " to go to ".format_action($action);
  echo ".<br><br>";
  display_login_form($action);
}
```

Функция **check_auth_user()** входит в библиотеку **user_auth_fns.php**. Очень похожий код использовался в предыдущих проектах для проверки, вошел ли пользователь в систему. Если нет, как в данном случае, отображается форма входной регистрации, показанная на рис. 27.2. Эта форма генерируется функцией **display_login_form()** библиотеки **output_fns.php**.

Если пользователь правильно заполняет форму и нажимает кнопку Log In (войти в систему), отображается вывод, показанный на рис. 27.3.

При выполнении этого сценария активизируются различные разделы кода. Форма входной регистрации содержит два поля, **$username** и **$password**. Если они оба заполнены, активизируется следующий фрагмент кода предварительной обработки:

```
if($username||$password)
{
  if(login($username, $passwd))
```

```
    {
        $status .= "<p>Logged in successfully.<br><br><br><br><br><br>";
        $auth_user = $username;
        session_register("auth_user");
    }
    else
    {
        $status .= "<p>Sorry, we could not log you in with that
                    username and password.<br><br><br><br><br><br>";
    }
}
```

Можно видеть, что вызывается функция **login()**, подобная той, что использовалась в главах 24 и 25. В случае успешного выполнения имя пользователя регистрируется в переменной сеанса **$auth_user.**

Помимо кнопок, отображаемых до входа в систему, добавлена кнопка, позволяющая пользователю снова выйти из системы:

```
if(check_auth_user())
{
    $buttons[4] = 'log-out';
}
```

Кнопка *Log* Out показана на рис. 27.3.

На втором этапе снова отображаются заголовок и кнопки. В теле сценария выводится сообщение состояния, созданное ранее:

```
echo $status;
```

После этого остается вывести нижний колонтитул и ожидать последующих действий пользователя.

Создание учетных записей

Когда пользователь впервые начинает работать с системой Warm Mail, ему необходимо создать учетные записи. После щелчка на кнопке Account Setup (настройка учетной записи), переменной **$action** устанавливается значение **account-setup** и вызывается сценарий **index.php**. Отображаемый после этого вывод показан на рис. 27.4.

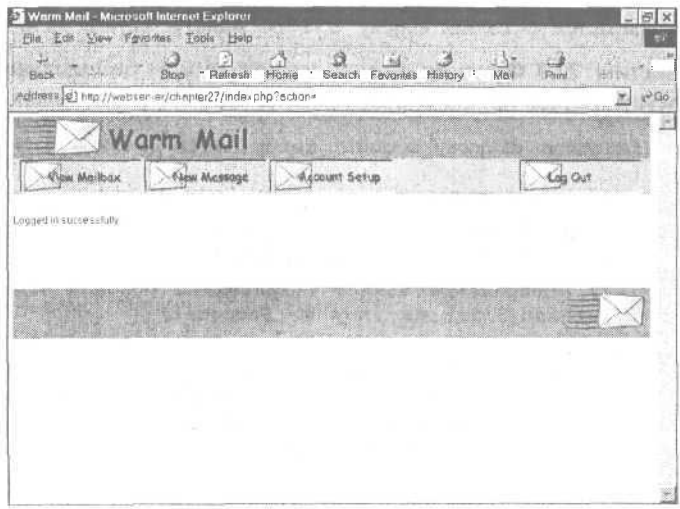


РИСУНОК 27.3
После успешного входа в систему пользователь может начать работу с приложением.

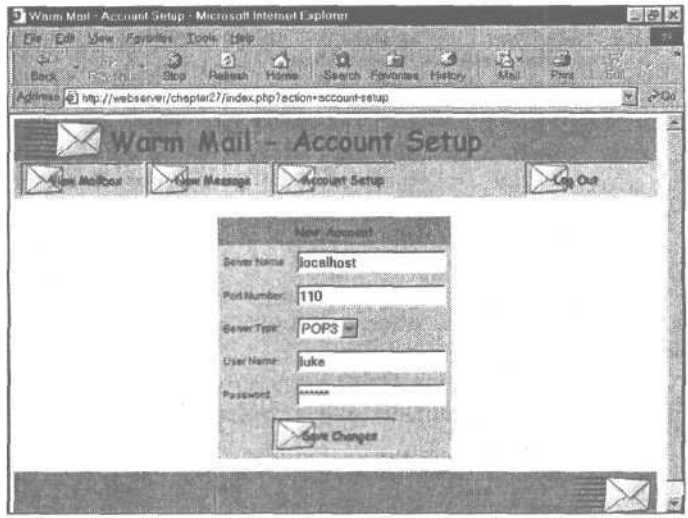


РИСУНОК 27.4
Прежде чем пользоваться электронной почтой, клиент должен ввести данные своей учетной записи.

Вернемся к сценарию из листинга 27.2. На этот раз значение переменной **$action** диктует другое поведение.

Заголовок создается с некоторыми отличиями:

```
do_html_header($auth_user, "Warm Mail - ".
                    format_action($action), $selected_account);
```

Важнее отличия в теле страницы:

```
case 'store-settings' :
case 'account-setup' :
case 'delete-account' :
{
   display account_setup($auth_user);
   break;
}
```

Это типичное поведение — каждая команда вызывает функцию. В данном случае вызывается функция display_account_setup(). Ее код показан в листинге 27.3.

Листинг 27.3 Функция display_account_setup() библиотеки outputjns.php осуществляет прием и отображение данных учетной записи

```
function display_account_setup($auth_user)
{
   // отображение пустой формы "new account"  ("новая учетная запись")
   display_account_form($auth_user);
   $list = get_accounts($auth_user);

   // отображение каждой сохраненной учетной записи
   foreach($list as $key => $account)
   {

      // Отображение форм для всех учетных записей.
      // Обратите внимание на отправку пароля для всех учетных
      // записей в HTML-содержимом.
      // Это не лучшее решение
```

```
        display_account_form($auth_user,
                             $account['accountid'],
                             $account['server'],
                             $account['remoteuser'],
                             $account['remotepassword'],
                             $account['type'],
                             $account['port']);
    }
}
```

При вызове эта функция отображает пустую форму для добавления новой учетной записи. Затем следуют редактируемые формы, содержащие данные всех учетных записей пользователя. Функция **display_account_form()** отображает форму, показанную на рис. 27.4. Можно видеть, что функция используется двумя способами: при ее вызове без параметров отображается пустая форма, а при вызове с полным набором параметров выводится существующая запись. Эта функция входит в библиотеку **output_fns.php**. Она просто выводит HTML-содержимое и здесь не рассматривается.

Для извлечения существующих учетных записей применяется функция **get_accounts()** библиотеки **mail_fns.php**. Она показана в листинге 27.4.

Листинг 27.4 Функция get_accounts() библиотеки mail_fns.php извлекает данные всех учетных записей для определенного пользователя

```
function get_accounts($auth_user)
{
   $list = array();
   if(db_connect())
   {
      $query = "select * from accounts where username = '$auth_user'";
      $result = mysql_query($query);
              if($result)
      {
         while($settings = mysql_fetch_array($result))
            array_push( $list, $settings);
      }
      else
         return false;
   }
   return $list;
}
```

Эта функция выполняет подключение к базе данных, извлекает все учетные записи для определенного пользователя и возвращает их в виде массива.

Создание новой учетной записи

Когда пользователь заполняет форму учетной записи и выполняет щелчок на кнопке Save Changes (сохранить изменения), активизируется действие **store-settings.** Рассмотрим код обработки этого события из сценария **index.php** На этапе предварительной обработки выполняется следующий код:

```
case 'store-settings' :
{
   store_account_settings($auth_user, $HTTP_POST_VARS);
   break;
}
```

Функция **store_account_settings()** записывает данные новой учетной записи в базу данных. Код этой функции показан в листинге 27.5.

Листинг 27.5 Функция store_account_settings() библиотеки mail_fns.php сохраняет для пользователя данные новой учетной записи

```
function store account_settings($auth_user, $settings)
{
  if(!filled_out($settings))
  {
    echo "All fields must be filled in.  Try again.<br><br>";
    return false;
  }
  else
  {
    if($settings['account']>0)
      $query = "update accounts  set server = '$settings[server]',
                    port = $settings[port], type = '$settings[type]',
                    remoteuser = '$settings[remoteuser]',
                    remotepassword = '$settings[remotepassword]'
                  where accountid = $settings[account]
                    and username='$auth_user'";
    else
      $query = "insert into accounts values ('$auth_user',
                      '$settings[server]', $settings[port],
                      '$settings[type]','$settings[remoteuser]',
                      '$settings[remotepassword]',NULL)";
    if(db connect() && mysql_query($query))
    {
      return true;
    }
    else
    {
      echo "could not store changes.<br><br><br><br><br><br>";
      return false;
    }
  }
}
```

Функция реализует две опции — ввод новой учетной записи и обновление существующей. Она выполняет запрос сохранения данных учетной записи.

После сохранения данных учетной записи осуществляется возврат к этапу реализации тела страницы сценария **index.php:**

```
case 'store-settings' :
case 'account-setup' :
case 'delete-account' :
{
  display account setup($auth_user);
  break;
}
```

Затем, как и прежде, выполняется функция **display_account_setup()** для вывода списка данных учетной записи пользователя. Вновь созданная учетная запись будет помещена в базу данных.

Изменение существующей учетной записи

Процесс изменения учетной записи во многом подобен описанному выше. Пользователь может изменить данные и щелкнуть на кнопке Save Changes. И снова будет активизировано действие **store-settings.** Однако на этот раз произойдет обновление данных учетной записи, а не добавление новой записи в базу данных.

Удаление учетной записи

Для этого необходимо щелкнуть на кнопке Delete Account (удалить учетную запись), которая отображается под каждым списком учетных записей. В результате активизируется действие **delete-account.**

В первом разделе сценария **index.php** выполняется следующий код:

```
case 'delete-account' :
{
  delete_account($auth_user, $account);
  break;
}
```

Этот код вызывает функцию **delete_account()**, показанную в листинге 27.6. Удаление учетных записей необходимо выполнять до обработки заголовка, поскольку именно внутри заголовка предоставляется выбор учетной записи для использования. Список учетных записей необходимо обновить, чтобы он мог быть правильно отображен.

Листинг 27.6 Функция delete_account() библиотеки mail_fns.php удаляет данные единственной учетной записи

```
function delete_account ($auth_user, $accountid)
{
  // удаление из базы данных одной из учетных записей
  // текущего пользователя
  $query = "delete from accounts where
                  accountid='$accountid' and
                  username = '$auth_user'";
  if(db_connect())
  {
    $result = mysql_query($query);
  }
  return $result;
}
```

После выполнения функций происходит возврат к разделу тела страницы сценария **index.php.** При этом выполняется следующий код:

```
case 'store-settings' :
case 'account-setup' :
case 'delete-account' :
{
  display_account_setup($auth_user);
  break;
}
```

Тот же код выполнялся ранее — он просто отображает список учетных записей пользователя.

Чтение почтовых сообщений

После того как пользователь создаст несколько учетных записей, можно перейти к главному — подключению к серверу с помощью учетных записей и чтение почты.

Выбор учетной записи

Для чтения почты необходимо выбрать одну из учетных записей. Текущий выбор хранится в переменной сеанса **$selected_account**.

Если пользователь зарегистрировал в системе единственную учетную запись, она автоматически выбирается при входе в систему:

```
if(number_of_accounts($auth_user)==1)
{
   $accounts = get_account_list($auth_user) ;
   $selected_account = $accounts[0];
   session_register("selected_account");
}
```

Функция **number_of_accounts()** библиотеки **mail_fns.php** служит для выявления случаев, когда пользователь имеет более одной учетной записи. Функция **get_account_list()** извлекает массив имен учетных записей пользователя. В данном случае запись одна. Ее можно извлечь как значение массива с индексом 0.

Функция **number_of_accounts()** показана в листинге 27.7.

Листинг 27.7 Функция number_of_accounts() библиотеки mail_fns.php вычисляет, сколько учетных записей зарегистрировал пользователь _

```
function number_of_accounts ($auth_user)
{
   // определение количества учетных записей,
   // принадлежащих пользователю

   $query = "select count(*) from accounts where username =
   '$auth_user'";

   if(db_connect())
   {
      $result = mysql_query($query);
         if ($result)
            return mysql_result($result, 0, 0) ;
   }
   return 0;
}
```

Функция **get_account_list()** подобна рассмотренной ранее функции **get_accounts()** за исключением того, что извлекаются только имена учетных записей.

Если пользователь зарегистрировал несколько учетных записей, потребуется выбрать одну из них для использования. В этом случае заголовки будут содержать список SELECT, в котором перечислены доступные учетные записи. При выборе одной из них автоматически отображается соответствующий почтовый ящик, как показано на рис. 27.5.

Данная опция **SELECT** генерируется в функции **do_html_header()** библиотеки **output_fns.php**, как показано в следующем фрагменте кода:

```
<?
   // поле списка выводится только в случае, когда
   // пользователь имеет более одной учетной записи
```

```
    if(number_of_accounts($auth_user)>1)
    {
      echo "<form target='index.php?action=open-mailbox' method=post>";
      echo '<td bgcolor = "#ff6600" align = right valign = middle>';
      display_account_select($auth_user, $selected_account);
      echo '</td>';
      echo "</form>";
    }
?>
```

Обычно мы избегаем обсуждения HTML-кода, используемого в примерах данной книги. Однако для HTML-кода, генерируемого функцией **display_account_select()** делается исключение.

В зависимости от учетных записей текущего пользователя, функция **display_account_select()** генерирует HTML-код следующим образом:

```
<select onchange=window.location=this.options[selectedIndex].value
   name=account>
   <option value = 0 selected>
     Choose Account</a>
   <option value = 'index.php?action=select-account&account=10'>
     mail.domain.com
   </option>
   <option value = 'index.php?action=select-account&account=11'>
     mail.server.com
   </option>
   <option value='index.php?action=select-account&account=9'>
     localhost
   </option>
</select>
```

Большая часть кода создает элемент списка, но в нем также содержится небольшой сценарий на JavaScript. Подобно тому, как PHP генерирует HTML-содержимое, этот язык можно использовать для создания сценариев со стороны клиента.

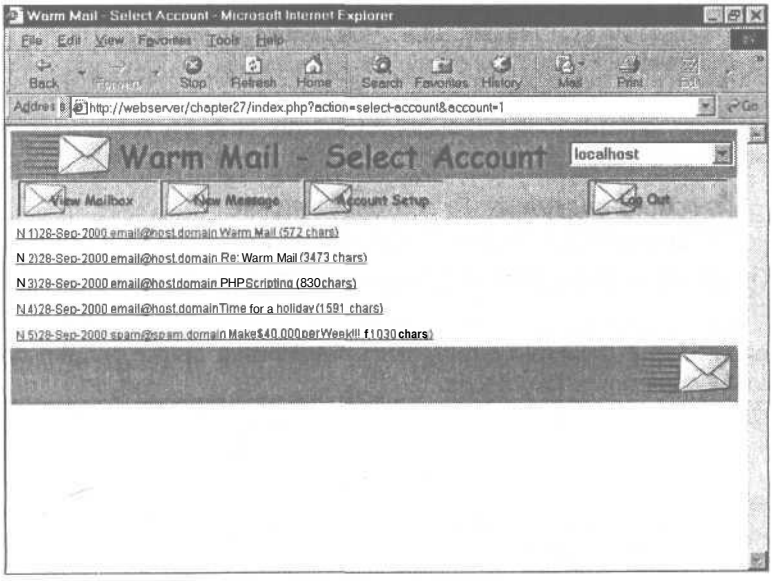


РИСУНОК 27.5
После выбора в списке SELECT учетной записи localhost загружается и отображается содержимое почтового ящика.

Как только для списка происходит событие изменения, JavaScript-сценарий для свойства **window.location** устанавливает значение опции. Если пользователь выбирает первую опцию списка **SELECT,** для свойства **window.location** будет установлено значение **'index.php?action=select-account&account=10'**. В результате будет загружен ресурс по данному URL-адресу. Очевидно, если браузер пользователя не поддерживает JavaScript либо выполнение JavaScript-сценариев отключено, этот код не даст желаемого результата.

Функция **display_account_select()** библиотеки **output_fns.php** служит для извлечения и отображения списка допустимых учетных записей. Кроме того, она вызывает рассмотренную ранее функцию **get_account_list()**.

В результате выбора одной из опций списка **SELECT** активизируется событие **select_account**. Как видно из рис. 27.5, значение свойства **window.location** присоединено к URL-адресу вместе с идентификатором выбранной учетной записи.

Получается двойной эффект. Во-первых, на этапе предварительной обработки (сценария **index.php**) выбранная учетная запись сохраняется в переменной сеанса **$selected_account** следующим образом:

```
case 'select-account' :
{
// если выбрана допустимая учетная запись,
// она сохраняется в переменной сеанса
if($account&&account_exists($auth_user, $account))
{
   $selected_account = $account;
   session_register('selected_account');
}
}
```

Во-вторых, когда выполняется этап построения тела страницы, задействуется следующий код:

```
case 'select-account' :
case 'view-mailbox' :
{
   // если почтовый ящик только что выбран либо выбрана
   // опция view mailbox, отобразить его содержимое
   display_list($auth_user, $selected_account);
   break;
}
```

Предпринимаются те же действия, как если бы пользователь выбрал опцию View Mailbox, которая рассматривается в следующем разделе.

Просмотр содержимого почтового ящика

Содержимое почтового ящика можно просмотреть с помощью функции **display _list()**. Она отображает список всех сообщений почтового ящика. Код функции показан в листинге 27.8.

Листинг 27.8 Функция display_list() библиотеки output_fns.php отображает все сообщения почтового ящика

```
function display_list($auth_user, $accountid)
{
   // отображение списка сообщений данного почтового ящика

   global $table_width;
   if(!$accountid)
```

```
    {
       echo "No mailbox selected<br><br><br><br><br><br>." ;
    }
    else
    {
       $imap = open_mailbox($auth_user, $accountid);

       if($imap)
       {
          echo "<table width = $table_width cellspacing = 0
                        cellpadding = 6 border = 0>" ;
          $headers = imap_headers($imap);
          // можно повторно сформатировать эти данные либо
          // получить другую информацию с помощью
          // imap_fetchheaders, но и такой отчет неплох, поэтому
          // просто воспользуемся оператором echo
          $messages = sizeof($headers) ;
          for($i = 0; $i<$messages; $i++)
          {
             echo "<tr><tdbgcolor = ' " ;
             if($i%2)
                echo "#ffffff";
             else
                echo "#ffffcc";
             echo " '><a href = ' index.php?action=view-
                 message&messageid=".($i+1)."'>";
             echo $headers[$i] ;
             echo "</a></td></tr>\n";
          }
          echo "</table>";
       }
       else
       {
          $account = get_account_settings ($auth_user, $accountid) ;
          echo "could not open mail box
                          " .$account['server'] ." .<br><br><br><br>"
       }
    }
}
```

В этой функции начинается использование IMAP-функций PHP. Двумя ключевыми моментами являются открытие почтового ящика и чтение заголовков сообщений.

Почтовый ящик для учетной записи пользователя открывается с помощью функции **open_mailbox()** из библиотеки **mail_fns.php.** Ее код показан в листинге 27.9.

Листинг 27.9 Функция open_mailbox() библиотеки mail_fns.php выполняет подключение к почтовому ящику пользователя

```
function open_mailbox($auth_user, $accountid)
{
   // выбор почтового ящика, если он единственный
   if(number_of_accounts($auth user)==1)
   {
      $accounts = get_account_list($auth_user);
      $selected_account = $accounts[0];
      session_register("selected_account");
      $accountid = $selected_account;
   }
   // подключение к серверу POP3 или IMAP, выбранному пользователем
```

```
    $settings = get_account_settings ($auth_user, $accountid) ;
    if ( !sizeof ($settings) ) return 0;
    $mailbox = "{".$settings[server];
    if ($settings[type] =='POP3')
       $mailbox .= '/pop3';
    $mailbox.= ":".$settings[port]."}INBOX";

// запрет предупреждений, необходимо проверять возвращаемое значение
@  $imap = imap_open ($mailbox, $settings [remoteuser] ,
                      $settings[remotepassword]) ;

    return  $imap;

}
```

Почтовый ящик открывается с помощью функции **imap_open().** Она описывается следующим образом:

```
int imap_open (string mailbox, string username,
               string password [, int flags])
```

Ниже перечислены параметры, которые необходимо передать функции:

- **mailbox** — этот сценарий должен содержать имена сервера и почтового ящика, а также, необязательно, номер порта и название протокола. Данная строка имеет следующий формат:

  ```
  {hostname/protocol:port}boxname
  ```

  Если протокол не указан, по умолчанию принимается IMAP. Из рассмотренного выше фрагмента кода видно, что, когда пользователь указывает протокол POP3 для определенной учетной записи, именно этот протокол и задается.

  Например, чтобы читать почту с локального компьютера с использованием стандартных портов, для IMAP применяется следующее имя почтового ящика:

  ```
  {localhost:143}INBOX
  ```

  а данное имя для протокола POP3:

  ```
  {localhost/pop3:110}INBOX
  ```

- **username** — имя пользователя для учетной записи
- **password** — пароль для учетной записи

Кроме того, допускается передача необязательных флагов для указания опций, таких как **"open mailbox in read-only mode"** (открыть почтовый ящик в режиме "только для чтения").

Обратите внимание, что строка **mailbox** до передачи функции **imap_open()** составляется из фрагментов с помощью операции конкатенации. Создавать строку следует внимательно, поскольку строки, содержащие символы {$, в языке PHP 4 создают проблемы.

В результате вызова функции возвращается поток IMAP, если почтовый ящик может быть открыт, и значение **false** в противном случае.

После завершения работы с потоком ШАР его можно закрыть с помощью функции **imap_close(*imap_stream*)**.

В рассматриваемой функции поток IMAP передается обратно в главную программу. Затем применяется функция **imap_headers()**, которая извлекает заголовки почтовых сообщений с целью их отображения:

```
$headers = imap_headers($imap);
```

Эта функция возвращает информацию заголовков для всех сообщений почтового ящика, к которому выполнено подключение. Информация возвращается в виде массива, где каждая строка соответствует сообщению. Вывод не форматируется. Сообщения отображаются построчно, как показано на рис. 27.5.

Можно вывести более полную информацию о заголовках с помощью функции, название которой легко спутать с предыдущей — **imap_header()**. В данном случае функция **imap_headers()** выводит вполне достаточные сведения.

Чтение почтовых сообщений

Каждая строка, выводимая функцией display_Iist(), представляет собой ссылку на определенное сообщение электронной почты.

Ниже приводится форма всех ссылок:

```
index.php?action=view-message&messageid=6
```

Здесь mcssagcid представляет собой числовую последовательность, использованную в извлеченных ранее заголовках. Обратите внимание, что нумерация IMAP-сообщений начинается с единицы, а не с нуля.

После щелчка на одной из этих ссылок генерируется вывод, подобный показанному на рис. 27.6.

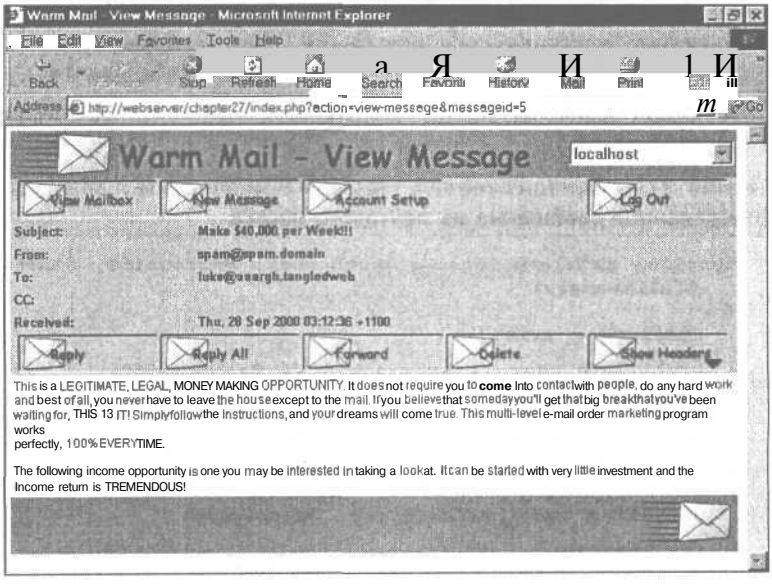


РИСУНОК 27.6
С помощью действия view-message выводится определенное сообщение. В данном случае это спэмминг.

При вводе данных параметров в сценарий **index.php** выполняется следующий код:

```
case 'show-headers' :
case 'hide-headers' :
case 'view-message' :
{
   // если сообщение только что выбрано из списка, либо в
   // режиме его просмотра выбрана опция сокрытия или
   // просмотра заголовков, загрузить сообщение
   $fullheaders = ($action=='show-headers') ;
   display_message($auth_user, $selected_account, $messageid,
       $fullheaders)
   break;
}
```

Проверяется равенство переменной **$action** строке **'show-headers'**. В данном случае проверка дает результат false и для переменной **$fullheaders** также устанавливается значение false. Действие **'show-headers'** рассматривается немного позже.

Строку

```
$fullheaders = ($action==' show-headers ');
```

можно заменить не таким лаконичным, но, возможно, более понятным вариантом:

```
if ($action==' show-headers ')
   $fullheaders = true ;
else
   $fullheaders = false;
```

Затем вызывается функция display_message(). Функция в основном реализует простой вывод HTML-содержимого, поэтому здесь не рассматривается. Она вызывает функцию **retrieve_message()** для извлечения требуемого сообщения из почтового ящика:

```
$message - retrieve_message ($auth_user, $accountid,
                             $messageid, $fullheaders) ;
```

Функция **retrieve_message()** содержится в библиотеке **mail_fns.php**. Ее код приводится в листинге 27.10.

Листинг 27.10 Функция **retrieve_message()** библиотеки **mail_fns.php** извлекает определенное сообщение из почтового ящика

```
function retrieve_message($auth_user, $accountid, $messageid,
   $fullheaders)
{
   $message = array () ;

   if(!($auth_user && $messageid && $accountid))
      return false;

   $imap = open_mailbox($auth_user, $accountid);
   if(!$imap)
      return false;

   $header = imap_header($imap, $messageid);

   if (! $header)
      return false;

   $message['body'] = imap_body($imap, $messageid);
   if(!$message['body'])
      $message['body'] = '[This message has no body]\n\n\n\n\n\n';
```

```
    if ($fullheaders)
       $message [ ' f ullheaders ' ] = imap_fetchheader ($imap,$messageid);
    else
       $message['fullheaders'] = '';

    $message [ ' subject ' ] =$header->subject;
    $message [ ' fromaddress ' ] -$header->fromaddress;
    $mess age [ ' toaddress ' ] = $header->toaddress;
    $message['ccaddress'] =   $header->ccaddress;
    $message['date'] =    $header->date;

    // обратите внимание на возможность получения более
    // подробной информации с помощь» полей from и to
    // вместо fromaddress и toaddress, но так проще

    imap_close($imap);
    return $message;
}
```

Опять-таки, для открытия почтового ящика используется функция **open_mailbox().** Однако на этот раз требуется получить определенное сообщение. С помощью этой библиотеки функций заголовки и тело сообщения загружаются отдельно.

Здесь применены три IMAP-функции — **imap_header(), imap_fetchheader() и imap_body().** Обратите внимание на отличия двух первых функций от использованной ранее функции **imap_headers.** Их названия легко перепутать. Вот краткая информация по функциям:

- **imap_headers()** — возвращает перечень заголовков всех сообщений почтового ящика в виде массива, где каждому сообщению соответствует один элемент.
- **imap_header()** — возвращает заголовки определенного сообщения в форме объекта.
- **imap_fetchheader()** — возвращает заголовки определенного сообщения в форме строки.

В данном случае функция **imap_header()** используется для заполнения полей определенного заголовка, а **imap_fetchheader()** — для отображения всех заголовков, если это затребовано. (Мы еще вернемся к этой теме.)

Функции **imap_header() и imap_body()** служат для создания массива, содержащего все элементы интересующего сообщения.

Функция **imap_header()** вызывается следующим образом:

```
$header = imap_header ($imap, $messageid) ;
```

Затем можно извлечь из объекта каждое необходимое поле:

```
$message['subject'] = $header->subject;
```

Для добавления тела сообщения в массив вызывается функция **imap_body()**:

```
$message['body'] = imap_body ($imap, $messageid) ;
```

Наконец, функция **imap_close()** закрывает почтовый ящик и возвращает созданный массив. Затем можно с помощью функции **display_message()** отобразить поля сообщения в виде, показанном на рис. 27.6.

Просмотр заголовков сообщений

На рис. 26.7 можно заметить кнопку Show Headers (показать заголовки). Она активизирует действие **show-headers,** которое выводит все заголовки в процессе отображения сообщения. Генерируемый после щелчка на кнопке вывод приведен на рис. 27.7.

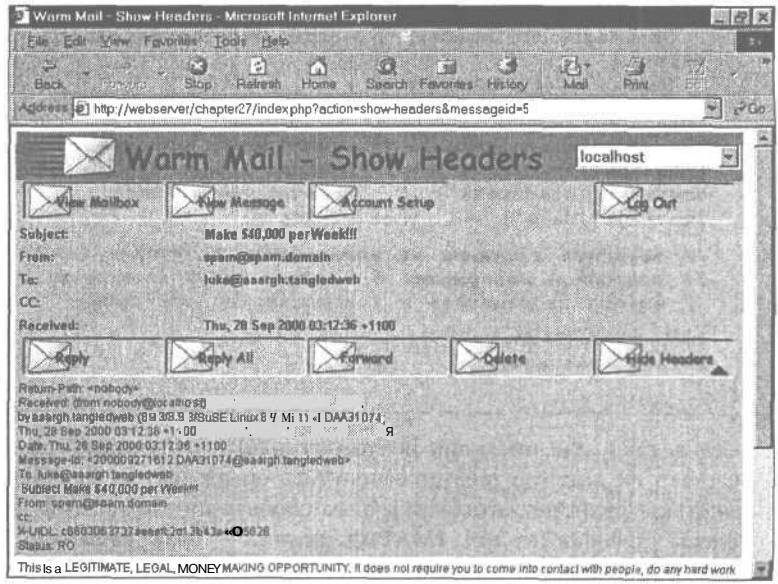


РИСУНОК 27.7 *Действие show-headers, отображающее все заголовки определенного сообщения, помогает пользователю отслеживать происхождение спэмминга.*

Обработка события **view-message** охватывает также действие **show-headers** (и его дополнение — **hide-headers).** Когда эта опция выбирается, выполняются те же операции, что и раньше. Однако в функции **retrieve_message()** осуществляется дополнительный вывод полного текста заголовков:

```
if($fullheaders)
     $message['fullheaders'] = imap_fetchheader($imap, $messageid);
```

Затем эти заголовки можно отобразить пользователю.

Удаление почтовых сообщений

После щелчка на кнопке Delete во время отображения определенного сообщения активизируется действие **delete.** В результате выполняется следующий фрагмент кода сценария **index.php:**

```
case 'delete' :
{
   delete_message($auth_user, $selected_account, $messageid);
   // оператор 'break' опущен умышленно — будет выполнен
   // переход к следующему оператору case
}
case 'select-account' :
case 'view-mailbox' :
{
   // если почтовый ящик только что открыт либо находится
   // в режиме просмотра, отобразить его содержимое
   display_list($auth_user, $selected_account);
   break;
}
```

Сначала с помощью функции **delete_message()** удаляется сообщение, а затем отображается обновленное содержимое почтового ящика, как описывалось ранее.

Код функции **delete_message()** показан в листинге 27.11.

Листинг 27.11 Функция delete_message() библиотеки mail_fns.php удаляет определенное сообщение из почтового ящика

```
function delete_message($auth_user, $accountid, $message_id)
{
   // удаление одного сообщения из сервера
   $imap = open_mailbox($auth_user, $accountid);
   if ($imap)
   {
      imap_delete($imap, $message_id);
      imap_expunge($imap);
      imap_close($imap);
      return true;
   }
   return false;
}
```

Здесь использован ряд функций IMAP. К новым из них относятся **imap_delete() и imap_expunge()**. Обратите внимание, что функция **imap_delete()** только помечает сообщения для удаления. Можно помечать любое количество сообщений. В результате вызова функции **imap_expunge()** сообщения действительно удаляются.

Отправка почты

Наконец, перейдем к отправке почты. Для этого сценарий предоставляет несколько способов: пользователь может отправить новое сообщение, ответить на сообщение, а также переадресовать его. Рассмотрим реализацию этих возможностей.

Отправка нового сообщения

Пользователь может выбрать эту опцию, щелкнув на кнопке New Message (Новое сообщение). В результате будет активизировано действие **new-message,** которое выполняет следующий фрагмент кода сценария **index.php**:

```
case 'new-message' :
{
   display_new_message_form($auth_user, $to, $cc, $subject, $body);
   break;
}
```

Форма нового сообщения представляет собой просто форму отправки сообщения. Ее вид показан на рис. 27.8. В действительности на рисунке показан режим переадресации, но форма используется та же самая.

В результате щелчка на кнопке Send Message вызывается действие **send-message,** которое выполняет следующий код:

```
case 'send-message' :
{
   if (send_message($to, $cc, $subject, $message))
      echo "<p>Message sent.<br><br><br><br><br><br>";
   else
      echo "<p>Could not send message.<br><br><br><br><br><br>";
   break;
}
```

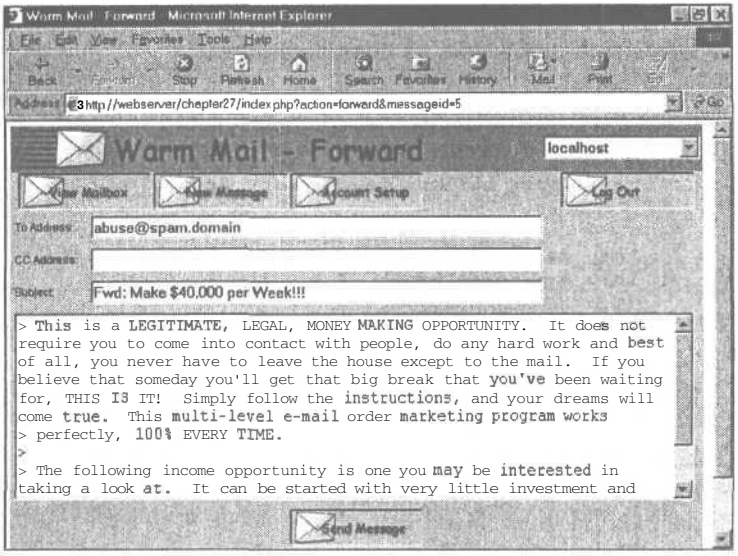


РИСУНОК 27.8
С помощью переадресации можно заявить на отправителя спэмминга.

Здесь вызывается функция **send_message(),** которая осуществляет отправку сообщения. Ее код показан в листинге 27.12.

Листинг 27.12 Функция send_message() библиотеки mail_fns.php отправляет напечатанное пользователем сообщение

```
function send_message($to, $cc, $subject, $message)
{
   // отправка одного сообщения средствами PHP
   global $auth_user;
   if (!db_connect())
   {
      return false;
   }
   $query = "select address from users where username='$auth_user'";
   $result = mysql_query($query);
   if (!$result)
   {
      return false;
   }
   else if (mysql_num_rows($result)==0)
   {
      return false;
   }
   else
   {
      $other = "From: ".mysql_result($result, 0, "address")."\r\ncc: $cc";
      if (mail($to, $subject, $message, $other))
         return true;
      else
      {
         return false;
      }
   }
}
```

Можно видеть, что здесь для отправки сообщения электронной почты используется функция **mail()**. Однако сначала выполняется загрузка адреса электронной почты пользователя из базы данных. Этот адрес будет указан в поле From (от).

Ответ на сообщение или его переадресация

Функции Reply (ответить), Reply All (ответить всем) и Forward (переслать) выполняют отправку сообщений таким же образом, как и функция New Message. Их различие заключается в частичном заполнении формы сообщения перед ее отображением пользователю. Вернемся к рис. 27.8. Текст пересылаемого сообщения выводится с отступом. Его строки начинаются с символа ">". В начале строки Subject (тема) указывается выполняемое действие. Функции Reply и Reply All заполняют поля адреса получателя и темы, а также выводят текст прежнего сообщения с отступом подобным же образом.

Все это реализует код третьего раздела сценария **index.php**:

```
case 'reply-all' :
{
   //установка прежнего значения поля адреса отправки копии
   if(!$imap)
      $imap = open_mailbox($auth_user, $selected_account);
   if($imap)
   {
      $header = imap_header($imap, $messageid);
      if($header->reply_toaddress)
         $to = $header->reply_toaddress;
      else
         $to e $header->fromaddress;
      $cc = $header->ccaddress;
      $subject = 'Re: '.$header->subject;
      $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
      imap_close($imap);

      display new_message_form($auth_user, $to, $cc, $subject, $body);
   }
   break;
}
case 'reply' :
{
   //установка в поле To адреса из поля Reply-to либо From
   if(!$imap)
      $imap = open mailbox($auth_user, $selected account);
   if($imap)
   {
      $header = imap_header($imap, $messageid);
      if($header->reply_toaddress)
         $to = $header->reply_toaddress;
      else
         $to = $header->fromaddress;
      $subject = 'Re: '.$header->subject;
      $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
      imap_close($imap);

      display_new_message_form($auth user, $to, $cc, $subject, $body);
   }

   break;          //оператор 'break' опущен умышленно
}
```

```
case 'forward' :
{
 //помещение прежнего сообщения, выделенного кавычками, в текущее сообщение
  if(!$imap)
     $imap = open_mailbox($auth_user, $selected_account);
  if($imap)
  {
     $header = imap_header($imap, $messageid);
     $body = add_quoting(stripslashes(imap_body($imap, $messageid)));
     $subject = 'Fwd: '.$header->subject;
     imap_close($imap);

     display_new_message_form($auth_user, $to, $cc, $subject, $body);
  }
  break;
}
```

Каждая из рассмотренных опций создает соответствующие заголовки, применяет элементы форматирования и вызывает функцию **display_new_message()** для вывода формы.

Этим исчерпывается набор функциональных возможностей Web-приложения управления электронной почтой.

Расширение проекта

Проект можно дополнить многими расширениями и усовершенствованиями. За образец можно принять используемую программу электронной почты. Ниже перечислены некоторые полезные расширения проекта:

- Предусмотреть возможность регистрации пользователей на сайте. (Для этого можно повторно использовать некоторые фрагменты кода из главы 24.)
- Многие пользователи имеют несколько адресов электронной почты, например, личный и служебный. Можно дать им возможность применять несколько адресов путем перемещения сохраненных адресов из таблицы пользователей в таблицу учетных записей. Потребуется внести изменения и в ряд других фрагментов кода. Форма отправки сообщения должна будет содержать поле со списком, в котором выбирается используемый адрес.
- Добавить возможность отправки, приема и просмотра сообщений с присоединениями. Если пользователи могут отправлять присоединения, необходимо предусмотреть функции загрузки файлов, о чем шла речь в главе 16. Отправка почты с присоединениями рассматривается в главе 28.
- Реализовать функции адресной книги
- Добавить функции чтения новостей. Чтение содержимого NNTP-сервера с помощью функций IMAP практически не отличается от чтения сообщений из почтового ящика. Достаточно лишь при вызове функции **imap_open()** указать другой номер порта и протокол. Вместо присвоения почтовому ящику имени INBOX можно именовать группу новостей, из которой будет выполняться чтение. В комбинации с функциями создания потоков из проекта главы 29 можно создать Web-приложение чтения новостей, в котором реализованы потоки.

Что дальше

Следующая глава посвящена другому проекту, связанному с электронной почтой. Это приложение, реализует отправку информационных бюллетеней по нескольким темам подписчикам сайта.

Глава 28

Создание менеджера списков рассылки

После того как создана база подписчиков Web-сайта, было бы замечательно иметь возможность поддерживать с ними контакт, рассылая информационные бюллетени. В этой главе мы реализуем интерфейс менеджера списков рассылки (mailing list manager, в дальнейшем MLM). Некоторые MLM позволяют каждому подписчику отправлять сообщения другим подписчикам. Наша программа будет представлять собой систему информационных бюллетеней, в которой только администратор может отправлять сообщения. Давайте назовем ее Pyramid-MLM.

Эта система будет аналогична другим доступным на рынке программам. Чтобы получить некоторое представление о стоящих перед нами задачах, обратитесь к сайту

```
http://www.topica.com
```

Создаваемое нами приложение позволит администратору создавать несколько списков рассылки и отправлять информационные бюллетени отдельно в каждый из этих списков. Приложение будет использовать загрузку файлов, чтобы администратор мог загружать текст и HTML-версии информационных бюллетеней, созданные в автономном режиме. Другими словами, для создания информационных бюллетеней администраторы могут использовать любые программы по своему выбору.

Пользователи смогут подписываться на любые списки, представленные в нашем сайте, и выбирать, в какой форме они желают получать информационные бюллетени — простой текст или HTML.

Задача

Нам необходимо создать интерактивный набор информационных бюллетеней и систему отправки. Эта система должна допускать создание и рассылку пользователям различных информационных бюллетеней и давать возможность пользователям подписываться на один или несколько информационных бюллетеней.

В частности, система должна удовлетворять следующим требованиям:

- Администраторы должны иметь возможность определять и изменять списки рассылки.
- Администраторы должны иметь возможность рассылать информационные бюллетени в текстовой форме и в виде HTML всем подписчикам в рамках одного списка рассылки.
- Пользователи должны иметь возможность регистрироваться на сайте и вводить и изменять сведения о себе.
- Пользователи должны иметь возможность подписываться на любые списки, представленные на сайте.
- Пользователи должны иметь возможность отменять подписку на ранее подписанные списки рассылки.
- Пользователи должны иметь возможность сохранять свои предпочтения относительно получения информационных бюллетеней в формате HTML или в виде простого текста.
- В целях обеспечения безопасности пользователи не должны иметь возможность отправлять сообщения электронной почты в списки рассылки или видеть адреса электронной почты других подписчиков.
- Пользователи и администраторы должны иметь возможность просматривать информацию о списках рассылки.
- Пользователи и администраторы должны иметь возможность просматривать прошлые информационные бюллетени, отправленные в список (архив).

Компоненты решения

Для выполнения предъявляемых требований потребуется ряд компонентов. Основными являются определение базы данных списков, подписчиков и заархивированных информационных бюллетеней; загрузка информационных бюллетеней, которые были созданы в автономном режиме; отправка сообщений электронной почты с присоединениями.

Определение базы данных списков и подписчиков

Мы будем отслеживать имя пользователя и пароль каждого пользователя системы, а также список списков рассылки, на которые они подписались. Мы будем также хранить предпочтения каждого пользователя относительно получения сообщений в виде простого текста или в формате HTML, чтобы ему можно было отправлять соответствующую версию информационного бюллетеня.

Администратором будет пользователь, наделенный особыми правами создавать новые списки рассылки и отправлять в них информационные бюллетени.

Для подобной системы желательно иметь архив ранее отправленных информационных бюллетеней. Подписчики могут не хранить предшествующие сообщения, но, возможно, захотят просмотреть некоторые из них. Архив может также служить в качестве

рекламного средства, поскольку потенциальные подписчики смогут посмотреть, как выглядят информационные бюллетени.

Определение этой базы данных в MySQL и интерфейса к ней в PHP не будет представлять собой ничего нового или сложного.

Загрузка файлов

Как уже упоминалось, нам требуется интерфейс, позволяющий администратору отправлять информационные бюллетени. Но мы еще ничего не сказали о том, как администраторы будут создавать эти бюллетени. Можно было создать форму, в которую администраторы могли бы вводить или вставлять содержимое бюллетеня. Однако пользователям будет удобнее, если администраторы смогут создавать бюллетень в предпочитаемом ими редакторе, а затем загружать файл на Web-сервер. Это также упростит администратору добавление изображений в информационный бюллетень HTML.

Для решения этой задачи можно использовать возможность загрузки файлов, описанную в главе 16.

Нам придется использовать несколько более сложную форму, чем та, которая применялась ранее. Требуется, чтобы администратор мог загружать как текстовую, так и HTML версию бюллетеня, а также любые изображения, внедряемые в HTML.

После того как информационный бюллетень загружен, администратору требуется интерфейс, чтобы просмотреть бюллетень перед его отправкой. Таким образом администратор может убедиться в корректности загрузки всех файлов.

Отправка сообщений электронной почты с присоединениями

В этом проекте желательно, чтобы в соответствии с предпочтениями пользователей бюллетени им можно было отправлять в виде простого текста или в виде "украшенной" HTML-версии.

Для отправки HTML-файла с внедренными в него изображениями требуется отыскать способ отправки присоединений. Простая PHP-функция **mail()** не обеспечивает удобной поддержки отправки присоединений. Поэтому вместо нее мы будем использовать замечательный класс HTML MIME Mail, созданный Ричардом Хейесом (Richard Heyes). Это класс может выполнять обработку HTML-присоединений и будет автоматически присоединять любые изображения, содержащиеся в HTML-файле.

Наиболее современную версию этого класса можно получить из Web-сайта

```
http://www.heyes-computing.net/scripts/
```

(Этот класс можно найти также на сопровождающем книгу CD-ROM.)

Вы можете свободно использовать этот сценарий для собственных целей. В случае его использования достаточно отправить открытку автору. Адрес электронной почты автора можно найти на его Web-сайте.

Обзор решения

При разработке этого проекта мы снова воспользуемся управляемым событиями подходом к созданию кода, как это было сделано в главе 27.

Как и ранее, мы начнем разработку, начертив набор схем, изображающих последовательности действий, которые пользователи могут выполнять в системе. В данном случае три схемы представляют три различных набора взаимодействий пользователя с системой. Пользователи имеют право выполнять различные действия, когда они не зарегистрированы в системе, когда зарегистрированы в качестве обычных пользователей

и когда зарегистрированы в качестве администраторов. Возможные действия показаны, соответственно, на рис. 28.1, 28.2 и 28.3.



РИСУНОК 28.1 *Для незарегистрированных пользователей доступно ограниченное количество действий.*

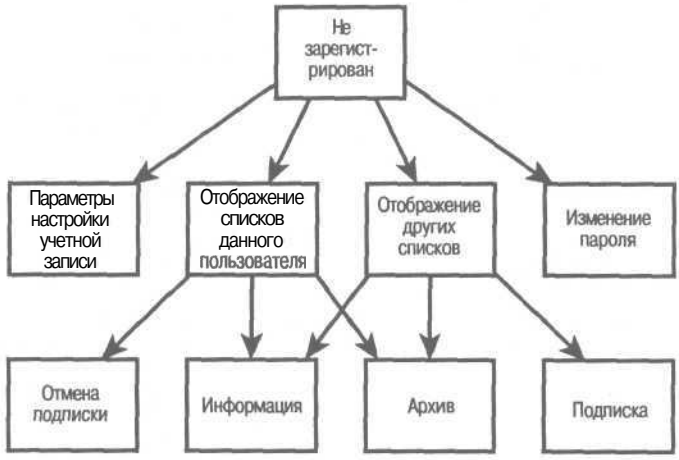


РИСУНОК 28.2 *После регистрации пользователи могут изменять свои предпочтения за счет изменения ряда параметров.*



РИСУНОК 28.3 *Администраторам доступны дополнительные действия*

На рис. 28.1 показаны действия, которые может предпринять незарегистрированный пользователь. Как видите, он может зарегистрироваться (если уже имеет учетную запись), создать учетную запись (если он ее еще не имеет) или просмотреть списки рассылки, доступные для подписки (это является частью тактики маркетинга).

Действия, которые пользователь может предпринять после регистрации, показаны на рис. 28.2. Он может изменять параметры настройки своей учетной записи (адрес электронной почты и предпочтений), изменять свой пароль и изменять набор списков рассылки, на которые он подписался ранее.

Действия, доступные для зарегистрированного администратора, показаны на рис. 28.3. Как видите, администратору доступно большинство действий, разрешенных для обычного пользователя, а также ряд дополнительных возможностей. Администратор может создавать новые списки рассылки, создавать новые сообщения для списков рассылки путем загрузки файлов и просматривать сообщения перед их отправкой.

Поскольку при разработке использовался управляемый событиями подход, основная часть приложения содержится в одном файле **index.php**, который вызывает набор библиотек функций. Краткое описание файлов этого приложения приведено в табл. 28.1.

Таблица 28.1 Файлы приложения менеджера списков рассылки

Имя файла	*Тип*	*Описание*
index.php	Приложение	Основной сценарий, реализующий поведение всего приложения.
include_fns.php	Функции	Коллекция файлов включения для этого приложения.
data_valid_fns.php	Функции	Коллекция функций для проверки вводимых данных
db_fns.php	Функции	Коллекция функций для подключения к базе данных **mlm**.
mlm_fns.php	Функции	Коллекция функций, специфичных для этого приложения.
output_fns.php	Функции	Коллекция функций для вывода HTML.
upload.php	Компонент	Сценарий, который управляет компонентом загрузки файлов администратором. Этот сценарий выделен в отдельный файл для упрощения поддержки безопасности.
user_auth_fns.php	Функции	Коллекция функций для аутентификации пользователей.
create_database.sql	SQL	SQL-код для создания базы данных mlm и для создания пользователя Web и пользователя с правами администратора.

Мы начнем реализацию проекта с создания базы данных, в которой будем хранить информацию о подписчиках и списках рассылки.

Создание базы данных

Для реализации этого приложения потребуется хранить следующую информацию:

- Lists (Списки): списки рассылки, доступные для подписки.
- Subscribers (Подписчики): пользователи системы и их предпочтения.
- Sub_lists (Списки подписки): запись списков, на которые подписались пользователи (отношение типа многие-ко-многим).

- Mail (Почта): запись об отправленных сообщениях электронной почты.
- Images (Изображения): поскольку требуется иметь возможность отправки сообщений электронной почты, состоящих из нескольких файлов (т.е., текста, кода HTML и набора изображений), требуется также отслеживать, какие изображения отправляются с каждым сообщением электронной почты.

SQL-код, создающий эту базу данных, приведен в листинге 28.1.

Листинг 28.1 create_database.sql — SQL-код для создания базы данных mlm

```
create database mlm;

use mlm;

create table lists
(
    listid int auto_increment not null primary key,
    listname char(20) not null,
    blurb varchar(255)
);

create table subscribers
(
  email char(100) not null primary key,
  realname char(100) not null,
  mimetype char(1) not null,
  password char(16) not null,
  admin tinyint not null
);

# хранит отношение между подписчиком и списком рассылки
create table sub_lists
(
  email char(100) not null,
  listid int not null
);

create table mail
(
  mailid int auto_increment not null primary key,
  email char(100) not null,
  subject char(100) not null,
  listid int not null,
  status char(10) not null,
  sent datetime,
  modified timestamp
);

# хранит изображения, которые отправляются с конкретным сообщением
create table images
(
  mailid int  not null,
  path char(100) not null,
  mimetype char(100) not null
);

grant select, insert, update, delete
on mlm.*
to mlm@localhost identified by 'password';

insert into subscribers values
('admin@localhost', 'Administrative User', 'H', password('admin'), 1);
```

Вспомните, что этот SQL-код можно выполнить, введя

```
mysql -u root -p < create_database.sql
```

При этом потребуется ввести свой пароль пользователя root. (Конечно, этот сценарий можно было бы выполнить из-под любого пользователя MySQL, обладающего соответствующими правами доступа; в данном случае права доступа пользователя root были использованы только ради простоты.) Прежде чем выполнять этот код, в нем следует изменить пароль для пользователя **mlm** и администратора.

Некоторые поля базы данных требуют небольших дополнительных пояснений, поэтому давайте бегло ознакомимся с ними.

Таблица **lists** содержит адреса электронной почты **(email)** и имена подписчиков **(realname).** В ней хранятся также их пароли **password** и флаг **(admin),** указывающий, является ли данный пользователь администратором. В **mimetype** будет также храниться тип сообщений электронной почты, предпочтительный для данного пользователя. Он может иметь значение H для HTML или T для текста.

Таблица **sublists** содержит адреса электронной почты **(email)** из таблицы **subscribers** и идентификаторы списков **listid** из таблицы **lists.**

Таблица **mail** содержит информацию о каждом сообщении электронной почты, отправляемом через систему. В ней хранится уникальный идентификатор **(mailid),** адрес электронной почты, откуда отправлено сообщение **(email),** строка темы сообщения **(subject)** либо **listid** списка, в который оно отправлено или будет отправлено. Фактический текст или HTML-код сообщения может быть большим файлом, поэтому архив собственно сообщений будет храниться вне базы данных. Кроме того, мы будем отслеживать некоторую информацию об общем состоянии: отправлено ли сообщение (status), когда оно было отправлено (sent) и метку времени, указывающую время последнего изменения этой записи **(modified).**

Наконец, таблица **images** используется для отслеживания любых изображений, связанных с сообщениями в формате HTML. Эти изображения также могут быть большими, поэтому для большей эффективности они будут храниться вне базы данных. Вместо них мы будем отслеживать идентификаторы **mailid,** с которыми они связаны, путь **path к** месту действительного хранения изображения и MIME-тип изображения **(mimetype),** например, **image/gif.**

Приведенный ранее SQL-код создает также пользователя, с использованием регистрационной записи которого должна подключаться PHP-программа, и пользователя с правами администратора для системы.

Архитектура сценария

Как и ранее, в этом проекте используется управляемый событиями подход. Основная часть приложения хранится в файле **index.php.** Этот сценарий имеет следующие четыре основных раздела:

1. Предварительная обработка: реализует обработку, которая должна быть выполнена до отправки верхних колонтитулов.
2. Создание и отправка верхних колонтитулов: создает и отправляет начало HTML-страницы.
3. Выполнение действия: отвечает на переданное событие. Как и в предыдущем примере, событие содержится в переменной **$action**.
4. Отправка нижних колонтитулов.

Почти вся обработка осуществляется в рамках этого файла. Как упоминалось ранее, приложение использует также библиотеки функций, перечисленные в табл. 28.1.

Полный текст сценария **index.php** приводится в листинге 28.2.

Листинг 28.2 index.php — Главный файл приложения **Pyramid-MLM**

```
<?
/*********************************************************************
* Раздел 1: предварительная обработка
ft*********************************************************************/

   include ('include_fns.php');
   session_start();

   $buttons = array();

   // дополните эту строку, если какая-либо обработка выполняется перед
   // выводом верхнего колонтитула
   $status = '';

   // запросы на регистрацию или на выход из системы должны обрабатываться
   // прежде любых других
   if($email&&$password)
   {
      $login = login($email, $password);

      if($login = 'admin')
      {
         $status .= "<p><b>".get_real_name($email) . "</b> logged in"
                    ." successfully as <b>Administrator</b><br><br><br><br><br>";
         $admin_user = $email;
         session_register("admin_user");
      }
      else if($login == 'normal')
      {
         $status .= "<p><b>".get_real_name($email) . "</b> logged in"
                    ."successfully.<br><br>";
         $normal_user = $email;
         session_register("normal_user");
      }
      else
      {
         $status .= "<p>Sorry, we could not log you in with that
                     email address and password.<br>";
      }
   )

   if($action == 'log-out')
   {
      session_destroy();
      unset($action);
      unset($normal_user);
      unset($admin_user);
   }

/*********************************************************************
* Раздел 2: создание и отображение верхних колонтитулов
*********************************************************************/

   // определение кнопок, которые будут отображаться в панели инструментов
   if(check_normal_user())
   {
      // если пользователь является обычным
      $buttons[0] = 'change-password';
      $buttons[1] = 'account-settings';
      $buttons[2] = 'show-my-lists';
      $buttons[3] = 'show-other-lists';
      $buttons[4] = 'log-out';
   }
   else if(check_admin_user())
```

```
{
   // если пользователь является администратором
   $buttons[0] = 'change-password';
   $buttons[1] = 'create-list';
   $buttons[2] = 'create-mail';
   $buttons[3] = 'view-mail';
   $buttons[4] = 'log-out';
   $buttons[5] = 'show-all-lists';
   $buttons[6] = 'show-my-lists';
   $buttons[7] = 'show-other-lists';
}
else
{
   // если пользователь вообще не зарегистрирован
   $buttons[0] = 'new-account';
   $buttons[1] = 'show-all-lists';
   $buttons[4] = 'log-in';
)
if($action)
<
   // отображение верхнего колонтитула с названием приложения и описанием страницы
   // или действия
   do_html_header("Pyramid-MLM - ".
                  format_action($action));
}
else
{
  //отображение верхнего колонтитула, в котором отображается только название приложения
   do_html_header("Pyramid-MLM");
}
display_toolbar($buttons);

// отображение любого текста, сгенерированного функциями, вызванными
// перед отображением верхнего колонтитула
echo $status;

/**************************************************************************
* Раздел 3: выполнение действий
**************************************************************************/
// незарегистрированный пользователь может выполнять только эти действия
switch ( $action )
{
   case 'new-account' :
   <
      unset($normal_user);
      unset($admin_user);
      display_account_form($normal_user, $admin_user);
      break;
   }
   case 'store-account' :
   <
      if (store_account($normal_user, $admin_user, $HTTP_POST_VARS))
         $action = '';
      if(!check_logged_in())
         display_login_form ($action);
      break;
   }
   case 'log-in' :
   case '' :
   {
      if(!check_logged_in())
         display_login_form($action);
      break;
   }
   case 'show-all-lists' :
   {
      display_items("All Lists", get_all_lists(), 'information',
                    'show-archive','');
      break;
   }
```

```
    case 'show-archive' :
    {
      display_items("Archive For ".get_list_name($id),
                    get_archive($id), 'view-html', 'view-text', '');
      break;
    }
    case 'information' :
    {
      display_information($id);
      break;
    }

>                                   \
//для выполнения всех остальных действий пользователь должен быть зарегистрированным
if(check_logged_in())
{
   switch ( $action )
   {
      case 'account-settings' :
      {
        display_account_form($normal_user, $admin_user, get_email(),
              get_real_name(get_email()) , get_mimetype(get_email()));
        break;
      }
      case 'show-other-lists' :
      {
        display_items("Unsubscribed Lists",
                      get_unsubscribed_lists(get_email()) , 'information',
                      'show-archive', 'subscribe');
        break;
      }
      case 'subscribe' :
      {
        subscribe(get_email(), $id) ;
        display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                      'information', 'show-archive', 'unsubscribe');
        break;
      }
      case 'unsubscribe' :
      {
        unsubscribe(get_email(), $id);
        display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                      'information', 'show-archive', 'unsubscribe');
        break;
      }
      case '':
      case 'show-my-lists' :
      {
        display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                      'information', 'show-archive', 'unsubscribe');
        break;
      }
      case 'change-password' :
      {
        display_password_form();
        break;
      }
      case 'store-change-password' :
      {
        if(change_password(get_email(), $old_passwd,
            $new_passwd, $new_passwd2))
        {
          echo "<p>OK: Password changed.<br><br><br><br><br><br>";
        }
        else
        {
          echo "<p>Sorry, your password could not be changed.";
          display_password_form();
        }
```

```
            break;
        }
    }
}
// Следующие действия могут выполняться только пользователем,
// имеющим права администратора
if(check_admin_user())
{
    switch ( $action )
    {
        case 'create-mail' :
        {
            display_mail_form(get_email());
            break;
        }
        case 'create-list' :
        {
            display_list_form(get_email());
            break;
        }
        case 'store-list' :
        {
            if(store_list($admin_user, $HTTP_POST_VARS))
            {
                echo "<p>New list added<br>";
                display_items("All Lists", get_all_lists(), 'information',
                              'show-archive','');
            }
            else
                echo "<p>List could not be stored, please try "
                    . "again . <br><br><br><br><br>"

            break;
        }
        case 'send' :
        {
            send($id, $admin_user);
            break;
        }
        case 'view-mail' :
        {
            display_items("UnsentMail",get_unsent_mail(get_email()),
                          'preview-html','preview-text','send');
            break;
        }
    }
}

/*************************************************************************
* Раздел 4: отображение нижнего колонтитула
*************************************************************************/

  do_html_footer();
?>
```

В этом листинге помечены четыре раздела кода.

На этапе предварительной обработки создается сеанс и выполняются действия, которые должны быть выполнены до того, как можно будет отправить верхний колонтитул. В данном случае в их число входят регистрация и выход из системы.

На этапе обработки верхнего колонтитула создаются кнопки меню, которые увидит пользователь, и с помощью функции **do_html_header()** из файла **output_fns.php** отображаются соответствующие верхние колонтитулы. Эта функция всего лишь отображает строку верхнего колонтитула и меню, поэтому мы не будем подробно ее рассматривать.

В основной части сценария программа отвечает на выбранные пользователем действия. Эти действия разделены на три поднабора: действия, которые могут выполняться, если пользователь не зарегистрирован, действия, которые могут выполняться обычными пользователями, и действия, которые могут выполняться пользователями, имеющими права администратора. Проверка наличия доступа к последним двум наборам действий выполняется с помощью функций **check_logged_in() и check_admin_user()**. Эти функции размещаются в библиотеке функций **user_fns.php**. Их код и код функции **check_normal_user()** приведен в листинге 28.3.

Листинг 28.3 Функции из файла user_auth_fns.php — эти функции проверяют, зарегистрирован ли пользователь, и если да — то на каком уровне.

```
function check_normal_user()
// проверка, зарегистрирован ли пользователь,
// и уведомление его, если это не так
{
  global $normal_user;

  if (session_is_registered("normal_user"))
    return true;
  else
    return false;
}
function check_admin_user()
// проверка, зарегистрирован ли пользователь,
// и уведомление его, если это не так
{
  global $admin_user;

  if (session_is_registered("admin_user"))
    return true;
  else
    return false;
}
function check_logged_in()
{
  return ( check_normal_user() || check_admin_user() );
}
```

Как видите, для проверки, зарегистрирован ли пользователь, в этих функциях используются переменные сеанса **$normal_user и $admin_user**. Установка этих переменных сеанса будет рассмотрена несколько позже.

В заключительном разделе сценария мы отправляем нижний колонтитул HTML, используя для этого функцию **do_html_footer()** из файла **output fns.php**.

Давайте кратко рассмотрим возможные действия в системе. Они перечислены в табл. 28.2.

Следует отметить, что действие **store-mail,** которое фактически загружает информационные бюллетени, введенные администраторами при помощи действия **create-mail**, является исключением в этой таблице. Это единственное действие, которое в действительности реализовано в другом файле — **upload.php**. Мы поместили его в другой файл, поскольку это несколько упрощает решение проблем безопасности.

Мы рассмотрим реализацию этих действий в соответствии с тремя группами, перечисленными в табл. 28.2, т.е. действий, выполняемых незарегистрированными пользователями; действий, выполняемых зарегистрированными пользователями; действий, выполняемых администраторами.

Таблица 28.2 Возможные действия в приложении менеджера списков рассылки

Действие	*Кто-то может выполнять*	*Описание*
log-in	Любой пользователь	Отображает пользователю форму регистрации
log-out	Любой пользователь	Завершает сеанс
new-account	Любой пользователь	Создает новую учетную запись пользователя
store-account	Любой пользователь	Сохраняет подробную информацию учетной записи
show-all-lists	Любой пользователь	Отображает список доступных списков рассылки
show-archive	Любой пользователь	Отображает заархивированные информационные бюллетени для конкретного списка рассылки
information	Любой пользователь	Отображает основные сведения о конкретном списке рассылки
account-settings	Зарегистрированный пользователь	Отображает параметры настройки учетной записи пользователя
show-other-lists	Зарегистрированный пользователь	Отображает списки рассылки, на которые пользователь не подписался
show-my-lists	Зарегистрированный пользователь	Отображает списки рассылки, на которые пользователь подписался
subscribe	Зарегистрированный пользователь	Выполняет подписку пользователя на конкретный список рассылки
unsubscribe	Зарегистрированный пользователь	Отменяет подписку пользователя на конкретный список рассылки
change-password	Зарегистрированный пользователь	Отображает форму для изменения пароля
store-change-password	Зарегистрированный пользователь	Обновляет пароль пользователя в базе данных паролей
create-mail	Администратор	Отображает форму для загрузки информационных бюллетеней
create-list	Администратор	Отображает форму для создания новых списков рассылки
store-list	Администратор	Сохраняет сведения о списке рассылки в базе данных
view-mail	Администратор	Отображает информационные бюллетени, которые были загружены, но еще не отправлены
send	Администратор	Отправляет информационные бюллетени подписчикам

Реализация регистрации

Желательно, чтобы новые пользователи, посещающие сайт, выполнили три действия. Во-первых, взглянули на то, что им предлагается; во-вторых, получили учетную запись; в-третьих, зарегистрировались. Давайте по очереди рассмотрим эти три действия.

На рис. 28.4 показано окно, отображаемое при первом посещении сайта.

Создание учетной записи и регистрация будут рассмотрены в этом разделе, а к просмотру сведений о списке рассылке мы вернемся в разделах "Реализация функций пользователя" и "Реализация функций администратора".

Создание новой *учетной* записи

Если пользователь выбирает пункт меню New Account (Новая учетная запись), это активизирует действие **new-account.** В результате активизируется следующий фрагмент кода в файле **index.php:**

```
case 'new-account' :
{
   unset($normal_user);
   unset($admin_user);
   display_account_form($normal_user, $admin_user);
   break;
}
```

По существу, этот фрагмент кода осуществляет выход пользователя из системы, если он зарегистрирован, и отображает форму сведений об учетной записи, показанную на рис. 28.5.

Эта форма генерируется функцией **display_account_form()** из библиотеки **output_fns.php.** Эта функция используется как в данном действии, так и в действии по настройке параметров учетной записи для отображения формы, в которой пользователь может настроить учетную запись. Если функция вызывается из действия настройки учетной записи, форма будет заполняться существующими данными учетной записи пользователя. В данном случае форма пуста и готова для ввода сведений о новой учетной записи. Поскольку эта функция всего лишь выводит HTML-код, мы не будем подробно ее рассматривать.

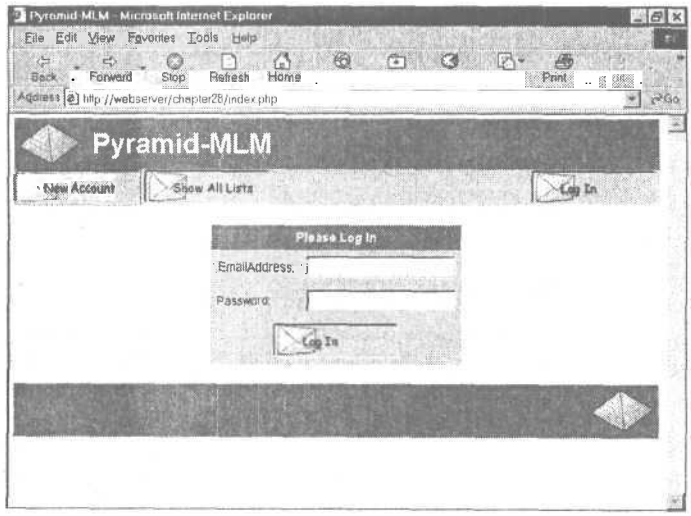


РИСУНОК 28.4
При первом посещении пользователи могут создать новую учетную запись, просмотреть доступные списки рассылки или просто зарегистрироваться.

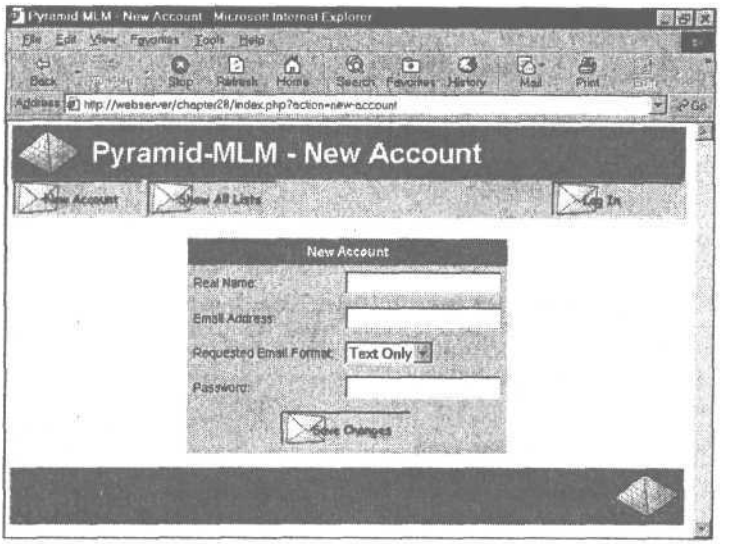


РИСУНОК 28.5
Форма создания новой учетной записи позволяет пользователям вводить сведения о себе.

Кнопка submit (отправить) этой формы вызывает действие **store-account,** которое реализуется следующим фрагментом кода:

```
case 'store-account' :
{
   if (store_account($normal_user, $admin_user, $HTTP_POST_VARS))
     $action = ' ';
   if(!check_logged_in())
     display_login_form($action);
   break;
}
```

Функция **store_account()** записывает сведения об учетной записи в базу данных. Код этой функции приведен в листинге 28.4.

Листинг 28.4 Функция store_account() из файла mlm_fns.php — Эта функция проверяет, зарегистрирован ли пользователь, и если да. то на каком уровне

```
// добавляет нового подписчика в базу данных или дает возможность
// пользователю изменить данные о себе
function store_account($normal_user, $admin_user, «details)
{
   if(!filled_out($details))
   {
      echo "All fields must be filled in.  Try again.<br><br>";
      return false;
   }
   else
   {
      if(subscriber_exists($details['email']))
      {
         //проверка наличия регистрации пользователя,
         //сведения о котором предпринимается попытка изменить
         if(get_email()==$details['email'])
         {
            «query = "update subscribers set realname = '$details[realname]',
                                             mimetype = '$details[mimetype]'
                      where email = '" . «details[email] . "'";
```

```
            if(db_connect() && mysql_query($query))
            {
               return true;
            }
            else
            {
               echo "could not store changes.<br><br><br><br><br><br>";
               return false;
            }
         }
         else
         {
            echo "<p>Sorry, that email address is already registered here.";
            echo "<p>You will need to log in with that address to change "
                 ." Web settings.";
            return false;
         }
      }
      else // новая учетная запись
      {
         $query = "insert into subscribers
                              values ('$details[email]',
                               '$details[realname]',
                               '$details[mimetype]',
                                password('$details[new_password]'),
                                                              0)";
         if(db_connect() Si mysql_query($query))
         {
            return true;
         }
         else
         {
            echo "Could not store new account.<br><br><br><br><br><br>";
            return false;
         }
      }
   }
}
```

Вначале эта функция проверяет, внес ли пользователь требуемые сведения.

Если это так, функция либо создает нового пользователя, либо обновляет сведения об учетной записи существующего пользователя. Пользователь может обновить сведения об учетной записи только того пользователя, в качестве которого он зарегистрирован.

Эта проверка выполняется с помощью функции **get_email()**, которая получает адрес электронной почты текущего зарегистрированного пользователя. Мы вернемся к этой функции позже, поскольку в ней задействованы переменные сеанса, которые определяются при регистрации пользователя.

Регистрация

Если пользователь заполнит регистрационную форму, показанную на рис. 28.4, и щелкнет на кнопке Log In (Вход), запускается сценарий **index.php** с установленными значениями переменных **$email** и **$password**. В результате активизируется код регистрации, касающийся стадии предварительной обработки сценария, как показано в следующем листинге:

Листинг 28.4 (продолжение)

```
// запросы на регистрацию или выход из системы должны быть
// обработаны до выполнения любых других действий
  if ($email&&$password)
  {
    $login = login ($email, $password) ;

    if($login == 'admin')
    {
      $status .= "<p><b>".get_real_name($email) . "</b> logged in"
              ." successfully as <b>Administrator</b><br><br><br><br><br>";
      $admin_user = $email;
      session_register ("admin_user") ;
    }
    else if($login == 'normal')
    {
      $status .= "<p><b>".get_real_name($email) . "</b> logged in"
                 ." successfully.<br><br>";
      $normal_user = $email;
      session_register("normal_user");
    }
    else
    {
      $status .= "<p>Sorry, we could not log you in with that
                  email address and password.<br>";
    }
  }
```

Как видите, вначале предпринимается попытка выполнить регистрацию с помощью функции **login()** из библиотеки **user_auth_fns.php.** Эта функция несколько отличается от функций регистрации, использованных в других примерах, поэтому давайте ее рассмотрим. Ее код приведен в листинге 28.5.

Листинг 28.5 Функция login() из библиотеки **user_auth_fns.php** — Эта функция проверяет сведения о регистрации пользователя _

```
function login($email, $password)
// сравнивает имя пользователя и пароль с данными,
// хранящимися в базе данных
// если они совпадают, возвращается тип регистрации
// в противном случае возвращается значение false
{
  // соединение с БД
  $conn = db_connect() ;
  if ( ! $conn)
    return 0 ;
  $query = "select admin from subscribers
                              where email='$email'
                              and password = password('$password')";
  //отображение на экране содержимого переменной $query
  $result =mysql_query($query) ;
  if  (!$result)
    return false;
  if  (mysql_num_rows($result)<1)
    return false;
  if (mysql_result($result, 0, 0)  == 1)
    return 'admin';
  else
    return 'normal';
}
```

Ранее задействованные функции регистрации возвращали значение **true** в случае успешной регистрации и **false** в случае неудачи. В данном случае функция также возвращает значение **false** в случае неудачи, но в случае успешной регистрации она возвращает тип пользователя: **'admin' или 'normal'**. Проверка типа пользователя выполняется путем получения значения, хранящегося в столбце администратора таблицы подписчиков и соответствующего конкретной комбинации адреса электронной почты и пароля. Если никакое значение не возвращается, функция возвращает **false.** Если пользователь является администратором, этим значением будет 1 **(true)** и функция вернет значение **'admin'**. В противном случае она вернет значение **'normal'**.

Возвращаясь к основной процедуре выполнения, мы регистрируем переменную сеанса для отслеживания того, кем является данный пользователь. Этой переменной будет либо **$admin_user,** если пользователь является администратором, либо **$normal_user**, если это обычный пользователь. Какая бы из этих переменных ни была определена, она будет содержать адрес электронной почты пользователя. С целью упрощения проверки адреса электронной почты пользователя используется ранее упоминавшаяся функция **get_email()**. Она приведена в листинге 28.6.

Листинг 28.6 Функция get_email из библиотеки mlmjns.php — Эта функция возвращает адрес электронной почты зарегистрированного пользователя

```
function get_email()
{
   global $normal_user;
   global $admin_user;

   if (session_is_registered("normal_user"))
     return $normal_user;
   if (session_is_registered("admin_user"))
    return $admin_user;

   return false;
}
```

После возврата к основной программе система со'общает пользователю, был ли он зарегистрирован, и если да, то на каком уровне.

Результат выполнения попытки регистрации приведен на рис. 28.6.

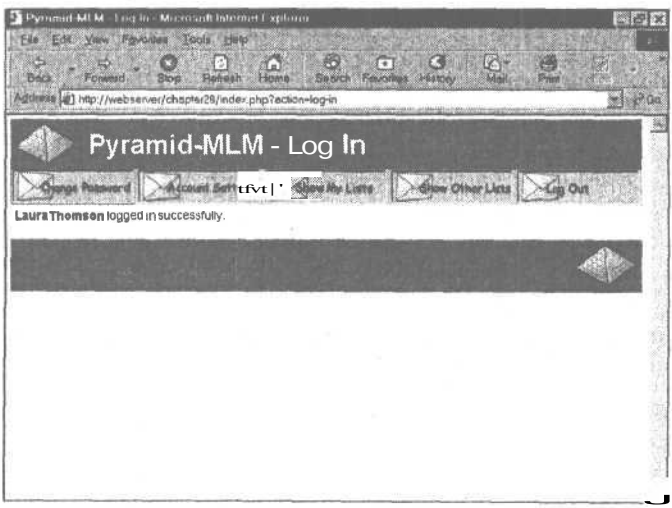


РИСУНОК 28.6
Система сообщает пользователю, что регистрация была выполнена успешно.

Теперь, после выполнения регистрации в качестве пользователя, можно переходить к реализации функций пользователя.

Реализация функций пользователя

Необходимо, чтобы после регистрации пользователи могли выполнять пять действий:

- Просматривать списки рассылки, доступные для подписки
- Подписываться на списки рассылки и отменять подписку на них
- Изменять настройки своих учетных записей
- Изменять свои пароли
- Выходить из системы

Большинство из этих пунктов меню показано на рис. 28.6. Ниже мы рассмотрим реализацию каждого из них.

Просмотр списков рассылки

Мы реализуем ряд пунктов меню для просмотра доступных списков рассылки и сведений о них. На рис. 28.6 показаны два из них: Show My Lists (Показать мои списки рассылки), который будет получать списки, на которые подписался данный пользователь, и Show Other Lists (Показать другие списки рассылки), который будет получать списки, на которые пользователь не подписался.

Если вы снова взглянете на рис. 28.4, то увидите, что существует еще один пункт — Show All Lists (Показать все списки рассылки), который будет отображать все доступные в системе списки рассылки. Чтобы система была действительно пригодной к использованию, ее следовало бы дополнить функцией разбиения на страницы (для отображения, скажем, 10 списков на каждой странице). Для краткости мы не стали этого делать.

Три упомянутых пункта меню активизируют, соответственно, действия **show-my-lists**, **show-other-lists** и show-all-lists. Несложно догадаться, что все эти действия работают весьма похоже. Их код имеет следующий вид:

```
case 'show-all-lists' :
<
  display_items("All Lists", get_all_lists(), 'information',
                'show-archive','');
  break;
}
case 'show-other-lists' :
{
    display_items("Unsubscribed Lists",
                  get_unsubscribed_lists(get_email()), 'information',
                  'show-archive', 'subscribe');
    break;
}
case '':
case 'show-my-lists' :
{
    display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                  'information', 'show-archive', 'unsubscribe');
    break;
}
```

Как видите, все эти действия вызывают функцию **display_items()** из библиотеки **output_fns.php,** но каждое из них вызывает ее с другими параметрами. Они использу-

ют также упоминавшуюся ранее функцию **get_email()** для получения соответствующего адреса электронной почты для данного пользователя. Результат выполнения этой функции показан на рис. 28.7.

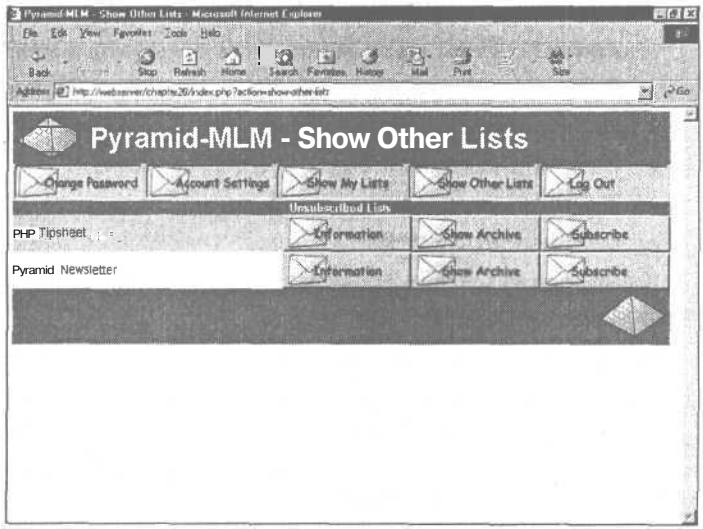


РИСУНОК 28.7
Функция display_items() используется для отображения списка списков рассылки, на которые пользователь не подписался.

На этом рисунке показана страница Show Other Lists (Показать другие списки рассылки).

Давайте рассмотрим код функции **display_items(),** который приведен в листинге 28.6

Листинг 28.6 функция display_items() из библиотеки output_fns.php — Эта функция используется для отображения списка элементов и связанных с ними действий

```
function display_items($title, $list, $action1='', $action2='', $action3='')
{
  global $table_width;
  echo "<table width = $table_width cellspacing = 0 cellpadding = 0
        border = 0>";

  // подсчет количества действий
  $actions = (($action1!='') + ($action2!='') + ($action3!=''));

  echo "<tr>
       <th colspan = ". (1+$actions) ." bgcolor='#5B69A6'> $title </th>
       </tr>";

  // подсчет количества элементов
  $items = sizeof($list);

  if($items == 0)
      echo "<tr>
           <td colspan = ". (1+$actions)."align = center>No Items to
    Display</td>
           </tr>";
    else
    {
      // печать каждой строки
      for($i = 0; $items; $i++)
      {
        if($i%2) // создание кнопок для действий
              // (до трех действий на строку)
          $bgcolor = "'#ffffff'";
```

```
        else
           $bgcolor = "'#ccccff'";
        echo "<tr
                <td bgcolor = $bgcolor
                 width = ". ($table_width - ($actions*149)) .">";
        echo $list[$i][1];
        if($list[$i][2])
          echo " - ".$list[$i][2];
        echo "</td>";

        // создание кнопок для максимум трех действий на строку
        for($j = 1; $j<=3; $j++)
        {
          $var = 'action'.$j;
          if($$var)
          {
            echo "<td bgcolor = $bgcolor width = 149>";
            // кнопки просмотра/предварительного просмотра составляют
            // особый случай, поскольку они связаны с файлом
            if($$var == 'preview-html'| |$$var== 'view-html' | |
                $$var == 'preview-text'| | $$var == 'view-text')
              display_preview button($list[$i][3], $list[$i][0], $$var);
            else
              display_button( $$var, "&id=" . $list[$i][0] );
            echo "</td>";
          }
        }
        echo "</tr>\ n";
      }
      echo "</table>";
    }
}
```

Эта функция будет выводить таблицу элементов, каждый из которых имеет до трех связанных с ним функциональных кнопок. Функция принимает следующие пять параметров:

- **$title** — заголовок, который отображается в верхней части таблицы; в данном случае, изображенном на рис. 28.7, в качестве заголовка передается "Unsubscribed Lists" ("Неподписанные списки рассылки"), как было показано в ранее рассмотренном фрагменте кода для действия "Show Other Lists".
- **$list** — массив элементов, которые должны отображаться в каждой строке таблицы. В данном случае это массив списков рассылки, на которые пользователь не подписан в текущий момент. Этот массив создается (в данном случае) в функции **get_unsubscribed_lists()**, которая будет рассмотрена несколько позже. Это многомерный массив, в котором каждая строка содержит до четырех элементов данных о каждой из строк. Эти элементы таковы:
  - Элемент **$list[*n*][0]** должен содержать идентификатор элемента, которым обычно будет номер строки. Этот элемент присваивает функциональным кнопкам идентификатор строки, применительно к которой они должны действовать. В данном случае будут использоваться идентификаторы из базы данных — об этом будет сказано чуть позже.
  - Элемент **$list[*n*][1]** должен содержать имя элемента. Им будет текст, отображаемый для конкретного элемента. Например, в случае, показанном на рис. 28.7, именем элемента в первой строке таблицы является PHP Tipsheet.

- Элементы **$list[n][2]** и **$list[n][3]** необязательны. Они используются для того, чтобы указать на присутствие дополнительной информации, и соответствуют дополнительному информационному тексту и идентификатору дополнительной информации. Пример использования этих двух параметров будет приведен при рассмотрении действия View Mail (Просмотр сообщений электронной почты) в разделе "Реализация функций администратора".

В качестве индексов массива можно было бы использовать также ключевые слова.

Третий, четвертый и пятый параметры функции используются для передачи трех действий, которые будут отображаться на кнопках, соответствующих каждому элементу. На рис. 28.7 ими являются три функциональных кнопки: Information (Информация), Show Archive (Показать архив) и Subscribe (Подписаться).

Эти три кнопки для страницы Show All Lists были получены за счет передачи имен действий — **information, show-archive и subscribe.** При помощи функции **display_button()** упомянутые действия были преобразованы в кнопки с отображаемыми на них словами и с соответствующими действиями.

Как видно из соответствующих действий, каждое действие **Show** вызывает функцию **dispay_items()** по-своему. Помимо того, что каждое из них имеет свой заголовок и выполняет свое действие, оно же использует собственную функцию для построения массива отображаемых элементов. Действие **Show All Lists** использует функцию **get_all_lists()**, **Show Other Lists** — функцию **get_unsubscribed_lists()**, **a Show My Lists** — функцию **get_subscribed_lists()**. Все эти функции работают аналогично. Они входят в состав библиотеки функций **mlm_fns.php**.

Мы рассмотрим функцию **get_unsubscribed_lists()**, поскольку именно этот пример рассматривался до сих пор. Ее код приведен в листинге 28.7.

Листинг 28.7 Функция get_unsubscribed_lists() из библиотеки mlm_fns.php — Эта функция используется для создания массива списков рассылки, на которые пользователь не подписался

```
function get_unsubscribed_lists ($email)
{
   $list = array () ;

   $query = "select lists.listid, listname,
            email from lists left join sub_lists
            on lists.listid = sub_lists.listid
            and email='$email' where email is NULL order by listname";
   if (db_connect ())
   {
      $result = mysql_query ($query);
      if (!$result)
         echo "<p>Unable to get list from database.";
      $num = mysql_numrows ($result);
      for($i = 0; $i<$num; $i++)
      {
         array_push($list, array(mysql_result($result, $i, 0),
                     mysql_result($result, $i, 1)));
      }
   }
   return $list;
}
```

Как видите, в эту функцию необходимо передать адрес электронной почты. Им должен быть адрес электронной почты подписчика, с которым выполняется работа.

Функции **get_subscribed_lists()** также требуется передача адреса электронной почты в качестве параметра, но, по очевидным причинам, для функции **get_all_lists()** этого не требуется.

Передавая адрес электронной почты подписчика, мы подключаемся к базе данных и получаем все списки рассылки, на которые подписчик не подписался. Как обычно, в случае подобного запроса в среде MySQL для отыскания несовпадающих элементов применяется **LEFT JOIN.**

Циклический просмотр результатов и построчное построение массива выполняется с использованием встроенной функции **array_push()**.

Теперь, когда уже известно, как создавать этот список, давайте рассмотрим связанные с этими действиями функциональные кнопки.

Просмотр сведений о списке рассылки

Кнопка Information (Информация), показанная на рис. 28.7, запускает действие **'information'**, которое имеет следующий код:

```
case 'information' :
     {
       display_information($id);
       break;
     }
```

Результат выполнения функции **display_information()** показан на рис. 28.8.

Функция отображает некоторые базовые сведения о конкретном списке рассылки, а также количество подписчиков и количество отправленных в список и доступных в архиве информационных бюллетеней (подробнее об этом — немного позже).

Код этой функции приведен в листинге 28.8.

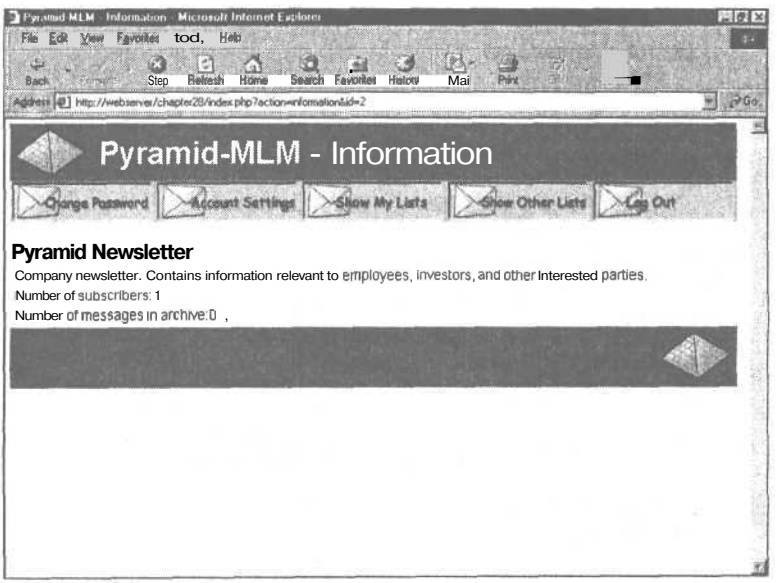


РИСУНОК 28.8
Функция display_information() отображает сведения о списке рассылки.

Листинг 28.8 Функция display_information() из библиотеки **output_fns.php** — Эта функция отображает сведения о списке рассылки

```
function display_information($listid)
{
   if(!$listid)
      return false;

   $info = load_list_info($listid);

   if($info)
   {
      echo"<h2>".pretty($info[listname]) . "</h2>";
      echo'<p>'.pretty($info[blurb]);
      echo '<p>Number of subscribers: ' . $info[subscribers];
      echo '<p>Number of messages in archive: ' . $info[archive];
   }
}
```

Для выполнения своей связанной с Web задачи функция **display _information()** использует две другие функции: **load_list_info() и pretty()**. Функция **load_list_info()** действительно получает данные из базы данных. Функция **pretty()** просто форматирует эти данные, удаляя из них символы косой черты, преобразуя символы новой строки в разрывы строк HTML и т.п.

Давайте бегло рассмотрим функцию **load_list_info().** Она входит в состав библиотеки функций **mlm_fns.php**, а ее код представлен в листинге 28.9.

Листинг 28.9 Функция **load_list_info()** из библиотеки **mlm_fns.php** — Эта функция создает массив сведений о списке рассылки _

```
function load_list info($listid)
{
   if(!$listid)
      return false;

   if (!db_connect())
      return false;

   $query = "select listname, blurb from lists where listid = $listid";
   $result = mysql_query($query);
   if(!$result)
   {
      echo "Cannot retrieve this list";
      return false;
   }
   $info = mysql_fetch_array($result);

   $query = "select count(*) from sub_lists where listid = $listid";
   $result = mysql_query($query);
   if ($result)
   {
      $info['subscribers'] = mysql_result($result, 0, 0);
   }
   $query = "select count(*) from mail where listid = $listid
             and status = 'SENT'";
   $result = mysql_query($query);
   if ($result)
   {
      $info['archive'] = mysql_result($result, 0, 0);
   }
   return $info;
}
```

Функция выполняет три запроса в базу данных для формирования имени и информационной строки для списка рассылки из таблицы **lists,** количества подписчиков из таблицы **sub_lists** и количества отправленных информационных бюллетеней из таблицы **mail.**

Просмотр архивов списков рассылки

Кроме просмотра информационной строки списка рассылки, пользователи могут, щелкнув на кнопке Show Archive, просмотреть все сообщения электронной почты, которые были отправлены в список рассылки. В результате активизируется действие show-archive, которое приводит к запуску следующего кода:

```
case 'show-archive' :
{
  display_items("Archive For " .get_list_name($id),
                get_archive($id), 'view-html', 'view-text', ' ');
  break;
}
```

Эта функция также использует функцию **display_items()** для вывода списка различных элементов сообщений электронной почты, отправленных в список рассылки. Получение этих элементов выполняется с помощью функции **get_archive()** из библиотеки **mlm_fns.php**. Ее **код** приведен в листинге 28.10.

Листинг 28.10 Функция get_archive() из библиотеки mlm_fns.php — Эта функция создает массив заархивированных информационных бюллетеней для данного списка рассылки

```
function get_archive($listid)
{
  //возвращает массив заархивированных сообщений
  //электронной почты для этого списка
  //массив содержит строки типа (идентификатор_строки, тема)

  $list = array();
  $listname = get_list_name($listid);

  $query = "select mailid, subject, listid from mail
            where listid = $listid and status = 'SENT' order by sent";

  if(db_connect())
  {
    $result = mysql_query($query);
    if(!$result)
    {
      echo "<p>Unable to get list from database - $query.";
      return false;
    }
    $num = mysql_numrows($result);
    for($i = 0; $i<$num; $i++)
    {
      $row = array(mysql_result($result, $i, 0),
                   mysql_result($result, $i, 1), $listname, $listid);
      array_push($list, $row);
    }
  }
  return $list;
}
```

Как и ранее, эта функция получает из базы данных требуемую информацию — в данном случае сведения о сообщениях электронной почты, которые были отправлены - и создает массив, подходящий для передачи в **display_items()**.

Подписка и отмена подписки

В перечне списков рассылки, показанном на рис. 28.7, каждый список рассылки имеет кнопку, которая дает возможность пользователям подписаться на него. Аналогично, если пользователи выбирают пункт меню Show My Lists для просмотра списков, на которые они уже подписались, рядом с каждым списком отображается кнопка Unsubscribe (Отменить подписку).

Эти кнопки активизируют действия subscribe (подписаться) и unsubscribe (отменить подписку), которые запускают, соответственно, следующие два фрагмента кода:

```
case 'subscribe' :
{
   subscribe(get_email(), $id);
   display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                 'information', 'show-archive', 'unsubscribe');
   break;
}
case 'unsubscribe' :
{
   unsubscribe(get_email(), $id);
   display_items("Subscribed Lists", get_subscribed_lists(get_email()),
                 'information', 'show-archive', 'unsubscribe');
   break;
}
```

В любом случае вызывается функция (**subscribe()** или **unsubscribe()**), а затем с помощью функции **display_items()** снова отображается перечень списков рассылки, на которые пользователь подписан в текущий момент.

Функции **subscribe()** и **unsubscribe()** приведены в листинге 28.11.

Листинг 28.11 Функции subscribe() и unsubscribe() из библиотеки mlm_fns.php — Эти функции добавляют и удаляют подписанные списки рассылки для данного пользователя

```
function subscribe($email, $listid)
{
   if(!$email||!$listid||!list_exists($listid)||!subscriber_exists($email))
      return false;

   //если на этот список рассылки уже имеется подписка,
   //осуществляется выход из функции
   if(subscribed($email, $listid))
      return false;

   if(!db_connect())
      return false;

   $query = "insert into sub_lists values ('$email', $listid)";

   $result = mysql_query($query);
           return $result;
}
function unsubscribe($email, $listid)
{
   if(!$email||!$listid)
      return false;

   if(!db_connect())
      return false;
```

```
    $query = "delete from sub_lists where email = '$email' and listid =
     $listid";
    $result = mysql_query($query);
    return $result;
}
```

Функция **subscribe()** добавляет в таблицу **sub_lists** строку, соответствующую подписке; функция **unsubscribe()** удаляет эту строку.

Изменение параметров настройки учетной записи

Щелчок на кнопке Account Settings (Параметры настройки учетной записи) активизирует действие account-settings. Оно имеет следующий код:

```
case 'account-settings' :
{
   display_account_form($normal_user, $admin_user, get_email(),
          get_real name(get_email()), get_mimetype(get_email()));
   break;
}
```

Как видите, опять-таки используется функция **display_account_form()**, которая применялась для первоначального создания учетной записи. Однако на сей раз ей передаются текущие сведения о пользователе, которые с целью упрощения внесения изменений будут отображаться в форме. Как уже отмечалось ранее, когда пользователь щелкает на кнопке представления этой формы, активизируется действие **store-account.**

Изменение паролей

Щелчок на кнопке Change Password (Изменить пароль) активизирует действие **change-password,** которое приводит к запуску следующего кода:

```
case 'change-password' :
{
   display_password_form();
   break;
}
```

Функция **display_password_form()** (из библиотеки **output_fns.php**) просто отображает форму, в которой пользователь может изменить свой пароль. Эта форма показана на рис. 28.9.

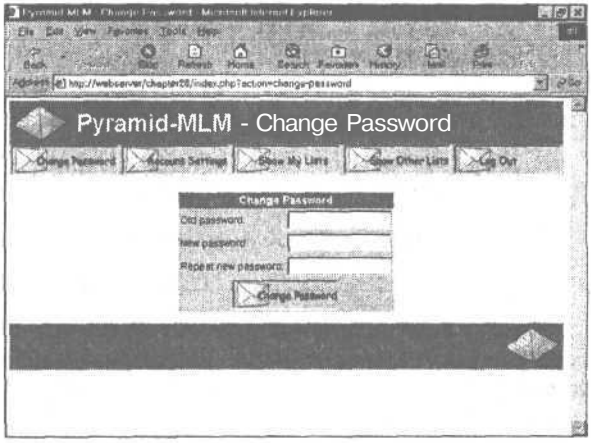


РИСУНОК 28.9
Функция *display_password_form() дает возможность пользователям изменять свои пароли.*

Когда пользователь щелкает на кнопке Change Password в нижней части этой формы, активизируется действие **store-change-password.** Оно имеет следующий код:

```
case 'store-change-password' :
{
  if(change_password(get_email(), $old_passwd,
     $new_passwd, $new_passwd2))
  {
    echo "<p>OK: Password changed.<br><br><br><br><br><br>";
  }
  else
  {
    echo "<p>Sorry, your password could not be changed.";
    display_password_form();
  }
  break;
}
```

Как видите, этот код предпринимает попытку с помощью функции **change_password()** изменить пароль и сообщает пользователю об успехе или неудаче выполнения. Функция **change_password()** находится в библиотеке функций **user_auth_fns.php**. Ее код приведен в листинге 28.12.

Листинг 28.12 Функция change_password() из библиотеки user_auth_fns.php — Эта функция проверяет и обновляет пароль пользователя_

```
function change_password($email, $old_password, $new_password,
                         $new_password_conf)
// изменяет старый пароль old_password для электронной
// почты на новый пароль new_password
// возвращает значение true или false
{
  // если старый пароль правилен,
  // изменяет пароль на new_password и возвращает значение true,
  // в противном случае возвращает значение false
  if (login($email, $old_password))
  {
    if($new_password==$new_password_conf)
    {
      if (!($conn= db_connect()))
        return false;
      $query = "update subscribers
                set password = password('$new_password')
                where email = '$email'";
      $result = mysql_query($query);
      return$result;
    }
    else
      echo "<p> Your passwords do not match.";
  }
  else
    echo "<p> Your old password is incorrect.";

  return false; // старый пароль неверен
}
```

Эта функция подобна остальным рассмотренным функциям определения и изменения паролей. Она сравнивает пару вновь введенных пользователем пароля, чтобы убедиться в том, что они совпадают, и если это так, пытается обновить пароль пользователя в базе данных.

Выход из системы

Когда пользователь щелкает на кнопке Log Out (Выйти из системы), запускается действие log-out. Выполняемый этим действием код находится в разделе предварительной обработки сценария и имеет следующий вид:

```
if($action == 'log-out')
{
   session_destroy();
   unset($action);
   unset($normal_user);
   unset($admin_user);
}
```

Этот фрагмент кода освобождает память, выделенную под переменные сеанса, и удаляет сеанс. Обратите внимание, что он освобождает также переменную **$action** — т.е. выполняется вход в оператор основной ветви без какого-либо действия, запускающий следующий код:

```
case '':
{
   if(!check_logged_in())
      display_login_form($action);
   break;
}
```

Это позволит зарегистрироваться другому пользователю или этому же пользователю под другим именем.

Реализация функций администратора

Если какой-либо пользователь регистрируется как администратор, ему доступны дополнительные пункты меню, показанные на рис. 28.10.

Дополнительными пунктами являются Create List (создание нового списка рассылки), Create Mail (создание нового информационного бюллетеня) и View Mail (просмотр и отправка созданных информационных бюллетеней, которые еще не были отправлены). Мы рассмотрим их по очереди.

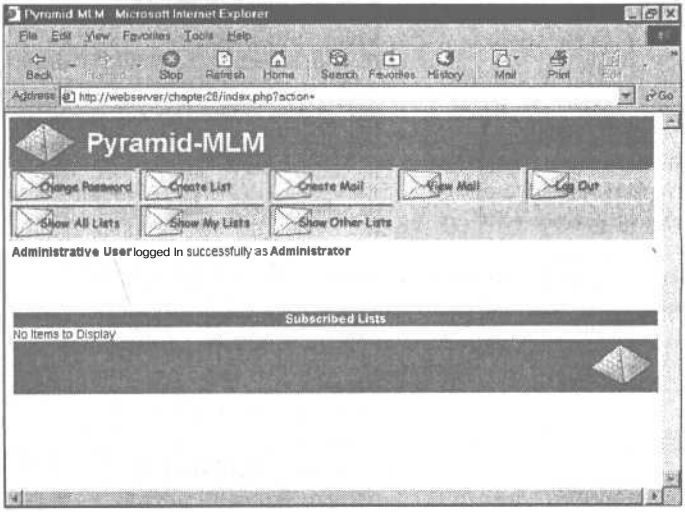


РИСУНОК 28.10
Меню администратора делает возможным создание и поддержку списков рассылки.

Создание нового списка рассылки

Если администратор решает создать новый список рассылки и выполняет щелчок на кнопке Create List (Создать список рассылки), он активизирует действие **create-list,** связанное со следующим кодом:

```
case 'create-list' :
{
  display_list_form(get_email());
  break;
}
```

Функция **display_list_form()** отображает форму, которая позволяет администратору ввести параметры нового списка рассылки. Эта функция находится в библиотеке **output_fns.php**. Она всего лишь выводит HTML-текст, поэтому мы не будем на ней останавливаться. Ее вывод показан на рис. 28.11.

Когда администратор щелкает на кнопке Save List (Сохранить список рассылки), активизируется действие **store-list,** которое запускает следующий код в файле **index.php:**

```
case 'store-list' :
{
  if(store_list($admin_user, $HTTP_POST_VARS))
  {
    echo "<p>New list added<br>";
    display_items("All Lists", get_all_lists(), 'information',
                  'show-archive','');
  }
  else
    echo "<p>List could not be stored, please try "
         ."again.<br><br><br><br><br>";
  break;
}
```

Несложно заметить, что код предпринимает попытку сохранить параметры нового списка рассылки, а затем отобразить новый перечень списков рассылки. Параметры списка рассылки сохраняются функцией **store_list().** Код этой функции показан в листинге 28.13.

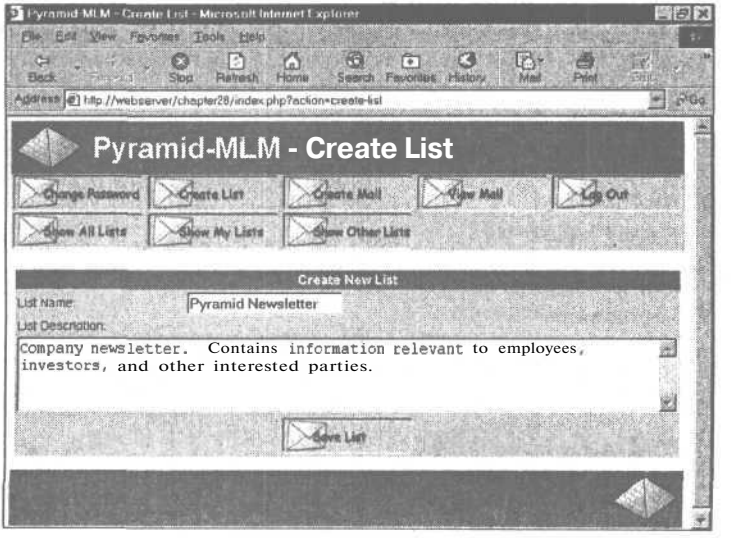


РИСУНОК 28.11
Команда меню Create List требует, чтобы администратор ввел имя и описание (или информационную строку) нового списка рассылки.

Листинг 28.13 Функция strore_list() из библиотеки mlm_fns.php — Эта функция вставляет новый список рассылки в базу данных

```
function store_list($admin_user, $details)
{
  if(!filled_out($details))
  {
    echo "All fields must be filled in.  Try again.<br><br>";
    return false;
  }
  else
  {
    if(!check_admin_user($admin_user))
      return false;
      // как эта функция может вызываться кем-либо,
      // не зарегистрированным в качестве администратора?

    if(!db_connect())
    {
      return false;
    }

    $query = "select count(*) from lists where listname = '$details[name]'";
    $result = mysql_query($query);
    if(mysql_result($result, 0, 0) > 0)
    {
      echo "Sorry, there is already a list with this name.";
      return false;
    }

    $query = "insert into lists values (NULL,
                                        '$details[name]',
                                        '$details[blurb]')";

    $result = mysql_query($query);
    return $result;
  }
}
```

Прежде чем выполнить запись в базу данных, эта функция выполняет несколько проверок: она проверяет ввод всех необходимых данных, является ли текущий пользователь администратором, а также уникальность имени списка рассылки. Если все правильно, список добавляется в таблицу **lists** базы данных.

Загрузка нового информационного бюллетеня

Наконец-то мы добрались до основной задачи этого приложения: загрузки и отправки информационных бюллетеней в списки рассылки.

Когда администратор щелкает на кнопке Create Mail (Создать сообщение электронной почты), активизируется следующее действие **create-mail**:

```
case 'create-mail' :
  display_mail_form(get_email());
  break;
```

Администратор увидит форму, показанную на рис. 28.12.

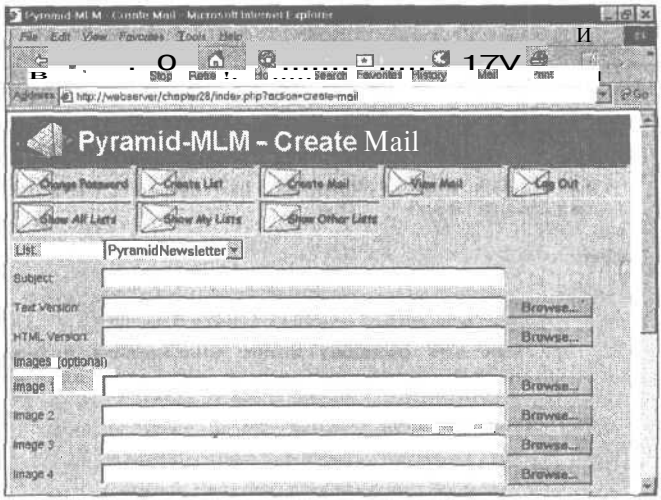


РИСУНОК 28.12
Пункт меню Create Mail (Создать сообщение электронной почты) предоставляет администратору интерфейс для загрузки файлов информационных бюллетеней.

Вспомните, что при разработке этого приложения предполагается, что администратор создал информационный бюллетень в автономном режиме в обоих форматах: HTML и текстовом, и перед отправкой будет загружать обе версии. Этот подход был выбран, дабы при создании информационных бюллетеней администраторы могли применять свои любимые программы. Это делает приложение более удобным.

Как видите, эта форма содержит ряд полей, которые должны заполняться администратором. В верхней части формы располагается выпадающий список списков рассылки, в котором можно выполнять выбор. Администратор должен также заполнить поле темы информационного бюллетеня — это строка Subject (Тема) будущего сообщения электронной почты.

Все остальные поля формы являются полями загрузки файлов, о чем свидетельствуют расположенные рядом с ними кнопки Browse (Обзор). Чтобы отправить информационный бюллетень, администратор должен указать как текстовую, так и HTML-версию этого бюллетеня (хотя, конечно, при необходимости это можно было бы и изменить). Существует также ряд необязательных полей изображения, в которых администратор может загружать любые изображения, внедренные в HTML. Каждый из этих файлов должен указываться и загружаться отдельно.

Отображаемая форма аналогична обычной форме загрузки файлов, за исключением того, что в этом случае она используется для загрузки нескольких файлов. Это обусловливает некоторые небольшие различия в синтаксисе формы и в способе обработки загруженных файлов на другом конце.

Код функции **display_mail_form()** приведен в листинге 28.14.

Листинг 28.14 Функция display_mail_form() из библиотеки output_fns.php — Эта функция отображает форму загрузки файлов

```
function display_mail_form($email, $listid=0)
{
   // отображение html-формы для загрузки нового сообщения
   global $table_width;
   $list = get_all_lists() ;
   $lists = sizeof($list);
?>
```

```
<table cellpadding = 4 cellspacing = 0 border = 0 width =
 <?=$table_width?>>
<form enctype='multipart/form-data' action='upload.php' method='post'>
<tr>
  <td bgcolor = "#cccccc">
    List:
  </td>
  <td bgcolor = "#cccccc">
    <select name = list>
    <?
    for($i = 0; $i<$lists; $i++)
    {
      echo "<option value = ".$list[$i][0];
      if ($listid== $list[$i][0]) echo " selected";
      echo ">".$list[$i][1]."</option>\n";
    }
    ?>
    </select>
  </td>
</tr>
<tr>
  <td bgcolor = "#cccccc">
    Subject:
  </td>
  <td bgcolor = "#cccccc">
    <input type = text name = subject value = "<?=$subject?>"
     size = 60 ></td>
</tr>
<trXtd bgcolor = "#cccccc">
  Text Version:
</td><td bgcolor = "#cccccc">
  <input type=file name='userfile[0]' size = 60>
</tdX/tr>
<trXtd bgcolor = "#cccccc">
  HTML Version:
</tdXtd bgcolor = "#cccccc">
  <input type=file name='userfile[1]' size = 60>
</tdX/tr>
<trXtd bgcolor = "#cccccc" colspan =2>Images: (optional)

<?
  $max_images = 10;
  for($i = 0; $i<10; $i++)
  {
    echo "<tr><td bgcolor = '#cccccc'>Image ". ($i+1) ." </td>";
    echo "<td bgcolor = '#cccccc'>";
    echo "<input type=file name='userfile[". ($i+2) ."] ' size = 60X/tdX/
  tr>";
  }
?>
<trxtd colspan = 2 bgcolor = '#cccccc' align = center>
<input type = hidden name = max_images value = <?=$max_images?>>
<input type = hidden name = listid value = <?=$listid?>>
<? display_form_button('upload-files'); ?>
</td>
</form>
</tr>
</table>
<?
}
```

Следует отметить, что имена файлов, которые нужно загрузить, будут вводиться в ряд текстовых полей, каждое из которых имеет тип **file,** и будут иметь форму от **userfile[0]** до **userfile[*n*]**. По существу, эти поля формы обрабатываются так же, как обрабатывались бы текстовые поля, а их именование выполняется в соответствии с соглашением, действующим по отношению к массивам.

Если при помощи PHP-сценария необходимо загрузить несколько файлов, следует придерживаться этого соглашения.

Сценарий, обрабатывающий эту форму, фактически создаст *три массива*. Давайте рассмотрим этот сценарий.

Выполнение загрузки нескольких файлов

Вероятно, читатели помнят, что код загрузки файлов был помещен в отдельный файл. Полный текст этого файла **upload.php** приведен в листинге 28.15.

Листинг 28.15 upload.php — Этот сценарий загружает все файлы, необходимые для информационного бюллетеня.

```
<?
  // эти функции помещены в отдельный файл, что позволяет
  // работать с ним более произвольным образом

  // если что-либо идет не так, как ожидается, осуществляется выход из сценария

  $max_size = 50000;

  include ('include_fns.php');
  session_start();

  // загружать файлы могут только администраторы
  if(!check_admin_user())
  {
    echo "You do not seem to be authorized to use this page.";
    exit;
  }

  // создание кнопок панели управления администратора
  $buttons = array();
  $buttons[0] = 'change-password';
  $buttons[1] = 'create-list';
  $buttons[2] = 'create-mail',-
  $buttons[3] = 'view-mail';•
  $buttons[4] = 'log-out';
  $buttons[5] = 'show-all-lists';
  $buttons[6] = 'show-my-lists';
  $buttons[7] = 'show-other-lists';

  do_html_header("Pyramid-MLM - Upload Files");

  display_toolbar($buttons);

  // проверка, что страница вызывается с обязательными данными
  if(!$userfile_name[0]||!$userfile_name[1]||!$subject||!$list)
  {
       echo "Problem: You did not fill out the form fully. The images are the
             only optional fields.  Each message needs a subject, text version
             and an HTML version.";
       do_html_footer();
       exit;
  }

  if(!db_connect())
  {
      echo "<p>Could not connect to db";
      do_html_footer();
      exit;
  }
```

```
// добавление сведений о сообщении электронной почты в БД
$query = "insert into mail values (NULL, '$admin_user',
                                        '$subject',
                                        '$list',
                                        'STORED', NULL, NULL)";
$result = mysql_query($query);
if(!$result)
{
   do_html_footer();
   exit;
}

//получение идентификатора, который MySQL присвоил этому сообщению
$mailid = mysql_insert_id();

if(!$mailid)
{
   do_html_footer();
   exit;
}

// создание каталога будет безуспешным, если это сообщение не является первым
// успешно заархивированным сообщением
@ mkdir("archive/$list", 0700);

// неудача создания конкретного каталога для этого сообщения
// свидетельствует о наличии проблемы
if(!mkdir("archive/$list/$mailid", 0700))
{
   do_html_footer();
   exit;
}

// циклический просмотр массива загруженных файлов
$i = 0;
while ($userfile[$i]&&$userfile[$i]!='none')
{
   echo "<p>Uploading ".$userfile_name[$i]." - ";
   echo $userfile_size[$i]." bytes.<br>";
   if ($userfile_size[$i]==0)
   {
      echo "Problem: $userfile_name[$i] is zero length";
      $i++;
      continue;
   }

   if ($userfile_size[$i]>$max_size)
   {
      echo "Problem: $userfile_name[$i] is over 10000 bytes";
      $i++;
      continue;
   }

   // желательно проверить, что загруженное изображение является изображением
   // если функция getimagesize() может обработать размер Web,
   // вероятно, это так и есть.
   if($i>1&&!getimagesize($userfile[$i]))
   {
      echo "Problem: $userfile_name[$i] is corrupt, or not a gif, jpeg or png";
      $i++;
      continue;
   }

   // файл типа 0 (текстовое сообщение) и файл типа 1 (сообщение html)
   // составляют отдельные случаи
   if($i=0)
      $destination = "archive/$list/$mailid/text.txt";
   else if($i == 1)
      $destination = "archive/$list/$mailid/index.html";
   else
   {
```

```
        $destination = "archive/$list/$mailid/".$userfile_name[$i];
        $query = "insert into images values ($mailid,
                                                '".$userfile_name[$i]."',
                                                '".$userfile_type[$i]."')";
        $result = mysql_query($query);
      }
      //при использовании версии PHP новее 4.03
/*
      if(!is_uploaded_file($userfile[$i]))
      {
        // выявлена возможная атака на загруженный файл
        echo "Something funny happening with '$userfile', not uploading.";
        do_html_footer();
        exit;
      }
      move_uploaded_file($userfile[$i], $destination);
*/
      // если версия старше 4.02
      copy ($userfile[$i], $destination);
      unlink($userfile[$i]);
      $i++;
    }
    display_preview_button($list, $mailid, 'preview-html');
    display_preview_button($list, $mailid, 'preview-text');
    display_button("send", "&id=$mailid");
    echo "<br><br><br><br><br>";
    do_html_footer();
?>
```

Давайте поэтапно выполним действия в листинге 28.15.

Прежде всего, мы начинаем сеанс и проверяем, что пользователь зарегистрирован как администратор — мы не хотим, чтобы кто-либо другой мог загружать файлы.

Строго говоря, следовало бы проверить также переменные **$list** и **$mailid** на предмет наличия недопустимых символов, но ради краткости мы опускаем эти действия.

Затем мы создаем и отправляем верхние колонтитулы страницы и проверяем корректность заполнения формы. В данном случае это важно, поскольку форма достаточно сложна для заполнения.

Затем мы создаем в базе данных запись для этого сообщения и создаем каталог в архиве, в котором будет храниться сообщение.

Затем дело доходит до основной части сценария, в которой выполняется проверка и перемещение каждого из загруженных файлов. Эта часть отличается в случае загрузки нескольких файлов. Теперь необходимо иметь дело с тремя массивами. Они названы **$userfile**, **$userfile_name** и **$userfile_size** и соответствуют аналогично названным эквивалентам при загрузке одного файла за исключением того, что каждый из них является массивом. Сведения о первом файле в форме будут содержаться в элементах массивов **$userfile[0]**, **$userfile_name[0]** и **$userfile_size[0]**.

Имея эти три массива, мы выполняем обычные проверки для обеспечения безопасности и перемещаем файлы в архив.

В заключение мы предоставляем администратору ряд кнопок, которые он может использовать для просмотра загруженного информационного бюллетеня перед его отправкой, и кнопку для отправки бюллетеня. Вывод, генерируемый сценарием upload.php. показан на рис. 28.13.

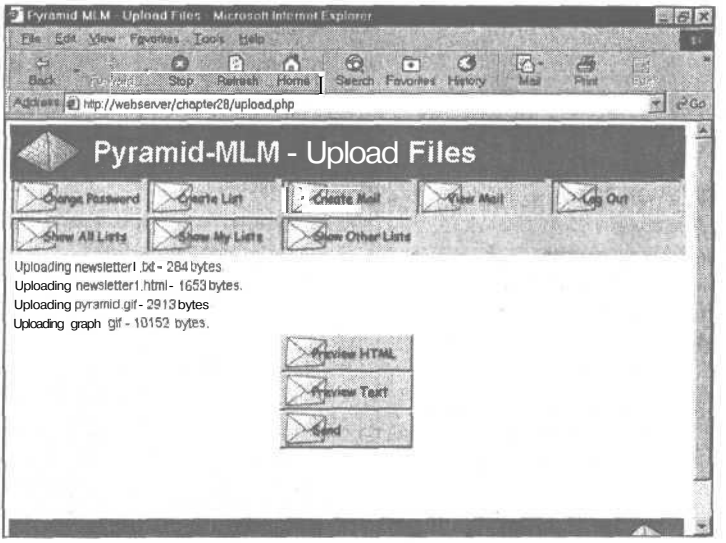


РИСУНОК 28.13
Сценарий загрузки сообщает имена загруженных файлов и их размеры.

Предварительный просмотр информационного бюллетеня

Администратор может просмотреть информационный бюллетень перед его отправкой двумя способами. Он может обратиться к функциям предварительного просмотра из экрана загрузки, если желает выполнить предварительный просмотр немедленно после загрузки.

Если же он хочет просмотреть и отправить сообщение позже, то может также щелкнуть на кнопке View Mail (Просмотреть сообщение), в результате чего отобразятся все неотправленные информационные бюллетени, хранящиеся в системе. Кнопка View Mail активизирует действие **view-mail,** которое запускает следующий код:

```
case 'view-mail' :
{
  display_items("Unsent Mail", get_unsent_mail(get_email()),
                'preview-html', 'preview-text', 'send');
  break;
}
```

Как видите, это действие также использует функцию **display_items(),** связанную с кнопками, для выполнения действий **preview-html, preview-text и send.**

Интересно отметить, что кнопки "Preview" в действительности не запускают действие, а вместо этого устанавливают связь непосредственно с информационным бюллетенем в архиве. Если вы снова посмотрите на листинги 28.6 и 28.15, то увидите, что для создания этих кнопок используется функция **display_preview_button(), а не display_button().**

Функция **display_button()** при необходимости создает воображаемую связь со сценарием с помощью параметров **GET;** функция **display_preview_button()** создает обычную ссылку внутрь архива. Эта ссылка будет отображаться в новом окне, что достигается с помощью атрибута **target=new** HTML-дескриптора привязки. Результат предварительного просмотра HTML-версии информационного бюллетеня показан на рис. 28.14.

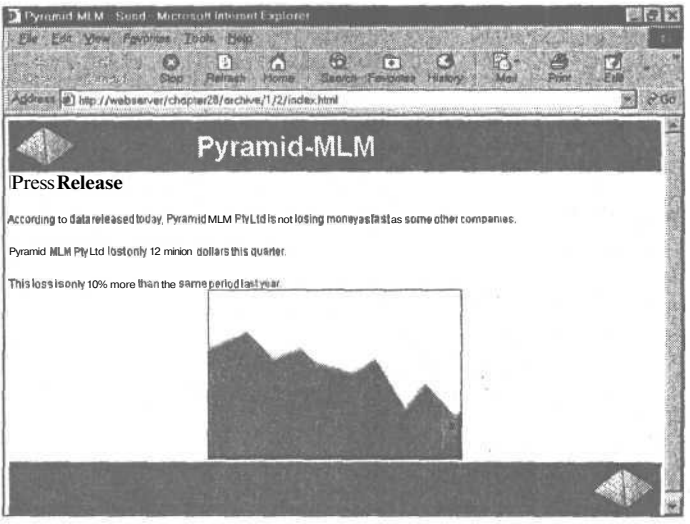


РИСУНОК 28.14
Просмотр информационного бюллетеня в формате HTML, дополненного изображениями.

Отправка сообщения

Щелчок на кнопке Send (Отправить) информационного сообщения активизирует действие **send,** которое приводит к запуску следующего кода:

```
case 'send' :
{
   send($id, $admin_user);
   break;
}
```

Это действие вызывает функцию **send()**, находящуюся в библиотеке **mlm_fns.php**. Это довольно-таки большая функция. Кроме того, именно в ней используется класс HTML MIME Mail.

Код этой функции показан в листинге 28.16.

Листинг 28.16 Функция send() из библиотеки mlm_fns.php — Эта функция, наконец-то, отправляет информационный бюллетень

```
// создание сообщения на основе хранящихся в базе данных записей и файлов
// отправка тестовых сообщений администратору или реальных сообщений в весь список
// рассылки
function send($mailid, $admin_user)
{
   if(!check_admin_user($admin_user))
      return false;

   if(!($info = load_mail_info($mailid)))
   {
      echo "Cannot load list information for message $mailid";
      return false;
   }
   $subject = $info[0];
   $listid = $info[1];
   $status = $info[2];
   $sent = $info[3];

   $from_name = 'Pyramid MLM';

   $from_address = 'return@address';

   $query = "select email from sub_lists where listid = $listid";
```

```
$result = mysql_query($query) ;
it (!$result)
{
   echo $query;
   return false;
)
else if (mysql_num_rows($result)==0)
{
   echo "There is nobody subscribed to list number $listid";
   return false;
}
else
{
   include('class.html.mime.mail.inc');

   $mail = new html_mime_mail() ;

   // считывание текстовой версии
   $filename = "archive/$listid/$mailid/text.txt";
   $fp = fopen ($filename, "r") ;
   $text = fread($fp, filesize ($filename) );
   fclose ($fp);

   // считывание HTML-версии
   $filename = "archive/$listid/$mailid/index.html";
   $fp = fopen ($filename, "r") ;
   $html = fread($fp, filesize ($filename) );
   fclose ($fp);

   // получение списка изображений, связанных с этим сообщением
   $query = "select path, mimetype from images where mailid = $mailid";
   if(db_connect())
   {
      $result = mysql_query($query) ;
      if(!$result)
      {
         echo "<p>Unable to get image list from database." ;
         return false;
      }
      $num = mysql_numrows ($result);
      for($i = 0; $i<$num; $i++)
      {
         //загрузка каждого изображения из диска
         $filename = "archive/$listid/$mailid/".mysql_result($result, $i, 0) ;
         $fp = fopen ($filename, 'r');
         $image = fread($fp, filesize ($filename) );
         fclose($fp);

       // добавление изображений к объекту mimemail
         $mail->add_html_image($image,
                               mysql_result($result, $i, 0),
                               mysql_result($result, $i, 1));
      }
   }

   // добавление HTML-кода и текста к объекту mimemail
   $mail->add_html($html, $text);

   // обратите внимание, что в данном случае сообщение создается и кодируется вне
   // цикла, а не повторно внутри него
   $mail->build_message();

   if($status == 'STORED')
   {
      //отправка HTML-версии сообщения администратору
      $mail->send(get_real_name($admin_user), $admin_user,
                  $from_name, $from_address, $subject);

      //отправка текстовой версии сообщения администратору
      mail(get_real_name($admin_user)." <".$admin_user.">", $subject,
           $text, "From: $from_name <$from_address>");

      echo "Mail sent to $admin_user";
```

```
        $query = "update mail set status = 'TESTED' where mailid = $mailid";
        if(db_connect())
        {
          $result = mysql_query($query);
        }

        echo "<p>Press send again to send mail to whole list.<center>";
        display_button('send',"&id=$mailid");
        echo "</center>";
      }
      else if($status == 'TESTED')
      {
        //отправка в весь список рассылки

        $query = "select subscribers.realname, sub_lists.email,
                         subscribers.mimetype
                  from sub_lists, subscribers
                  where listid = $listid and
                        sub_lists.email = subscribers.email";

        if(!db_connect())
          return false;

        $result = mysql_query($query);
        if(!$result)
          echo "<p>Error getting subscriber list";

        $count = 0;
        // для каждого подписчика
        while ( $subscriber = mysql_fetch_row($result) )
        {
          if($subscriber[2]=='H')
            //отправка HTML-версии тем пользователям, которые этого желаю
            $mail->send($subscriber[0], $subscriber[1], $from_name,
                        $from_address, $subject);
          else
            //отправка текстовой версии тек пользователям, которые этого желают
            mail($subscriber[0] ." <".$subscriber[1] .">", $subject,
                 $text, "From: $from_name <$from_address>" );
          $count++;
        }

        $query = "update mail set status = 'SENT', sent = now()
                  where mailid = $mailid";
        if(db_connect())
        {
          $result = mysql_query($query);
        }
        echo "<p>A total of $count messages were sent.";
      }
      else if($status == 'SENT')
      {
        echo "<p>This mail has already been sent.";
      }
    }
}
```

Эта функция осуществляет несколько различных действий.

Она выполняет тестовую отправку информационного бюллетеня администратору, прежде чем отправлять его в список рассылки. Она управляет этим процессом, отслеживая состояние фрагмента сообщения в базе данных. Когда сценарий загрузки загружает фрагмент сообщения, он устанавливает начальное состояние этого сообщения равным "STORED" ("СОХРАНЕНО").

Если функция **send()** обнаружит, что сообщение имеет состояние "STORED", она обновит его до "TESTED" ("ПРОВЕРЕНО") и отправит его администратору. Состоя-

ние "TESTED" означает, что информационный бюллетень прошел тест путем отправки его администратору. Если состояние - "TESTED", оно будет изменено на "SENT" ("ОТПРАВЛЕНО"), и сообщение будет отправлено всему списку рассылки.

Это означает, что фактически каждый фрагмент сообщения должен быть отправлен дважды: один раз в тестовом режиме и один раз в реальном.

Функция отправляет также два различных вида сообщений электронной почты: текстовую версию, которая отправляется при помощи PHP-функции **mail()**, и HTML-версию, которая отправляется с помощью класса HTML MIME Mail. Функция **mail()** многократно использовалась в этой книге, поэтому давайте рассмотрим использование класса HTML MIME Mail. Мы не будем освещать этот класс в полном объеме, но поясним, как он используется в этом достаточно типичном приложении.

Отправка начинается с включения файла класса и создания экземпляра класса:

```
include('class.html.mime.mail.inc');
$mail = new html.mime.mail();
```

Затем мы загружаем данные изображения из базы данных и просматриваем их в цикле, добавляя каждое изображение к сообщению, которое требуется отправить:

```
$mail->add_html_image($image,
                      mysql_result($result, $i, 0),
                      mysql_result($result, $i, 1));
```

Тремя параметрами, передаваемыми в функцию **add_html_image()**, являются фактическое содержимое изображения, считываемое из файла, имя файла и MIME-тип файла.

Кроме того, потребуется также добавить тело текста в текстовом и HTML-форматах:

```
$mail->add_html($html, $text);
```

Затем создается фактическое тело сообщения электронной почты:

```
$mail->build_message();
```

Наконец, имея созданное тело сообщения, можно его отправить. Это делается за счет получения и циклической обработки каждого из пользователей, подписавшихся на данный список рассылки, и применения функции **send()** HTML MIME Mail или обычной функции **mail(),** в зависимости от типа MIME, предпочитаемого пользователем:

```
if($subscriber[2]=='H')
   //отправка HTML-версии тем пользователям, которые этого желают
   $mail->send($subscriber[0], $subscriber[1], $from_name,
               $from_address, $subject);
else
   //отправка текстовой версии тем пользователям, которые этого желают
   mail($subscriber[0]." <".$subscriber[1].">", $subject,
        $text, "From: $from_name <$from_address>");
```

Первым параметром оператора **$mail->send()** должно быть действительное имя пользователя, а вторым — его адрес электронной почты.

Вот и все! Мы завершили создание приложения списков рассылки.

Расширение проекта

Как обычно случается с проектами подобного рода, существует множество способов расширения его функциональных возможностей. Может потребоваться:

- Подтверждение участия со стороны подписчиков, чтобы для пользователей нельзя было выполнить подписку без их согласия. Обычно это делается путем отправки сообщений электронной почты в их учетные записи и удаления тех сообщений, на которые не получен ответ. Этот подход будет обеспечивать также удаление из базы данных любых неправильно записанных адресов электронной почты.
- Предоставление администратору права утверждать или отклонять пользователей, которые желают подписаться на списки рассылки.
- Добавить функциональные возможности открытого списка рассылки, которые позволяют любому члену отправлять сообщение в список.
- Позволять только зарегистрированным членам просматривать архив конкретного списка рассылки.
- Предоставить пользователям возможность искать списки рассылки, соответствующие конкретному критерию. Например, пользователей могли бы интересовать информационные бюллетени, касающиеся гольфа. Как только количество информационных бюллетеней начинает превосходить некоторый заданный размер, функция поиска могла бы пригодиться для отыскания конкретных бюллетеней.

Что дальше

В следующей главе мы реализуем приложение Web-форума, которое дает возможность пользователям вести интерактивные дискуссии, структурированные по темам и цепочкам бесед.

Глава 29

Создание Web-форумов

Один из эффективных способов привлечения пользователей к сайту — создание Web-форумов. Форумы могут использоваться для решения различных задач, от поддержки дискуссионных групп по различным философским вопросам до технической поддержки продукции компании. В этой главе Web-форум будет реализован в среде PHP. В качестве альтернативы для создания форумов можно было бы использовать один из существующих пакетов, таких как Phorum.

Иногда Web-форумы называют также *дискуссионными трибунами* или *группами тематических дискуссий*. Идея форумов состоит в том, чтобы пользователи могли отправлять в них статьи или вопросы, а другие пользователи могли просматривать эти вопросы и отвечать на них. Каждая тема дискуссии в форуме называется *цепочкой*.

Мы реализуем Web-форум, названный blah-blah и позволяющий пользователям выполнять следующие действия:

- Начинать новые цепочки дискуссии, отправляя статьи
- Отправлять статьи в ответ на существующие статьи
- Просматривать отправленные в форум статьи
- Просматривать цепочки беседы в форуме
- Просматривать взаимосвязь между статьями, т.е. видеть, какие статьи являются ответами на другие статьи.

Задача

Создание форума — действительно достаточно интересная задача. Нам потребуется какой-либо способ хранения статей в базе данных с сохранением информации об авторе, заголовке и содержимом. На первый взгляд такая база данных может казаться не слишком отличающейся от базы данных Book-O-Rama.

Однако, особенность большинства программ тематических дискуссий в том, что наряду с отображением доступных статей они будут отображать взаимосвязь между статьями. Т.е., пользователь может видеть, какие статьи являются ответами на другие статьи (и за какой статьей они следуют), а какие — новыми темами обсуждения.

Примеры дискуссионных трибун, реализующих эти функциональные возможности, можно найти на многих сайтах, в том числе на сайте Slashdot:

```
http://slashdot.org
```

Способ отображения этих взаимосвязей должен быть тщательно продуман. Пользователь должен иметь возможность просматривать отдельное сообщение, цепочку беседы с показанными взаимосвязями или все цепочки в системе.

Пользователи должны также иметь возможность отправлять статьи по новым темам или ответы на существующие статьи. Эта часть приложения реализуется достаточно просто.

Компоненты решения

Как упоминалось ранее, сохранение и получение информации об авторе и текста сообщения сложности не представляет.

Наиболее трудная часть этого приложения — отыскание структуры базы данных, которая будет хранить требуемую информацию, и способа эффективного перемещения по этой структуре.

Структура статей в дискуссии может выглядеть подобно показанной на рис. 29.1.

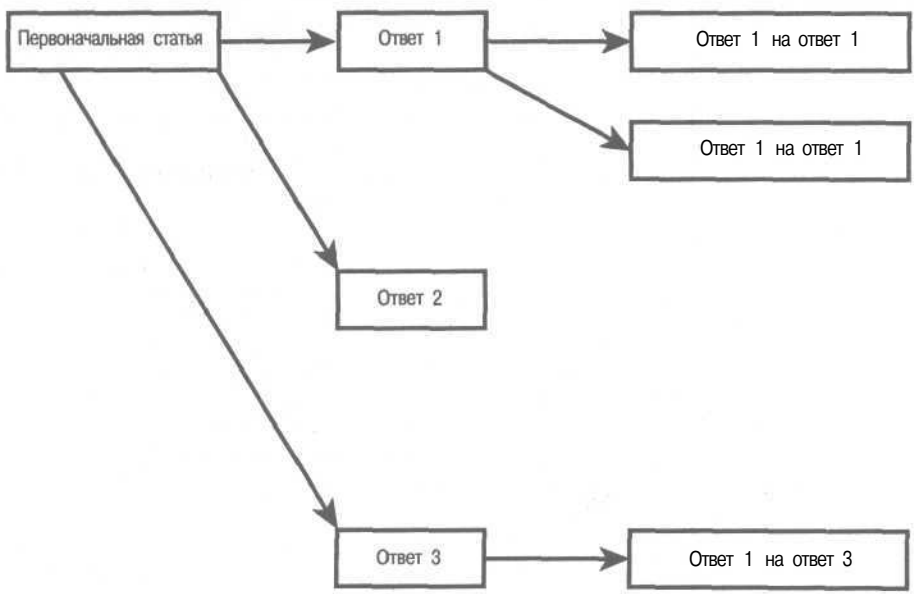


РИСУНОК 29.1 *Статья в тематической дискуссии может быть первой статьей новой темы, но чаще она является ответом на другую статью.*

Из этой схемы видно, что имеется первоначальная статья, с которой начинается тема, и три ответа на нее. Некоторые из ответов также имеют ответы. В свою очередь, эти ответы могли бы также иметь ответы, и т.д.

Схема дает ключ к тому, как можно хранить и получать данные статей и связи между статьями. Эта схема представляет *структуру дерева*. Те пользователи, которые обладают достаточным опытом в программировании, знают, что это — одна из наиболее важных используемых структур данных. Подобно любой структуре дерева схема имеет *узлы* (или статьи) и *связи* (или отношения между статьями). (Если вы еще не знакомы с использованием деревьев в качестве структур данных, не беспокойтесь — в процессе изложения материала мы рассмотрим их основные свойства.)

Чтобы эта схема работала, нужно решить следующие две основные задачи:

1. Найти способ преобразования этой структуры дерева в хранилище данных; в конкретном случае — в базу данных MySQL.
2. Найти способ восстановления данных при необходимости.

Мы начнем с реализации базы данных MySQL, которая позволить хранить статьи между их использованием.

В процессе работы мы создадим простые интерфейсы, обеспечивающие сохранение статей.

При загрузке списка статей, предназначенных для просмотра, заголовки всех статей будут загружаться в PHP-класс **tree_node**. Каждый объект **tree_node** будет содержать заголовки статьи и набор ответов на данную статью.

Ответы будут храниться в массиве. В свою очередь, каждый ответ будет объектом **tree_node**, который может содержать массив ответов на данную статью, которые сами являются объектами **tree_node,** и т.д. Это продолжается вплоть до так называемых *узлов-листьев* дерева — узлов, которые не содержат никаких ответов. В результате образуется структура дерева, которая выглядит подобно показанной на рис. 29.1.

Давайте определим некоторые термины: сообщение, на которое мы отвечаем, можно назвать *родительским узлом* текущего узла. Любые ответы на это сообщение можно назвать *дочерними узлами* текущего узла. Это легко запомнить, если представлять структуру дерева в виде генеалогического дерева.

Первую статью этой структуры дерева — не имеющую родительского узла — иногда называют *корневым* узлом.

ПРИМЕЧАНИЕ

Это может казаться не вполне естественным, поскольку обычно корневой узел рисуют в верхней части схемы, в отличие от реальных деревьев.

Для построения и отображения этой структуры дерева мы создадим рекурсивные функции. (Рекурсия была описана в главе 5.)

Для этой структуры было решено использовать класс, поскольку это — простейший способ построения сложной динамически расширяемой структуры данных для этого приложения. Это же позволяет использовать достаточно простой и элегантный код для выполнения относительно сложных действий.

Обзор решения

Чтобы действительно понять, что было сделано с данным проектом, вероятно, имеет смысл подробно рассмотреть код, который будет создан несколько позже. Это приложение несколько менее громоздко, чем ряд других, но зато код несколько более сложен.

Приложение содержит всего три реальных страницы.

Оно будет содержать основную индексную страницу, отображающую все статьи в форуме в виде связей со статьями. Из нее можно будет добавлять новые статьи, просматривать перечисленные статьи или изменять способ просмотра статей, раскрывая или сворачивая ветви дерева. (Подробнее об этом будет сказано ниже.)

Из страницы просмотра статьи можно будет отправлять ответ на эту статью или просматривать существующие ответы на нее.

Страница создания новой статьи позволяет вводить новую статью — будь то ответ на существующее сообщение или новое, ни с чем не связанное сообщение.

Схема выполняемых в системе действий показана на рис. 29.2.

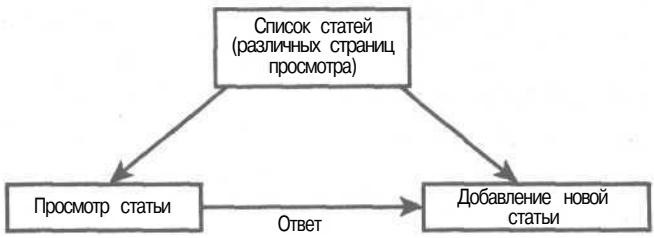


РИСУНОК 29.2
Система форума blah-blah содержит три основных части.

Краткое описание файлов этого приложения приведено в табл. 29.1

Таблица **29.1** Файлы **приложения** Web-форума

Имя файла	*Тип*	*Описание*
index.php	Приложение	Основная страница, которую пользователи будут видеть при входе на сайт. Содержит раскрываемый и свертываемый список всех статей, доступных в сайте.
new_post.php	Приложение	Форма, используемая для отправки новых статей.
store_new_post.php	Приложение	Хранит статьи, введенные в форму **new_post.php.**
view_post.php	Приложение	Отображает отдельное сообщение и список ответов на это сообщение.
treenode_class.php	Библиотека	Содержит класс **treenode,** который будет использоваться для отображения иерархии статей.
include_fns.php	Библиотека	Собирает воедино все остальные библиотеки функций (другие перечисленные в этой таблице файлы библиотечного типа), необходимые для этого приложения.
data_valid_fns.php	Библиотека	Функции проверки данных.
db_fns.php	Библиотека	Функции подключения базы данных.
discussion_fns.php	Библиотека	Функции для сохранения и получения статей.
output_fns.php	Библиотека	Функции для вывода HTML.
create_database.sql	SQL	Сценарий SQL для создания базы данных, необходимой для этого приложения.

Давайте теперь рассмотрим реализацию.

Разработка базы данных

Придется сохранять несколько атрибутов, связанных с каждой из статей, отправленных в форум: лицо, которое ее написало (*отправитель*); заголовок статьи; время ее отправки; тело статьи. Следовательно, нам потребуется таблица статей. Для каждой статьи будет создан уникальный идентификатор, названный **postid**.

Каждая статья должна содержать некоторую информацию о ее месте в иерархии. Информацию о дочерних статьях каждой статьи можно было бы хранить вместе с ней. Однако, каждая статья может иметь много ответов, поэтому такой подход может привести к определенным проблемам создания базы данных. Поскольку каждая статья может быть ответом только на одну другую статью, проще хранить ссылку на родительскую статью, т.е. статью, на которую отвечает данная.

При таком подходе для каждой статьи нужно хранить следующие данные:

- **postid:** уникальный идентификатор каждой статьи
- **parent: postid** родительской статьи
- **poster:** автор данной статьи
- **title:** заголовок данной статьи
- **posted:** дата и время отправки данной статьи
- **message:** тело статьи

В целях оптимизации приложения к этим атрибутам мы добавим ряд других.

При попытке определить, имеет ли статья какие-либо ответы, необходимо выполнить запрос на предмет наличия каких-либо других статей, для которых данная статья является родительской. Эта информация потребуется для каждой представленной в списке статьи. Чем меньше запросов придется выполнять, тем быстрее будет работать код. Избавиться от потребности в этих запросах можно, добавив поле, указывающее на наличие любых ответов. Мы назовем его **children** и сделаем по существу булевым — оно будет принимать значение 1, если узел имеет дочерние узлы, и 0, если не имеет.

Оптимизация никогда не дается даром. В данном случае сохраняются избыточные данные. Поскольку данные сохраняются двумя способами, необходимо обеспечить, чтобы оба представления были согласованы одно с другим. При добавлении дочернего узла необходимо обновлять родительский узел. Если мы разрешаем удаление дочернего узла, для обеспечения целостности базы данных также потребуется обновлять родительский узел. В данном проекте мы не собираемся создавать средства для удаления статей, поэтому решения потребует только одна часть проблемы. Если же решено расширить этот код, указанное обстоятельство необходимо иметь в виду.

Следует отметить, что некоторые базы данных могли бы несколько упростить задачу. Например, Oracle может поддерживать ссылочную целостность. При использовании MySQL, который не поддерживает триггеры или ограничения внешних ключей, придется создать собственные процедуры проверки и согласования для удостоверения в том, что данные по-прежнему имеют смысл при каждом добавлении или удалении записи.

Мы выполним еще одну оптимизацию. Тела сообщений будут отделены от остальных данных и храниться в отдельной таблице. Это связано с тем, что этот атрибут будет иметь тип **text** MySQL. Наличие этого типа в таблице может замедлить выполнение запросов данной таблицы. Поскольку для построения древовидной структуры потребуется выполнение множества небольших запросов, это привело бы значительному замедлению работы. Когда же тела сообщений хранятся в отдельной таблице, их можно получать только тогда, когда пользователь желает просмотреть конкретное сообщение.

MySQL может выполнять поиск записей фиксированного размера быстрее, чем поиск записей переменного размера. При необходимости использовать данные переменного размера производительность можно повысить, создав индексы в полях, которые будут использоваться при поиске в базе данных. Для некоторых проектов имело бы смысл оставить текстовое поле в той же записи, что и остальные данные, и указать индексы во всех столбцах, по которым будет выполняться поиск. Однако генерация индексов требует времени, а данные в форумах, скорее всего, будут изменяться постоянно, что привело бы к необходимости частого повторного генерирования индексов.

Кроме того, мы добавим также атрибут **area** на тот случай, если впоследствии решим реализовать несколько бесед с помощью одного приложения. В данном проекте эта возможность не будет реализована, но таким образом она резервируется на будущее.

SQL-код, создающий базу данных форума с учетом изложенных соображений, приведен в листинге 29.1.

Листинг 29.1 create_database.sql — SQL-код, создающий базу данных дискуссий

```
create database discussion;

use discussion;

create table header
(
  parent int not null,
  poster char(20) not null,
  title char(20) not null,
  children int default 0 not null,
  area int default 1 not null,
  posted datetime not null,
  postid int unsigned not null auto_increment primary key
);

create table body
(
  postid int unsigned not null primary key,
  message text
);

grant select, insert, update, delete
on discussion.*
to discussion@localhost identified by 'password';
```

Эту структуру базы данных можно создать, выполнив приведенный сценарий в среде MySQL следующим образом:

```
mysql -u root -p < create_database.sql
```

При этом потребуется ввести свой пароль пользователя root. Вероятно, нужно будет также изменить пароль, определенный для пользователя дискуссии, на что-то более практичное.

Дабы понять, как эта структура будет хранить статьи и их взаимосвязи, обратитесь к рис. 29.3.

Как видите, родительское поле каждой статьи в базе данных содержит **postid** статьи, расположенной в дереве над ней. Родительская статья — это статья, на которую дается ответ.

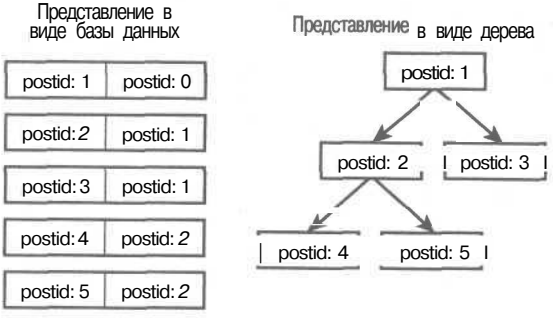


РИСУНОК 29.3
База данных хранит структуру дерева в плоской реляционной форме.

Как видно из рисунка, корневой узел **postid 1** не имеет родительского узла. Все новые темы дискуссии будут располагаться в этой позиции. Для статей этого типа их родительский узел хранится в базе данных как 0 (нуль).

Просмотр дерева статей

Теперь нам требуется способ извлечения информации из базы данных и представления ее снова в виде структуры дерева. Это будет выполнено через основную страницу **index.php**. Для иллюстрации излагаемого материала мы ввели несколько примеров статей посредством сценариев отправки статей **new_post.php** и **store_new_post.php**. Эти примеры будут рассмотрены в следующем разделе.

Вначале мы рассмотрим список статей, поскольку он служит основой сайта. После этого знакомство со всеми остальными компонентами труда не составит.

Первоначальное представление статей, которое увидит пользователь, показано на рис. 29.4.

Все статьи, представленные на этом рисунке, являются начальными. Ни одна из них не является ответом; все они — первые статьи той или иной конкретной темы.

Как видите, пользователь располагает рядом опций. В окне имеется панель меню, которая позволяет добавлять новую статью и раскрывать или сворачивать представление статей.

Чтобы понять значение этого, давайте взглянем на статьи. Рядом с некоторыми из них присутствуют символы плюса. Это означает, что на эти статьи были получены ответы. Чтобы увидеть ответы на конкретную статью, можно щелкнуть на символе плюса. Результат щелчка на одном из этих символов показан на рис. 29.5.

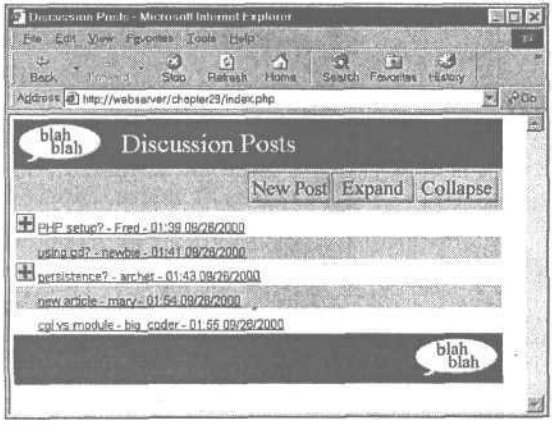


РИСУНОК 29.4
Первоначальное представление списка статей отображает статьи в "свернутой" форме.

Как видите, щелчок на символе плюса привел к отображению ответов на данную первоначальную статью. Теперь символ плюса превратился в символ минуса. Если щелкнуть на нем, все статьи в цепочке будут свернуты, приводя к исходному представлению.

Читатели могут заметить, также, что рядом с одним из ответов присутствует символ плюса. Это означает, что существуют ответы на этот ответ. Этот процесс может иметь произвольную глубину, и каждый набор ответов можно просмотреть, щелкнув на соответствующем символе плюса.

Две кнопки панели меню, Expand (Раскрыть) и Collapse (Свернуть), будут соответственно раскрывать и свертывать все возможные цепочки. Результат щелчка на кнопке Expand показан на рис. 29.6.

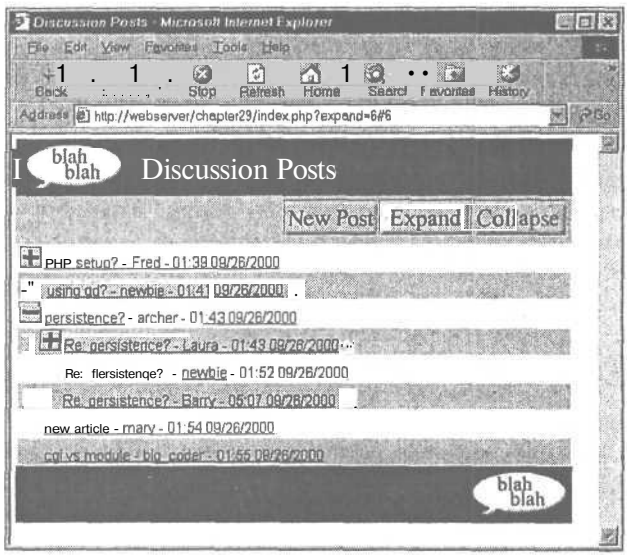


РИСУНОК 29.5
Раскрытая цепочка дискуссии по теме "persistence" ("устойчивость").

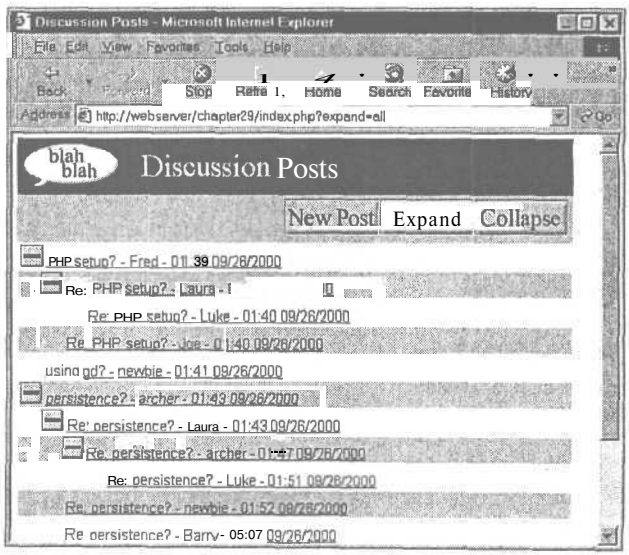


РИСУНОК 29.6
Теперь все цепочки раскрыты

Если внимательно присмотреться к рис. 29.5 и 29.6, можно видеть, что в командной строке некоторые параметры снова передаются в файл **index.php.** На рис. 29.5 Internet-адрес выглядит следующим образом:

```
http://webserver/chapter29/index.php?expand=6#6
```

Сценарий интерпретирует эту строку как "Раскрыть элемент с идентификатором postid 6". Символ # — всего лишь символ привязки HTML, который вызовет прокрутку страницы к только что раскрытой части.

На втором рисунке Internet-адрес имеет вид

```
http://webserver/chapter29/index.php?expand=all
```

Щелчок на кнопке Expand передал параметр **expand** со значением **all**.

Раскрытие и свертывание

Теперь давайте посмотрим, как это делается, рассмотрев сценарий **index.php,** приведенный в листинге 29.2.

Листинг 29.2 index.php — Сценарий создания представления статей на основной странице приложения

```
<?
  include  ('include_fns.php');
  session_start();

  // проверка, создана ли переменная сеанса
  if(!session_is_registered('expanded'))
  {
    $expanded = array();
    session_register('expanded');
  }

  // проверка, была ли нажата кнопка expand
  // значением параметра expand может быть 'all',
  // postid или же оно может быть не установлено
  if ($expand)
  {
    if($expand == 'all')
      expand_all($expanded);
    else
      $expanded[$expand] = true;
  }

  // проверка, была ли нажата кнопка collapse
  // значением параметра collapse может быть 'all',
  // postid или же оно может быть не установлено
  if($collapse)
  {
    if($collapse=="all")
      unset($expanded);
    else
      unset($expanded[$collapse]);
  }

  do_html_header("Discussion Posts");

  display_index_toolbar();

  // отображение древовидного представления
  display_tree($expanded);

  do_html_footer();
?>
```

Для выполнения задачи в этом сценарии используются три переменные:

- Переменная сеанса **$expanded**, отслеживающая раскрытые цепочки. Она может обновляться от представления к представлению, поэтому можно иметь несколько раскрытых цепочек. Эта переменная — ассоциативный массив, содержащий **postid** статей, ответы на которые будут отображаться в раскрытом виде.
- Параметр **$expand**, указывающий сценарию, какие новые цепочки следует раскрыть.
- Параметр **$collapse**, указывающий сценарию, какие цепочки следует свернуть.

При щелчке на символе плюса или минуса или на кнопке Expand или Collapse это приведет к повторному вызову сценария **index.php** с новыми значениями параметров **$expand** или **$collapse**. Переменная **$expanded** используется от страницы к странице для отслеживания того, какие цепочки должны быть раскрыты в каждом данном представлении.

Сценарий начинается с создания сеанса и добавления переменной **$expanded** в качестве переменной сеанса, если это еще не было сделано.

После этого сценарий проверяет, был ли ему передан параметр **$expand** или **$collapse**, и соответствующим образом изменяет массив **$expanded**. Взгляните на код, определяющий значение параметра **$expand**:

```
if($expand)
{
   if($expand == 'all')
      expand_all($expanded);
   else
      $expanded[$expand] = true;
}
```

При щелчке на кнопке Expand функция **expand_all()** вызывается для добавления всех цепочек, имеющих ответы, в массив **$expanded**. (Это рассматривается несколько позже.)

Чтобы раскрыть конкретную цепочку, ее **postid** будет передаваться через переменную **$expand**. Для отражения этого мы добавляем новую запись в массив **$expanded**.

Функция **expand_all()** представлена в листинге 29.3.

Листинг 29.3 Функция expand_all() из библиотеки discussion_fns.php — Эта функция обрабатывает массив $expanded для раскрытия всех цепочек в форуме_

```
function expand_all(&$expanded)
{
   // пометка всех цепочек с дочерними цепочками для отображения
   // их в раскрытом виде
   $conn = db_connect();
   $query = "select postid from header where children = 1";
   $result = mysql_query($query);
   $num = mysql_numrows($result);
   for($i = 0; $i<$num; $i++)
   {
      $expanded[mysql_result($result, $i, 0)]=true;
   }
}
```

Эта функция выполняет запрос базы данных для выяснения того, какие цепочки в форуме имеют ответы:

```
select postid from header where children = 1
```

После этого каждая из возвращенных статей добавляется в массив **$expanded**. Впоследствии этот запрос будет выполнен для экономии времени. Статьи можно было бы просто добавить в список раскрытых цепочек, попытка обработки несуществующих ответов была бы напрасной тратой времени.

Свертывание статей работает аналогично, но противоположным образом:

```
if($collapse)
{
  if($collapse=="all")
    unset($expanded);
  else
    unset($expanded[$collapse]);
}
```

Элементы из массива **$expanded** можно удалить, отменив их установку. Мы удаляем сворачиваемую цепочку или отменяем установку всего массива, если вся страница должна быть свернута.

Все описанные действия относятся к этапу предварительной обработки, поэтому известно, какие статьи должны быть отображены, а какие — нет. Основная часть сценария — вызов функции

```
display_tree($expanded);
```

которая действительно будет генерировать дерево отображенных статей.

Отображение статей

Давайте рассмотрим функцию **display_tree(),** представленную в листинге 29.4.

Листинг 29.4 Функция display_tree() из библиотеки output_fns.php — Эта функция создает корневой узел структуры дерева

```
function display_tree($expanded, $row = 0, $start = 0)
{
  // отображение вида дерева бесед

  global $table_width;
  echo "<table width = $table_width>";

  // проверка, отображается ли полный список или подсписок
  if($start>0)
    $sublist = true;
  else
    $sublist = false;

  // создание структуры дерева, представляющей структуру беседы
  $tree = new treenode($start, '', '', '', 1, true, -1, $expanded,
   $sublist);

  // указание дереву отображать себя
  $tree->display($row,$sublist);

  echo "</table>";
}
```

Основная роль этой функции — создание корневого узла структуры дерева. Она используется и для отображения всего индекса, и для создания поддеревьев ответов на странице **view_post.php**. Как видите, функция принимает три параметра. Первый, **$expanded**, — список идентификаторов статей **postid**, которые нужно отображать в раскрытом виде. Второй, **$row**, — индикатор номера строки, который будет использоваться для представления чередующихся цветов строк в списке.

Третий параметр, **$start**, сообщает функции, с какой статьи следует начинать отображение. Это — **postid** корневого узла дерева, которое должно быть создано и отображено. Если нужно отображать весь список, как имеет место на основной странице, значением этого параметра будет 0, т.е. приложение будет отображать все статьи, не имеющие родительских статей. В этом случае значение переменной **$sublist** устанавливается равным **false** и приложение отображает все дерево.

Если значение параметра больше 0, данный узел считается корневым узлом дерева, которое нужно отображать, значение **$sublist** устанавливается равным **true** и приложение создает и отображает только часть дерева. (Это будет использоваться в сценарии **view_post.php**.)

Наиболее важная задача, выполняемая этой функцией — создание экземпляра класса **treenode**, который представляет корень дерева. В действительности он не является статьей, но действует в качестве родительской статьи для статей первого уровня, которые не имеют родительской статьи. После того как дерево создано, для действительного отображения списка статей мы просто вызываем его функцию отображения.

Использование класса treenode

Код класса **treenode** приведен в листинге 29.5. (На этом этапе полезно еще раз просмотреть главу 6, чтобы вспомнить, как работают классы.)

Листинг 29.5 Класс treenode из файла **treenode_class.php** — Основная часть приложения

```
<?
// в этом файле находятся
// функции для загрузки, создания и отображения дерева
class treenode
{
  // каждый узел дерева имеет переменные-члены, которые содержат
  // все данные, необходимые для отправки статьи,
  // за исключением тела сообщения
  var $m_postid;
  var $m_title;
  var $m_poster;
  var $m_posted;
  var $m_children;
  var $m_childlist;
  var $m_depth;

  function treenode($postid, $title, $poster, $posted, $children,
                    $expand, $depth, $expanded, $sublist)
  {
    // конструктор устанавливает переменные-члены, но, что еще
    // важнее, он рекурсивно создает нижние части дерева
    $this->m_postid = $postid;
    $this->m_title = $title;
    $this->m_poster = $poster;
    $this->m_posted = $posted;
    $this->m_children =$children;
    $this->m_childlist = array();
    $this->m_depth = $depth;

    // списки, расположенные под этим узлом,
    // представляют интерес, только если он имеет дочерние
    // списки и помечен для раскрытия
    if(($sublist||$expand) && $children)
    {
      $conn = db_connect();
```

```
      $query = "select * from header where parent =
               $postid order by posted";
      $result = mysql_query ($query);

      for ($count=0; $row = @mysql_fetch_array ($result); $count++)
      {
        if ($sublist||$expanded[ $row['postid'] ] == true)
          $expand = true;
        else
          $expand = false;
        $this->m_childlist[$count]=
               new treenode($row['postid'],$row['title'],
             $row['poster'],$row['posted'],
             $row['children'],$expand,
             $depth+1, $expanded, $sublist);
      }
    }
}

function display ($row, $sublist = false)
{
  // поскольку это — объект, он отвечает аа
  // собственное отображение

  // $row указывает местоположение данной строки,
  // чтобы было известно, каким цветом нужно ее отображать

  // $sublist указывает, находимся ли мы на основной странице
  // или на странице сообщения. Для страниц сообщений переменная
  // $sublist - true.
  // В подсписках все сообщения раскрыты и не содержат
  // символы "+" или "-".

  // если данный узел — пустой корневой узел,
   // отображение пропускается
  if ($this->m_depth>-1)
  {
    // чередование цвета отображения строк
    echo "<tr><td bgcolor = " ;
    if ($row%2)
      echo "'#cccccc'>";
    else
      echo "'#ffffff'>";

    // отступ соответствует уровню вложенности
    for($i = 0; $i<$this->m_depth; $i++)
    {
      echo "<img src = 'images/spacer.gif' height = 22
                    width = 22 alt = '' valign = bottom>" ;
    >
    // отображение символа "+", "-" или пробела
    if ( !$sublist SS $this->m_children &S
           sizeof($this->m_childlist))
    // отображается основная страница, она содержит ряд
     // дочерних списков, которые раскрыты
    {
      // список раскрыт — требуется отображение
       // кнопки для свертывания
      echo "<a href = 'index.php?collapse=" .
                    $this->m_postid."#$this->m_postid'
          ><img src = 'images/minus.gif' valign = bottom
          height = 22 width = 22
             alt = 'Collapse Thread' border = 0></a>" ;
    }
```

```
        else if(!$sublist && $this->m_children)
        {
           // список свернут — требуется отображение кнопки для раскрытия
           echo "<a href = 'index.php?expand=" .
                 $this->m_postid."#$this->m_postid'>
                  <img arc = 'images/plus.gif'
                 height = 22 width = 22
                    alt = 'Expand Thread' border = 0></a>";
        }
        else
        {
           // список не имеет дочерних списков или же это —
           // подсписок — кнопка не отображается
           echo "<img src = 'images/spacer.gif' height = 22
                                 width = 22
                                 alt = ''valign = bottom>";
        }
        echo " <a name = $this->m_postid ><a href =
             'view_post.php?postid=$this->m_postid'>$this->m_title -
             $this->m_poster-".reformat_date($this->m_posted)."</a>";
        echo "</td></tr>";
        // увеличение значения счетчика строк для
        // обеспечения чередования цветов
        $row++;
     }
     // вызов функции display для каждого дочернего списка этого узла
     // обратите внимание, что узел будет иметь дочерние списки
     // только в том случае, если он раскрыт
     $num_children = sizeof($this->m_childlist);
     for($i = 0; $i<$num_children; $i++)
     {
        $row = $this->m_childlist[$i]->display($row, $sublist);
     }
     return $row;
  }
};
?>
```

Этот класс содержит функции, которые управляют видом дерева в этом приложении.

Один экземпляр класса **treenode** содержит подробную информацию о единственной статье и связи со всеми ответными статьями этого класса. Это обусловливает использование следующих переменных-членов:

```
var $m_postid;
var $m_title;
var $m_poster;
var $m_posted;
var $m_children;
var $m_childlist;
var $m_depth;
```

Обратите внимание, что **treenode** не содержит тела статьи. Нет никакой необходимости загружать его до тех пор, пока пользователь не обратится к сценарию **view_post.php**. Нужно постараться выполнить это сравнительно быстро, поскольку для отображения списка дерева приходится выполнять множество манипуляций с данными, а при обновлении страницы или нажатии кнопки приходится заново выполнять вычисления.

Именование этих переменных выполняется в соответствии со схемой именования, обычно используемой в объектно-ориентированных приложениях — их имена начинаются с символов **m_**, служащих напоминанием о том, что они являются переменными-членами класса.

Большинство этих переменных непосредственно соответствуют строкам таблицы **header** нашей базы данных.

Исключения составляют переменные **$m childlist и $m_depth.** Переменная **$m_childlist** будет использоваться для хранения ответов на данную статью. Переменная **$m_depth** будет хранить количество уровней дерева от корня до текущего уровня — она будет использоваться для отображения.

Функция конструктора устанавливает значения всех переменных:

```
function treenode($postid, $title, $poster, $posted, $children,
                  $expand, $depth, $expanded, $sublist)
{
  // конструктор устанавливает переменные-члены, но, что еще
  // важнее, он рекурсивно создает нижние части дерева
  $this->m_postid = $postid;
  $this->m_title = $title;
  $this->m_poster = $poster;
  $this->m_posted = $posted;
  $this->m_children =$children;
  $this->m_childlist = array();
  $this->m_depth = $depth;
```

При создании корневого **treenode из display_tree()** из основной страницы фактически мы создаем *фиктивный* узел, не имеющий никаких связанных с ним статей. Мы передаем некоторые начальные значения:

```
$tree = new treenode($start, '', '', '', 1, true, -1,
   $expanded, $sublist);
```

В результате создается корневой узел, значение **$postid** которого равно нулю. Эта особенность может использоваться для поиска всех статей первого уровня, поскольку они имеют нулевую родительскую статью. Глубина устанавливается равной -1, поскольку в действительности этот узел не является частью отображения. Все статьи первого уровня будут иметь нулевую глубину и будут отображаться у левого края экрана. Последующие уровни отображаются с отступом вправо.

Наиболее важное событие, происходящее в этом конструкторе — создание экземпляров дочерних узлов данного узла. Этот процесс начинается с проверки необходимости раскрытия дочерних узлов. Это выполняется только в том случае, если узел имеет какие-либо дочерние узлы, и было решено их отобразить:

```
if(($sublist||$expand) && $children)
{
  $conn = db_connect();
```

Затем мы подключаемся к базе данных и получаем все дочерние статьи следующим образом:

```
$query = "select * from header where parent =
          $postid order by posted";
$result = mysql_query($query);
```

Затем массив **$m_childlist** заполняется экземплярами класса **treenode,** содержащими ответы на статью, сохраненную в **treenode:**

```
for ($count=0; $row = @mysql_fetch_array($result); $count++)
{
   if($sublist||$expanded[ $row['postid'] ] == true)
      $expand = true;
   else
      $expand = false;
   $this->m_childlist[$count]= new treenode($row['postid'],$row['title'],
                               $row['poster'],$row['posted'],
                               $row['children'], $expand,
                               $depth+1, $expanded, $sublist);
}
```

Последняя строка создаст новые экземпляры **treenode** в соответствии с только что рассмотренным процессом, но для следующего уровня дерева. Это — рекурсивная часть функции. Родительский узел дерева вызывает конструктор **treenode,** передавая собственный **postid** в качестве родительского узла и добавляя 1 к собственному значению глубины перед его передачей.

Каждый экземпляр **treenode** будет по очереди создаваться и создавать собственные дочерние объекты до тех пор, пока не будет исчерпан список ответов или не будут раскрыты все требуемые уровни.

После того как все это выполнено, мы вызываем функцию отображения корневого экземпляра **treenode** (она определена в **display_tree()**):

```
$tree->display($row, $sublist);
```

Функция **display()** начинается с проверки того, является ли данный узел фиктивным корневым узлом:

```
if($this->m_depth>-1)
```

Таким образом фиктивный узел может быть исключен из отображения. Однако нам не требуется полностью игнорировать корневой узел. Он должен проявляться, но при этом должен уведомлять свои дочерние узлы о том, что они должны отображаться самостоятельно.

Затем функция начинает создавать таблицу, содержащую статьи. В ней операция взятия модуля (%) используется для определения требуемого цвета фона данной строки (поскольку цвета фона чередуются):

```
// чередование цвета отображения строк
echo "<tr><td bgcolor = ";if ($row%2)
   echo "'#cccccc'>";
else
   echo "'#ffffff'>";
```

Затем переменная-член **$m_depth** используется для выяснения величины отступа, которая необходима для текущего элемента. Как легко убедиться, взглянув на приведенные рисунки, чем глубже уровень ответа, тем больше его отступ. Это делается следующим образом:

```
// отступ соответствует уровню вложенности
for($i = 0; $i<$this->m_depth; $i++)
{
   echo "<img arc = 'images/spacer.gif' height = 22
                        width = 22 alt = ' ' valign = bottom>";
}
```

Следующая часть функции служит для выяснения того, нужно ли отображать кнопку плюса, минуса или вообще ничего:

```
// отображение символа "+", "-" или пробела
if ( !$sublist && $this->m_children && sizeof($this->m_childlist))
// отображается основная страница, она содержит ряд дочерних
// списков, и они раскрыты
{
  // список раскрыт — требуется отображение кнопки для свертывания
  echo "<a href = 'index.php?collapse=".
                  $this->m_postid."#$this->m_postid'
       ><img src = 'images/minus.gif' valign = bottom
   height = 22 width = 22 alt = 'Collapse Thread' border = 0></a>";
}
else if(!$sublist SS $this->m_children)
{
  // список свернут — требуется отображение кнопки для раскрытия
  echo "<a href = 'index.php?expand=".
       $this->m_postid."#$this->m_postid'><img src = 'images/plus.gif'
     height = 22 width = 22 alt = 'Expand Thread' border = 0></a>";
}
else
{
  // список не имеет дочерних списков, или же это —
  // подсписок — кнопка не отображается
  echo "<img src = 'images/spacer.gif' height = 22 width = 22
              alt = ''valign = bottom>";
}
```

Затем отображаются фактические сведения об этом узле:

```
echo " <a name = $this->m_postid Xa href =
        'view_post.php?postid=$this->m_postid'>$this->m_title -
         $this->m_poster - ".reformat_date($this->m_posted)."</a>";
echo "</td></tr>";
```

Для следующей строки цвет изменяется:

```
// увеличение значения счетчика строк
// для обеспечения чередования цветов
$row++;
```

Затем следует фрагмент кода, который будет выполняться всеми экземплярами **treenode,** в том числе корневым:

```
// вызов функции display для каждого дочернего списка этого узла
// обратите внимание, что узел будет иметь дочерние списки
// только в том случае, если он раскрыт
$num_children = sizeof($this->m_childlist);
for($i = 0; $i<$num_children; $i++)
{
  $row = $this->m_childlist[$i]->display($row, $sublist);
}
return $row;
```

Это — также рекурсивный вызов функции, который выполняется для каждого дочернего узла данного узла, чтобы они отображали себя. Мы передаем им цвет текущей строки, а они передают его обратно по завершении работы с ним. Поэтому можно отслеживать чередование цветов.

На этом мы завершим знакомство с этим классом. Код достаточно сложен. Возможно, читатели захотят поэкспериментировать с приложением, а затем, освоившись с его работой, решат снова взглянуть на код.

Просмотр отдельных статей

Вызов функции **display_tree()** приводит к созданию связей с набором статей. Если щелкнуть на одной из этих статей, мы войдем в сценарий **view_post.php** со значением параметра **postid,** соответствующим статье, которую нужно просмотреть. Пример вывода, генерируемого этим сценарием, показан на рис. 29.7.

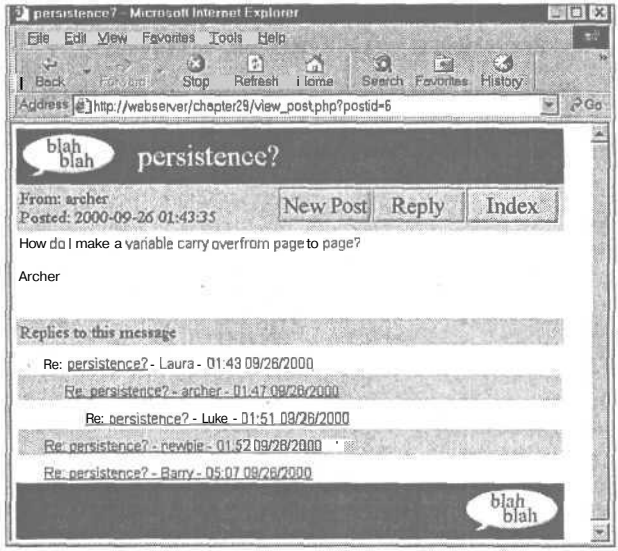


РИСУНОК 29.7
Теперь можно видеть тело сообщения данной статьи.

Этот сценарий отображает тело сообщения, а также ответы на данное сообщение. Как видите, ответы снова отображаются в виде дерева, но на этот раз полностью раскрытого без каких либо кнопок плюсов или минусов. Это — результат действия переключателя **$sublist**.

Давайте рассмотрим код сценария **viewjost.php,** приведенный в листинге 29.6.

Листинг 29.6 view_post.php — Этот листинг отображает тело отдельного сообщения

```
<?
   // включение библиотек функций
   include ('include_fns.php');

   // получение данных статьи
   $post = get_post($postid);

   do_html_header($post["title"]);

   // отображение статьи
   display_post($post);

   // если статья имеет какие-либо ответы, отображается их
   // представление в виде дерева
   if($post['children'])
   {
      echo "<br><br>";
```

```
        display_replies_line();
        display_tree($expanded, 0, $postid);
    }
    do_html_footer();
?>
```

Для выполнения стоящей задачи в этом сценарии используются три основных функции: **get_post()**, **display_post()** и **display_tree()**.

Функция **get_post()** извлекает информацию из базы данных. Код этой функции приведен в листинге 29.7.

Листинг 29.7 Функция get_post() из библиотеки discussion_fns.php - Эта функция получает сообщение из базы данных

```
function get_post($postid)
{
 //извлечение одного сообщения из базы данных и возврат его в виде массива
  if(!$postid) return false;
  $conn = db_connect();
  //получение всей информации заголовка из 'header'
  $query = "select * from header where postid = $postid";
  $result = mysql_query($query);
  if(mysql_numrows($result)!=1)
    return false;
  $post = mysql_fetch_array($result);
  //получение сообщения из тела и добавление его к предыдущему результату
  $query = "select * from body where postid = $postid";
  $result2 = mysql_query($query);
  if(mysql_numrows($result2)>0)
  {
    $body = mysql_fetch_array($result2);
    if($body)
    {
      $post['message'] = $body['message'];
    }
  }
  return $post;
}
```

Получив идентификатор **postid,** эта функция выполнит два запроса, необходимых для получения заголовка и тела сообщения для данной статьи, и поместит их в возвращаемый ею ассоциативный массив.

Затем результат выполнения этой функции передается в **display_post()** из библиотеки **output_fns.php**. Эта функция всего лишь выводит массив, выполняя небольшой объем форматирования HTML-текста, поэтому мы не останавливаемся на ней отдельно.

И наконец, сценарий **view_post.php** проверяет, имеются ли какие-либо ответы на данную статью, и вызывает функцию **display_tree()** для отображения их в формате подсписка — т.е. полностью раскрытыми без каких-либо кнопок плюсов или минусов.

Добавление новых статей

Теперь можно рассмотреть, как новая статья добавляется в форум. Пользователь может сделать это двумя способами: во-первых, щелкнув на кнопке New Post (Новая

статья) на странице индексного указателя, и во-вторых, щелкнув на кнопке Reply (Ответить) на странице **view_post.php**.

Оба эти действия активизируют один и тот же сценарий **new_post.php**, но с разными параметрами. На рис. 29.8 показан вывод, сгенерированный сценарием **new _post.php в** результате нажатия кнопки Reply.

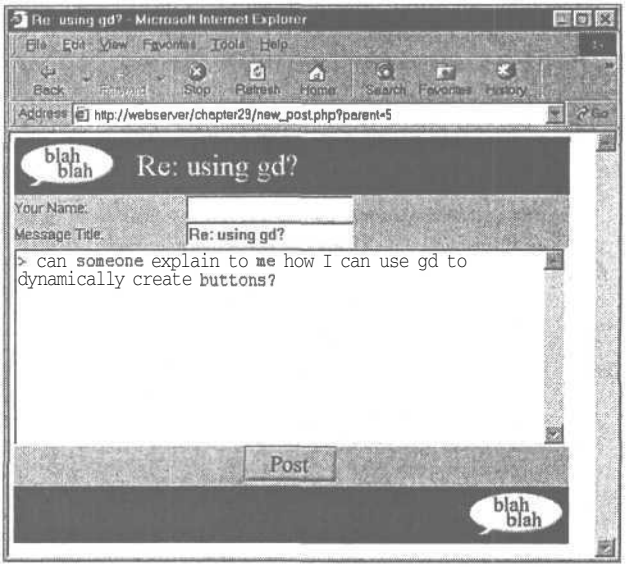


РИСУНОК 29.8
Ответы содержат текст оригинала, автоматически вставленный и помеченный.

Во-первых, взгляните на Internet-адрес:

```
http://webserver/chapter29/new_post.php?parent=5
```

Параметр, переданный в качестве **parent,** будет идентификатором **postid** родительского сообщения нового сообщения. Если вместо кнопки Reply щелкнуть на кнопке New Post, в Internet-адресе будет получено значение **parent=0**.

Во-вторых, как видите, в случае ответа текст исходного сообщения вставляется и помечается символом ">", как принято в большинстве программ чтения сообщений электронной почты и новостей.

В-третьих, заголовок этого сообщения по умолчанию повторяет заголовок исходного сообщения с префиксом "Re:".

Давайте рассмотрим код, который Генерирует этот вывод. Он приведен в листинге 29.8.

Листинг 29.8 new_post.php — Этот сценарий позволяет пользователю ввести новую статью или ответить на существующую статью

```
<?
   include ('include_fns.php');

   if(!$area)
      $area = 1;
   if (!$error)
   {
      if (!$parent)
      {
         $parent = 0;
```

```
        if(!$title)
          $title = "New Post";
      }
      else
      {
        // получение названия статьи
        $title = get_post_title($parent) ;

        // добавление префикса Re:
        if(strstr($title, "Re: ") == false )
          $title = "Re: ".$title;

        // проверка, что заголовок по-прежнему помещается в базе данных
        $title = substr($title, 0, 20);

        // добавление цитаты из статьи, на которую дается ответ
        $message = add_quoting(get_post_message($parent));
      }
    )
    do_html_header("$title");

    display new_post_form($parent, $area, $title, $message, $name);

    if ($error)
        echo "Your message was not stored.  Make sure you have filled in
              all fields and try again." ;

    do_html_footer() ;
?>
```

После выполнения рада начальных настроек этот сценарий проверяет, является ли родительская статья нулевой или какой-либо другой. Если она нулевая, значит это – новая тема и требуется некоторая дополнительная обработка.

Если это ответ (**$parent** — **postid** существующей статьи), сценарий устанавливает заголовок и текст исходного сообщения:

```
// получение названия статьи
$title = get_post_title($parent);

// добавление префикса Re:
if(strstr($title, "Re: ") == false )
   $title * "Re: ".$title;
// проверка того, что заголовок по-прежнему помещается в базу данных
$title = substr($title, 0, 20);
// добавление цитаты из статьи, на которую дается ответ
$message = add_quoting(get_post_message($parent));
```

В нем используются функции **get_post_title()**, **get_post_message()** и **add_quoting()** из библиотеки **discussion_fns.php.** Они приведены, соответственно, в листингах 29.9, 29.10 и 29.11.

Листинг 29.9 Функция get_post_title() из библиотеки discussion_fns.php — Эта функция получает заголовок сообщения из базы данных

```
function get_post_title($postid)
{
  // извлечение названия одной статьи из базы данных

  if(!$postid) return "";

  $conn = db_connect() ;

  // получение всей информации заголовка из 'header'
  $query = "select title from header where postid = $postid";
```

```
    $result = mysql_query($query);
    if(mysql_numrows($result)!=1)
      return "";
    return mysql_result($result, 0, 0);
}
```

Листинг 29.10 Функция get_post_message() из библиотеки discussion_fns.php — Эта функция получает тело сообщения из базы данных

```
function get_post_message($postid)
{
  // извлечение сообщения одной статьи из базы данных
  if(!$postid) return "";
  $conn = db_connect();
  $query = "select message from body where postid = $postid";
  $result = mysql_query($query);
  if(mysql_numrows($result)>0)
  {
    return mysql_result($result,0,0);
  }
}
```

Эти две функции получают, соответственно, заголовок и тело статьи из базы данных.

Листинг 29.11 Функция add_quoting из библиотеки discussion_fns.php — Эта функция отображает текст сообщения с отступами и символами ">"

```
function add_quoting($string, $pattern = "> ")
{
  // добавление символа цитаты для пометки текста,
  // процитированного в ответе
  return $pattern.str_replace("\n", "\n$pattern", $string);
}
```

Функция **add_quoting()** изменяет форматирование строки, начиная каждую строку исходного текста определенным символом, которым по умолчанию является >.

После того как пользователь вводит свой ответ и щелкает на кнопке Post (Отправить), активизируется сценарий **store_new_post.php**. Пример вывода, сгенерированного этим сценарием, показан на рис. 29.9.

В этом примере новая статья располагается в строке **Re: using gd? — Laura — 08:28 09/26/2000.** В остальном эта страница выглядит подобно обычной странице **index.php.**

Давайте рассмотрим код сценария **store_new_post.php**. Он приведен в листинге 29.12.

Листинг 29.12 store_new_post.php—Этот сценарий помещает новую статью в базу данных

```
<?
  include ("include_fns.php");
  if($id = store_new_post($HTTP_POST_VARS))
  {
    include ("index.php");
  }
  else
  {
    $error = true;
    include ("new_post.php");
  }
?>
```

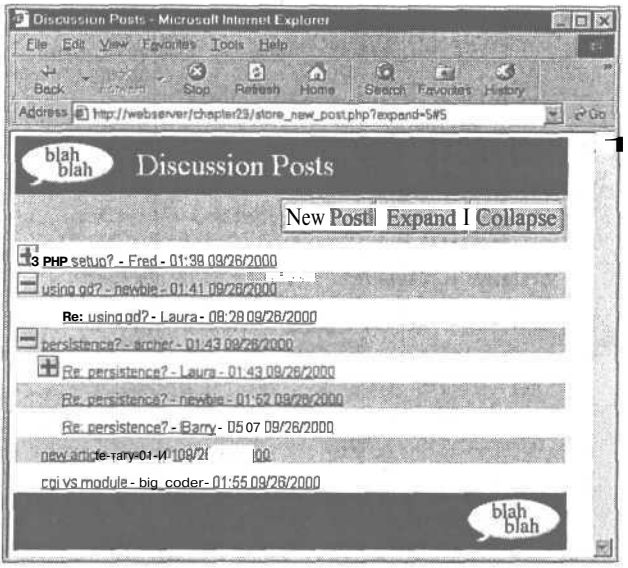


РИСУНОК 29.9
Теперь новая статья отображается в дереве.

Как видите, этот сценарий невелик. Его основная задача — вызов функции **store_new_post()**. Эта страница не имеет собственного видимого содержимого. Если сохранение выполняется успешно, мы видим страницу индексного указателя. В противном случае снова отображается страница **new_post.php**, чтобы пользователь мог повторить попытку.

Функция **store_new_post()** приведена в листинге 29.13.

Листинг 29.13 Функция store_new_post() из библиотеки discussion_fns.php — Эта функция проверяет и сохраняет новую статью в базе данных

```
function store_new_post($post)
{
   // проверка правильности и сохранение новой статьи

   $conn = db_connect() ;
   // проверка, что никакие поля не оставлены пустыми
   if(!filled_out($post))
      return false;

   $post = clean_all($post);

   // проверка существования родительской статьи
   if($post["parent"]!=0)
   {
      $query = "select postid from header where postid =
               '".$post['parent']."'";
      $result = mysql_query($query);
      if(mysql_numrows($result)!=1)
      {
         return false;
      }
   }

   // проверка, что статья не является дубликатом
   $query = "select header.postid from header, body where
               header.postid = body.postid and
               header.parent = ".$post['parent']." and
               header.poster = '".$post['poster']."' and
```

```
                header.title = '".$post['title']."' and
                header.area = ".$post['area'] . " and
                body.message = '".$post['message']."'";
    $result = mysql_query($query);
    if (!$result)
    {
        return false;
    }
    if(mysql_numrows($result)>0)
        return mysql_result($result, 0, 0);

    $query = "insert into header values
                ('".$post['parent']."',
                 '".$post['poster']."',
                 '".$post['title']."',
                 0,
                 '".$post['area']."',
                 now(),
                 NULL
                )";
    $result = mysql_query($query);
    if (!$result)
    {
        return false;
    }

    // обратите внимание, что теперь родительская статья
    // имеет дочернюю статью
    $query = "update header set children = 1 where postid =
     ".$post['parent'];
    $result = mysql_query($query);
    if (!$result)
    {
        return false;
    }

    // выяснение идентификатора данной статьи. Обратите внимание,
    // что может существовать несколько практически идентичных заголовков,
    // различающихся только идентификатором и, возможно, временем отправки.
    $query = "select header.postid from header left
                join body on header.postid = body.postid
                        where parent = '".$post["parent"]."'
                        and poster = '".$post["poster"]."'
                        and title = '".$post["title"]."'
                        and body.postid is NULL";
    $result = mysql_query($query);
    if (!$result)
    {
        return false;
    }
    if(mysql_numrows($result)>0)
       $id = mysql_result($result, 0, 0) ;

    if($id)
    {
        $query = "insert into body values ($id, '".$post["message"]."')";
        $result = mysql_query($query);
        if (!$result)
        {
           return false;
        }
       return $id;
    }
}
```

Эта функция длинна, но не слишком сложна. Ее длина обусловлена лишь тем, что вставка статьи означает вставку записей в таблицы заголовков и тела сообщений и обновление строки родительской статьи в таблице заголовков для отображения того, что теперь родительская статья имеет дочернюю статью.

На этом мы завершаем рассмотрение кода приложения Web-форума.

Расширение проекта

В этот проект можно добавить множество расширений:

- К кнопкам вида можно было бы добавить кнопки перемещения по страницам, чтобы от одного сообщения можно было перейти к следующему сообщению, предыдущему сообщению, к следующему сообщению в цепочке или предыдущему сообщению в цепочке.
- Можно было бы добавить интерфейс администрирования для создания новых форумов и для удаления старых статей.
- Можно было бы добавить аутентификацию пользователя, чтобы только зарегистрированные пользователи могли выполнять отправку статей.
- Можно было добавить какой-либо цензорный механизм с целью смягчения выражений.

Идеи по расширению приложения можно почерпнуть из существующих систем.

Использование существующих систем

Существует несколько заслуживающих внимания систем.

Phorum — проект Web-форумов программного обеспечения с открытым исходным кодом. Его средства навигации по страницам и семантика отличаются от описанных, но его структуру сравнительно легко приспособить под конкретный сайт. Примечательным свойством приложения Phorum является то, что реальным пользователем оно может быть настроено для отображения статей в виде цепочек или в двумерном виде. Дополнительную информацию о нем можно получить на сайте:

```
http://www.phorum.org
```

Еще один интересный проект — phpslash. Это — программный портал, используемый для поддержки дискуссионных трибун Slashdot. Хотя исходная программа написана на Perl, доступна и PHP-версия. Ее можно получить на сайте:

```
http://www.phpslash.org
```

Что дальше

В главе 30 рассматривается применение PDF-формата для создания привлекательных, единообразно печатаемых и частично защищенных документов. Это пригодится для ряда приложений с использованием услуг, таких как интерактивное заключение контрактов.

Глава 30

Генерация персонифицированных документов в формате переносимых документов (PDF)

В ориентированных на поддержку той или иной службы сайтах зачастую требуется создавать персонифицированные документы, генерируемые в ответ на вводимую посетителями информацию. Это может использоваться для предоставления автоматически заполняемых форм или для генерации таких персонифицированных документов, как юридические документы, письма или сертификаты.

В примере, который будет рассматриваться в этой главе, пользователю предоставляется интерактивная страница экзамена на профессиональную подготовку и генерируется соответствующий сертификат.

Мы поясним:

- Как использовать обработку PHP-строк, чтобы объединить шаблон с данными пользователя с целью создания документа в расширенном текстовом формате (Rich Text Format, RTF).
- Как использовать аналогичный подход для создания документа в формате переносимых документов (Portable Document Format, PDF).
- Как использовать PHP-функции из PDFlib для генерации аналогичного PDF-документа.

Задача

Нам требуется возможность подвергнуть посетителей экзамену, состоящему из ряда вопросов. В случае правильного ответа на достаточное количество вопросов для посе-

тителей будет генерироваться сертификат, подтверждающий успешную сдачу экзамена.

Чтобы компьютер мог оценить ответы, каждый вопрос будет связан с рядом потенциальных ответов. Из всех потенциальных ответов на каждый из вопросов только один будет правильным.

Если пользователь наберет достаточное количество баллов за ответы на вопросы, ему будет выдан сертификат. В идеальном случае формат файла сертификата должен соответствовать следующим требования:

1. Быть простым в разработке.
2. Быть пригодным для помещения в него ряда различных элементов, таких как растровые и векторные изображения.
3. Обеспечивать высокое качество печати.
4. Требовать выгрузки только небольшого файла.
5. Генерироваться практически мгновенно.
6. Требовать небольших затрат для создания.
7. Работать под управлением многих операционных систем.
8. С трудом поддаваться подделке или модификации.
9. Не требовать никакого специального обеспечения для просмотра или печати.
10. Обеспечивать единообразие отображения и печати для всех получателей.

Подобно множеству решений, которые приходится принимать время от времени, для принятия решения, удовлетворяющего максимальному количеству из этих десяти требований, вероятно, придется пойти на компромисс.

Оценка форматов документов

Важнее всего выбрать формат, в котором должен предоставляться сертификат. Возможными форматами являются бумажный документ, ASCII-текст, HTML, формат Microsoft Word или какого-либо другого текстового процессора, RTF, PostScript и PDF. Определив перечисленные ранее требования, можно рассмотреть и сравнить некоторые из доступных возможностей.

Бумажный *документ*

Предоставление сертификата на бумаге обладает рядом очевидных преимуществ. Мы сохраняем полный контроль над процессом. Прежде чем отправлять адресату каждый экземпляр сертификата, можно видеть, как в точности он выглядит. Не нужно беспокоиться о программном обеспечении или пропускной способности сети, а отпечатанный сертификат можно защитить от подделки.

Сертификат в этом формате соответствовал бы всем перечисленным требованиям, за исключением 5-го и 6-го. Его нельзя было быстро создать и доставить. Для доставки по почте могло бы потребоваться несколько дней или недель, в зависимости от местонахождения адресата.

Затраты на печать и доставку по почте каждого сертификата составляла бы от нескольких центов до нескольких долларов, а при доставке посыльным — вероятно, еще больше. Автоматическая доставка по электронной почте обошлась бы дешевле.

ASCII

Доставка документов в формате ASCII или обычного текста имеет некоторые преимущества. Совместимость не будет составлять никаких проблем. Требуемая пропускная способность может быть небольшой, поэтому стоимость доставки достаточно низка. Простота конечного результата обусловит простоту разработки и очень большую скорость генерирования документа сценарием.

Однако, если посетителям предоставляется ASCII-файл, внешний вид сертификата очень мало поддается контролю. Мы не можем управлять шрифтами или разрывами страниц. Можно только вводить текст и лишь в очень незначительной степени управлять форматированием. Мы лишены возможности контролировать дублирование или модификацию документа получателем. При использовании этого метода получателю проще всего подделать свой сертификат.

HTML

Естественным форматом при доставке документа через Web является HTML. Hypertext Markup Language (язык гипертекстовой разметки) специально предназначен для этого. Как, несомненно, читатели уже знают, он включает в себя управление форматированием, синтаксис для вставки таких объектов, как изображения, и совместим (с некоторыми различиями) со множеством операционных систем и программ. Этот формат очень прост, поэтому разработка документа в нем будет проста, а генерация и доставка его сценарием будет выполняться быстро.

Недостатки использования HTML-формата для этого приложения следующие: ограниченная поддержка таких связанных с печатью атрибутов форматирования, как разрывы страниц; недостаточное единообразие вывода на различных платформах и в различных программах; варьирующееся качество печати. Кроме того, хотя HTML-документ может содержать внешние элементы любого типа, отображение или использование этих элементов браузером для нестандартных типов не гарантируется.

Форматы текстовых процессоров

Для проектов, предназначенных для локальных сетей, предоставление документов в формате текстового процессора имеет определенный смысл. Однако при доставке через Internet использование собственного формата текстового процессора приведет к отсеву некоторых посетителей. Тем не менее, учитывая его преобладание на рынке, использование формата Microsoft Word имело бы смысл. Большинство пользователей будут иметь доступ либо к Word, либо к текстовому процессору, который может читать файлы в этом формате.

Пользователи Windows, не имеющие Word, могут загрузить бесплатную программу просмотра документов Word из

```
http://www.microsoft.com/office/000/viewers.htm
```

Генерация документа в формате Microsoft Word обладает рядом преимуществ. При наличии копии программы Word разработка документа не представляет трудности. Можно в большой степени управлять внешним видом документов при печати, а также иметь множество возможностей в отношении содержимого. Кроме того, изменение документа получателем можно сделать сравнительно трудным, указав Word на необходимость запроса пароля.

К сожалению, файлы в формате Word могут быть большими, особенно если они содержат изображения или другие сложные элементы. Кроме того, отсутствует простой

способ динамической их генерации из PHP. Формат задокументирован, однако является двоичным, а документация по формату поставляется в соответствии с условиями лицензионного соглашения.

Расширенный текстовый формат

Расширенный текстовый формат, или RTF-формат, обеспечивает большинство возможностей, доступных в формате Word, но файлы проще генерировать. Мы по-прежнему располагаем множеством возможностей в отношении макета и форматирования печатной страницы. В документ по-прежнему можно помещать такие элементы, как векторные или растровые изображения. Опять-таки, по-прежнему можно не сомневаться, что при просмотре или печати документа пользователь получит результаты, аналогичные нашим.

RTF — это текстовый формат Microsoft Word. Он предназначен для использования в качестве формата обмена при передаче документов между различными программами. В определенном смысле он аналогичен HTML-формату. Для передачи информации о форматировании в нем применяется синтаксис и ключевые слова, а не двоичные данные, поэтому он сравнительно читабелен для человека.

Этот формат хорошо задокументирован. Его спецификация доступна бесплатно и ее можно найти по адресу

```
http://msdn.microsoft.com/library/specs/rtfspec.htm
```

Простейший способ генерации RTF-документа — выбор команды Save As RTF (Сохранить как RTF) в используемом текстовом процессоре. Поскольку RTF-файлы содержат только текст, их можно генерировать непосредственно, а существующие документы легко изменять.

Поскольку формат задокументирован и его документация доступна бесплатно, документы в этом формате могут читаться большим количеством программ, нежели документы в двоичном формате Word. Однако следует иметь в виду, что пользователи, открывающие сложный RTF-файл в более ранних версиях Word или в других текстовых процессорах, часто получают несколько иные конечные результаты. Каждая новая версия Word добавляет в RTF-формат новые ключевые слова. Поэтому, как правило, более ранние версии будут игнорировать элементы управления, которые они не распознают или которые было решено не реализовывать.

Что касается ранее приведенного перечня требований, то сертификат в RTF-формате можно было бы легко разработать, используя Word или какой-либо другой текстовый процессор; этот сертификат может содержать множество различных элементов типа векторных или растровых изображений; этот формат позволяет получить отпечаток высокого качества; документ может быть сгенерирован легко и быстро; он может доставляться электронными средствами при низких затратах.

Формат будет работать с разнообразными приложениями и операционными системами, хотя и приводя к несколько различным результатам. С другой стороны, RTF-документ свободно и без труда может быть изменен любым пользователем, что нежелательно для сертификатов и ряда других типов документов. Размеры сложных документов могут оказаться достаточно большими.

RTF-формат вполне подходит для многих приложений доставки документов, поэтому в рассматриваемом примере мы будем его использовать в качестве одной из опций.

PostScript

PostScript, разработанный компанией Adobe, представляет собой язык описания страниц. Это обладающий большими возможностями и сложный язык программирования, предназначенный для представления документов в независящей от устройств форме. Другими словами, он обеспечивает описание, которое будет приводить к одинаковым результатам на различных устройствах, таких как принтеры и мониторы. Этот формат очень хорошо задокументирован. Ему посвящены, по меньшей мере, три объемных книги, а также бесчисленное множество Web-сайтов.

PostScript-документ может содержать очень точное форматирование, текст, изображения, внедренные шрифты и другие элементы. PostScript-документ можно легко генерировать из приложения, выводя его в драйвер принтера PostScript. При желании можно было бы даже научиться программировать его непосредственно.

PostScript-документы в достаточной степени пригодны для переноса из одной системы в другую. Они будут давать единообразные качественные отпечатки на различных устройствах и под управлением различных операционных систем.

Использование формата PostScript для распространения документов имеет несколько значительных недостатков:

- Файлы могут иметь очень большие размеры.
- Чтобы их использовать, многим пользователям потребуется загрузить дополнительное программное обеспечение.

Большинство пользователей UNIX смогут работать с PostScript-файлами, но пользователям Windows, как правило, придется загружать такую программу просмотра, как GSview, в которой применяется интерпретатор Ghostscript PostScript. Существуют варианты этой программы для множества платформ. Хотя она доступна бесплатно, мы вовсе не намерены вынуждать пользователь загружать дополнительное программное обеспечение.

Подробнее о программе Ghostscript можно прочесть на сайте:

```
http://www.ghostscript.com/
```

а загрузить ее можно из:

```
http://www.cs.wisc.edu/~ghost/
```

Применительно к создаваемому приложению формат PostScript очень хорошо подходит для создания единообразного высококачественного вывода, но не обеспечивает выполнение большинства других требований.

Переносимый формат документов

К счастью, есть формат, обеспечивающий большинство возможностей PostScript, но обладающий существенными преимуществами. Формат переносимых документов (Portable Document Format, PDF), также разработанный компанией Adobe, является средством распространения документов, которые вели бы себя одинаково на различных платформах и обеспечивали бы предсказуемый высококачественный вывод на экран или на бумагу.

В документации, выпускаемой компанией Adobe, PDF описывается как "открытый стандарт де-факто для распространения электронных документов по всему миру. PDF компании Adobe — универсальный формат файлов, который сохраняет все шрифты, цвета форматирования и графические изображения любого исходного документа неза-

висимо от приложения и платформы, использованных для его создания. PDF-файлы компактны и могут использоваться совместно, просматриваться, управляться и печататься именно так, как требуется, причем любым пользователем, располагающим бесплатной программой Adobe Acrobat Reader".

Документацию по PDF можно найти по адресу:

```
http://partners.adobe.com/asn/developer/technotes.html
```

а также на множестве других Web-сайтов и в официальных книгах.

Учитывая выдвинутые нами требования, PDF выглядит весьма и весьма многообещающе.

PDF-документы обеспечивают единообразный, высококачественный вывод и могут содержать такие элементы, как растровые и векторные изображения. В них может использоваться сжатие с целью получения небольшого файла. Они могут недорого доставляться электронными средствами и пригодны для использования в среде основных операционных систем. Эти документы могут содержать элементы управления безопасностью.

Недостаток PDF состоит в том, что большинство программ для создания PDF-документов являются коммерческими.

Для просмотра PDF-файлов требуется программа чтения, но Adobe распространяет бесплатные версии Acrobat Reader для Windows, UNIX и Macintosh. Многие посетители сайта будут уже знакомы с расширением .pdf и, скорее всего, уже будут иметь установленную программу чтения.

PDF-файлы служат хорошим средством распространения привлекательных, пригодных для печати документов, особенно когда нежелательно, чтобы получатели могли их легко модифицировать. Мы рассмотрим два различных способа генерирования сертификата в PDF-формате.

Компоненты решения

Чтобы система работала, потребуется иметь возможность проэкзаменовать знания пользователей и (если они выдержали тест) сгенерировать сертификат, удостоверяющий их квалификацию. Мы поэкспериментируем с генерацией такого сертификата тремя различными способами: двумя с использованием PDF-формата и одного с использованием RTF-формата.

Давайте несколько подробнее рассмотрим требования, предъявляемые к каждому из этих компонентов.

Система вопросов и ответов

Создание гибкой системы интерактивного экзаменования, которая позволила бы применять множество различных типов вопросов и различных типов носителей для дополнительной информации, обеспечивающей полезную реакцию на неправильные ответы и продуманный сбор статистической информации, равно как и создание отчетов — сама по себе сложная задача.

В этой главе основное внимание уделено задаче создания персонифицированных документов, предназначенных для доставки через Web. Поэтому мы создадим только очень простою экзаменационную систему.

Эта экзаменационная система не требует использования никакого специального программного обеспечения. В ней HTML-форма используется для задания вопросов, а PHP-сценарий — для обработки ответов. Мы уже делали подобное, начиная с главы 1.

Программное обеспечение для генерации документов

Для генерации RTF- или PDF-документов из шаблонов никакое дополнительное программное обеспечение Web-сервера не требуется, однако оно будет необходимо для создания шаблонов. Чтобы использовать PHP-функции создания PDF-документов, поддержку PDF придется скомпилировать в PHP-сценарий. (Подробнее этот вопрос будет рассматриваться несколько позже.)

Программное обеспечение для создания шаблона RTF-документов

Для генерации RTF-файлов можно использовать любой текстовый процессор по своему выбору. Для создания шаблона сертификата применялся Microsoft Word. Этот шаблон можно найти на сопровождающем книгу CD-ROM в каталоге для этой главы.

Если предпочтение отдано другому текстовому процессору, все же имеет смысл протестировать готовый шаблон в программе Word, поскольку именно эту программу будет использовать большинство посетителей вашего сайта.

Программное обеспечение для создания шаблона PDF-документов

Создание PDF-документов несколько сложнее. Простейший путь — приобрести программу Adobe Acrobat. Она позволит создавать качественные PDF-документы из среды различных приложений. Именно эта программа применялась для создания файла шаблона для данного проекта.

Для создания документа использовался Microsoft Word. Один из инструментов, входящих в состав пакета Acrobat — Adobe Distiller. В среде Distiller необходимо выбрать несколько параметров, не устанавливаемых по умолчанию. Файл должен быть сохранен в формате ASCII, а сжатие должно быть отключено. После того как эти параметры установлены, создание PDF-документа столь же просто, как и печать.

Более подробную информацию о программе Acrobat можно найти на сайте

```
http:/www.adobe.com/products/acrobat/
```

Приобрести ее можно либо через Internet, либо у обычного продавца программного обеспечения.

Еще одну возможность создания PDF-документов предоставляет программа преобразования **ps2pdf**, которая, как видно из ее имени, преобразует PostScript-файлы в PDF-файлы. Ее преимущество состоит в том, что она распространяется бесплатно. Однако она не всегда обеспечивает хорошие результаты для документов, которые содержат изображения или нестандартные шрифты. Программа преобразования **ps2pdf** поставляется с упоминавшимся ранее пакетом Ghostscript.

Понятно, что если решено создавать PDF-файл этим способом, вначале придется создать PostScript-файл. Как правило, для этого пользователи UNIX будут применять утилиту a2ps или **dvips**.

При работе в среде Windows PostScript-файлы можно создать, не прибегая к использованию программы Adobe Distiller, хотя и посредством несколько более сложного процесса. Придется установить драйвер принтера PostScript. Например, можно использовать драйвер Apple LaserWriter IINT. Если драйвер PostScript не установлен, его можно выгрузить из сайта компании Adobe:

```
http://www/adobe.com/support/downloads/5672.htm
```

Для создания PostScript-файла потребуется выбрать соответствующий принтер и команду Print to File (Печать в файл), которая обычно находится в диалоговом окне Print (Печать).

После этого большинство Windows-приложений создают файл с расширением .prn. Этот файл должен быть PostScript-файлом. Вероятно, его необходимо переименовать, изменив расширение на .ps. Он должен быть пригодным для просмотра с помощью GSview либо другой программы просмотра PostScript-файлов или для создания PDF-файла с помощью утилиты **ps2pdf.**

Имейте в виду, что различные драйверы принтеров создают PostScript-вывод различного качества. Может оказаться, что некоторые созданные PostScript-файлы вызывают ошибки при попытке их обработки утилитой **ps2pdf**. В этом случае мы предлагаем воспользоваться другим драйвером печати.

Если вы намерены создавать лишь небольшое количество PDF-файлов, для выполнения этой задачи может подойти интерактивная служба, предоставляемая компанией Adobe. При уплате ежемесячной абонентской платы в размере $9,99 вы получаете возможность загружать на сервер файлы в ряде различных форматов и выгружать из него PDF-файл. Эта служба успешно работала для созданного нами сертификата, но она не позволяет выбирать опции, которые важны для рассматриваемого проекта. Созданный PDF-файл будет сохранен в виде двоичного файла со сжатием. Это обстоятельство весьма затрудняет его изменение.

Упомянутую службу можно найти на сайте

```
http://createpdf.adobe.com/
```

Для тех, кто желает ее испытать, в этой службе имеется возможность бесплатного пробного использования.

Бесплатный, использующий FTP, интерфейс к утилите **ps2pdf** входит также в состав пакета Net Distiller:

```
http://www.babinszki.com/distiller/
```

Программное обеспечение для создания PDF-документов программным путем

Поддержка создания PDF-документов доступна и в PHP. Существуют две библиотеки функций, предназначенные для выполнения аналогичных задач. Поскольку в них используются внешние библиотеки, ни одна из них не компилируется в PHP по умолчанию.

PHP-функции используют библиотеку PDFlib, доступную по адресу:

```
http://www.pdflib.com
```

Другая библиотека функций, ClibPDF, доступная по адресу:

```
http://www.fastio.com/
```

Обе эти библиотеки аналогичны. Они предоставляют API функций для генерации PDF-документов. Мы предпочитаем использовать библиотеку PDFlib, поскольку, как нам кажется, она обновляется и обслуживается более регулярно.

Следует отметить, что ни одна из этих библиотек не принадлежит к программному обеспечению с открытым исходным кодом. Обе допускают определенное бесплатное некоммерческое использование, но требуют платы за лицензию, если вы собираетесь с их помощью предоставлять коммерческие услуги.

Проверить, установлена ли уже библиотека PDFlib в системе, можно, взглянув на результаты вывода функции **phpinfo()**. В разделе pdf можно выяснить, включена ли поддержка PDFlib, а также номер используемой версии этой библиотеки.

Для установки PDFlib потребуется также установить и библиотеку TIFF, которая доступна на сайте

```
http://www.libtiff.org/
```

и библиотеку JPEG, доступную по адресу

```
ftp://ftp.uu.net/graphics/jpeg/
```

В системе UNIX эти компоненты программного обеспечения устанавливаются обычным способом при помощи команд **configure и make.** При этом придется перекомпилировать PHP с использованием ключа **--with-pdflib**.

На сервере Windows простейший способ получения поддержки PDFlib — загрузка одного из неофициальных заранее скомпилированных двоичных файлов, доступных в Web. Для проверки кода, создание которого описывается в этой главе, на компьютере Windows мы выгрузили заранее скомпилированный двоичный файл из сайта

```
http://php.weblogs.com/easywindows
```

Еще одна популярная надстройка доступна на сайте

```
http://www.php4win.de
```

Обзор решения

Мы создадим систему, которая будет приводить к трем возможным результатам. Как видно из рис. ЗОЛ, мы будем задавать контрольные вопросы, оценивать ответы, а затем генерировать сертификат одним из трех способов:

- Мы будем генерировать RTF-документ из пустого шаблона.
- Мы будем генерировать PDF-документ из пустого шаблона.
- Мы будем генерировать PDF-документ программным путем за счет использования функций библиотеки PDFlib.

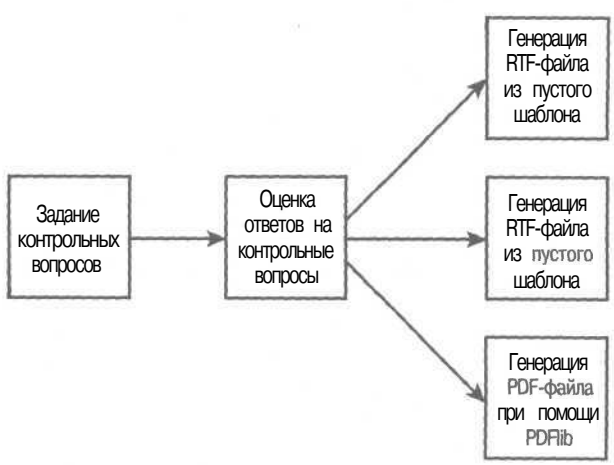


РИСУНОК 30.1 *Наша система сертификации будет генерировать один из трех различных сертификатов.*

Краткий обзор файлов, используемых в проекте системы сертификации, приведен в таблице ЗОЛ.

Таблица 30.1 Файлы, используемые в приложении Certification

Имя	*Тип*	*Описание*
index.html	HTML-страница	HTML-форма, содержащая экзаменационные вопросы
score.php	Приложение	Сценарий для оценки ответов пользователей
rtf.php	Приложение	Сценарий для генерации RTF-сертификата из шаблона
pdf.php	Приложение	Сценарий для генерации PDF-сертификата из шаблона
pdflib.php	Приложение	Сценарий для генерации PDF-сертификата за счет использования библиотеки PDFlib
signature.tif	Изображение	Растровое изображение подписи, которое будет включено в сертификат PDFlib
PHPCertification.rtf	RTF-файл	Шаблон сертификата в RTF-формате
PHPCertification.pdf	PDF-файл	Шаблон сертификата в PDF-формате

Приступим к исследованию приложения.

Задание вопросов

Файл **index.html** весьма прост. Он должен содержать HTML-форму, запрашивающую пользователя о его имени, и ответы на ряд вопросов. Скорее всего, в реальном экзаменационном приложении эти ответы следовало бы получать из базы данных. Но в данном случае нас интересует создание сертификата, поэтому мы просто жестко запрограммируем в HTML-коде несколько ответов.

Поле **name** является полем ввода текста. Каждый ответ имеет три переключателя, позволяющие пользователю указать правильный, по его мнению, ответ. Форма содержит кнопку с изображением, служащую для представления ответа.

Код этой страницы приведен в листинге 30.1.

Листинг 30.1 index.html — HTML-страница, содержащая контрольные вопросы

```
<html>
  <body>
    <h1><p align = center>
          <img src = "rosette.gif" alt = "">
          Certification
          <img src = "rosette.gif" alt = ""></h1>
    <p>You too can earn your highly respected PHP certification
       from the world famous Fictional Institute of PHP Certification.
    <p>Simply answer the questions below:

    <form action = score.php method = post>

      <p>Your Name <input type = text name = name>

      <p>What does the PHP statement echo do?
      <ol>
        <liXinput type = radio name = q1 value = 1>
            Outputs strings.
        <liXinput type = radio name = ql value = 2>
            Adds two numbers together.
        <lixinput type = radio name = ql value = 3>
```

```
            Creates a magical elf to finish writing your code.
        </ol>

        <p>What does the PHP function cos() do?
        <ol>
          <lixinput type = radio name = q2 value = 1>
              Calculates a cosine in radians.
          <liXinput type = radio name = q2 value = 2>
              Calculates a tangent in radians.
          <liXinput type = radio name = q2 value = 3>
              It is not a PHP function it is a lettuce.
        </ol>

        <p>What does the PHP function mail O do?
        <ol>
          <lixinput type = radio name = q3 value = 1>
              Sends a mail message.
          <liXinput type = radio name = q3 value = 2>
              Checks for new mail.
          <liXinput type = radio name = q3 value * 3>
              Toggles PHP between male and female mode.
        </ol>

        <p align = centerXinput type = image src = "certify-me.gif"
                  border = 0>
    </form>
  </body>
</html>
```

Результат загрузки файла **index.html** в Web-браузер показан на рис. 30.2.

Оценка ответов

Когда пользователь представляет свои ответы на вопросы, приведенные в **index.html,** их нужно оценить и подсчитать общее количество баллов. Это выполняется в сценарии **score.php,** код которого приведен в листинге 30.2.

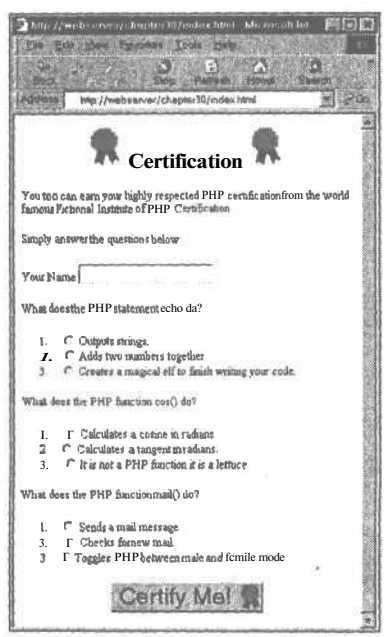


РИСУНОК 30.2
*Форма **index.html** предлагает посетителю ответить на контрольные вопросы.*

Листинг 30.2 score.php — Сценарий для оценки результатов экзамена

```
<?
  // проверка получения всех необходимых данных
  if($q1==''||$q2==''||$q3==''||$name=='')
  {
    echo "<h1><p align = center><img src = 'rosette.gif' alt = ' '>
                                Sorry:
                                <img src = 'rosette.gif' alt = ' '></h1>";
    echo "<p>You need to fill in your name and answer all questions";
  }
  else
  {
    // добавление к общей сумме набранных баллов
    $score = 0;
    if ($q1 == 1) // правильный ответ на q1 — 1
      $score++;
    if($q2 == 1) // правильный ответ на q2 — 1
      $score++;
    if($q3 == 1) // правильный ответ на q3 — 1
      $score++;

    // преобразование суммы баллов в проценты
    $score = $score / 3 * 100;

    if($score < 50)
    {
      /I посетитель не выдержал экзамен
      echo "<h1><p align = centerXimg src = 'rosette.gif' alt = ' ' >
                                Sorry:
                                <img src = 'rosette.gif' alt = ' '></h1>";
      echo "<p>You need to score at least 50% to pass the exam" ;
    }
    else
    {
      // создание строки, содержащей итог экзамена с точностью
      // до одного десятичного разряда
      $score = number_format($score, 1);

      echo "<h1><p align = centerXimg src = 'rosette.gif' alt = ' '>
                                Congratulations
                                img src = 'rosette.gif' alt = ' '></h1>";
      echo "<p>Well done $name, with a score of $score%,
                you have passed the exam. ";

      // обеспечение связей со сценариями,
      // которые генерируют сертификаты
      echo "<p>Please click here to download your certificate as
                a Microsoft Word (RTF) file. ";
      echo "<form action = 'rtf.php' method = get>";
      echo "<center>
              <input type = image src = 'certificate.gif' border = 0>
            </center>";
      echo "<input type = hidden name = score value = '$score'>";
      echo "<input type = hidden name = name value = '$name'>";
      echo "</form>";

      echo "<p>Please click here to download your certificate as
                a Portable Document Format (PDF) file. ";
      echo "<form action = 'pdf.php' method = get>";
      echo "<center>
              <input type = image src = 'certificate.gif' border = 0>
            </center>";
      echo "<input type = hidden name = score value = '$score'>";
```

```
            echo "<input type = hidden name = name value = '$name'>";
            echo "</form>";
            echo "<p>Please click here to download your certificate as
              a Portable Document Format (PDF) file generated with PDFLib. ";
            echo "<form action = 'pdflib.php' method = get>";
            echo "<center>
                    <input type = image src = 'certificate.gif' border = 0>
                  </center>";
            echo "<input type = hidden name = score value = '$score'>";
            echo "<input type = hidden name = name value = '$name'>";
            echo "</form>";
        }
    }
?>
```

Этот сценарий будет отображать сообщение, если пользователь не ответил на все вопросы или набрал менее определенного количества баллов.

Если пользователь успешно ответил на вопросы, для него разрешается генерация сертификата. Результат успешного посещения сайта показан на рис. 30.3.

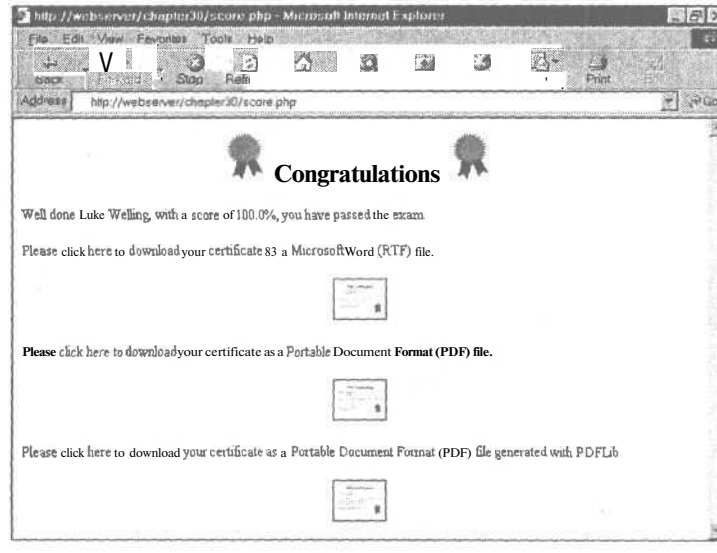


РИСУНОК 30.3 *Сценарий score.php предоставляет пользователям, успешно выдержавшим экзамен, возможность сгенерировать сертификат одним из трех способов.*

С этого момента пользователю доступно три возможности. Он может получить RTF-сертификат или один из двух PDF-сертификатов. Мы рассмотрим сценарии, отвечающие за генерацию каждого из них.

Генерация *RTF-сертификата*

Теперь ничто не может помешать нам сгенерировать RTF-документ, записав ASCII-текст в файл или в строковую переменную, но для этого требует изучить еще один набор элементов синтаксиса.

Ниже приведен очень простой RTF-документ:

```
{\rtf1
{\fonttbl {\f0 Arial;}{\f1 Times New Roman;}}
\f0\fs28 Heading\par
\f1\fs20 This is an rtf document.\par
}
```

Этот документ устанавливает таблицу шрифтов, содержащую два шрифта: Arial, ссылка на который будет выглядеть как на **f0**, и Times New Roman, ссылка на который будет осуществляться через П. Затем он записывает строку Heading, используя шрифт **fo** (Arial) размером 28 (14 пунктов). Управляющая последовательность **\par** указывает конец абзаца. Затем мы записываем строку **This is an rtf document,** используя шрифт **f1** (Times New Roman) размера 20 (10 пунктов).

Подобный этому документ можно было бы сгенерировать вручную, однако PHP не содержит никаких встроенных функций, которые могли бы упростить выполнение таких трудных задач, как внедрение графических изображений. К счастью, во многих документах структура, стиль и значительная часть текста являются статическими, и только небольшие фрагменты изменяются от одного лица к другому. Более эффективный способ генерации документа состоит в использовании шаблона.

Сложный документ, подобный показанному на рис. 30.4, можно легко создать, используя текстовый процессор.

Показанный шаблон содержит заполнители, подобные **«NAME»** для пометки мест вставки динамической даты. Внешний вид этих заполнителей не имеет значения. Мы используем понятное описание, помещенное между двумя наборами угловых скобок. Важно выбрать заполнители, случайное появление которых в остальной части документа маловероятно. Создание макета шаблона упрощается, если заполнители имеют приблизительно такую же длину, как и данные, которыми они будут замещены.

В этом документе используются заполнители **«NAME», «Name», «score»** и **<<mm/dd/yyyy>>**. Обратите внимание, что используются и NAME, и **Name,** поскольку для их замещения предполагается использовать чувствительный к регистру метод.

Теперь, когда шаблон создан, требуется сценарий для его персонификации. Этот сценарий называется **rtf.php** и его код приведен в листинге 30.3.

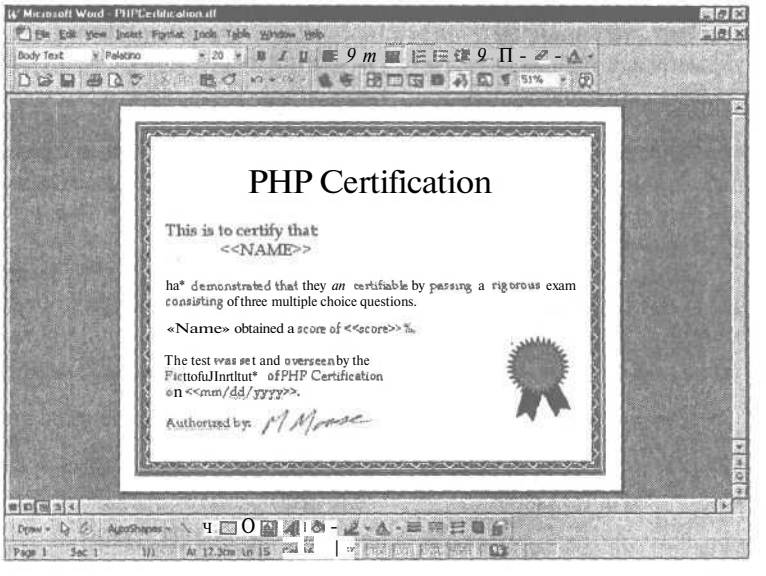


РИСУНОК 30.4
Используя текстовый процессор, можно легко создать сложный и привлекательный шаблон.

Листинг 30.3 rtf.php - Сценарий для создания персонифицированного сертификата в RTF-формате

```
<?
   // проверка наличия всех необходимых параметров
   if ( !$name I I !$score )
   {
      echo "<h1>Error:</h1>This page was called incorrectly";
   }
   else
   {
      // генерация заголовков, призванных облегчить браузеру выбор
      // требуемого приложения
      header ( "Content-type: application/msword" ) ;
      header ( "Content-Disposition:inline, filename=cert . rtf");

      $date = date( "F d, Y" ) ;

      // открытие файла шаблона
      $filename = "PHPCertification.rtf";
      $fp = fopen ( $filename, "r" ) ;

      // считывание шаблона в переменную
      $output = fread( $fp, filesize( $filename ) ) ;

      fclose ( $fp ) ;

      // замещение заполнителей в шаблоне данными
      $output = str_replace ( "<<NAME>>", strtoupper( $name ) , $output ) ;
      $output = str_replace( "<<Name>>", $name, $output ) ;
      $output = str_replace( "<<score>>", $score, $output ) ;
      $output = str_replace( "<<mm/dd/yyyy>>", $date, $output ) ;

      // отправка сгенерированного документа в браузер
      echo$output;
   }
?>
```

Этот сценарий выполняет некоторую общую проверку на наличие ошибок, дабы убедиться в передаче ему всех сведений о пользователе, после чего приступает к созданию сертификата.

Сценарий создает вывод в виде RTF-файла, *а* не HTML-файла, поэтому необходимо предупредить об этом браузер пользователя. Важно, чтобы браузер мог предпринять попытку открытия файла с помощью подходящего приложения или отображал диалоговое окно типа Save As... (Сохранить как), если он не распознает расширение RTF-формата.

MIME-тип выводимого файла для отправки соответствующего HTTP-заголовка указывается с помощью PHP-функции **header()** следующим образом:

```
header ( "Content-type: application/msword" ) ;
header( "Content-Disposition: inline, filename=cert.rtf");
```

Первый заголовок уведомляет браузер, что мы отправляем файл Microsoft Word (наличие этого вспомогательного приложения для открытия RTF-файла наиболее вероятно, хотя это и не всегда так).

Второй заголовок указывает браузеру автоматически отобразить содержимое файла, при чем для него предлагается имя файла **cerf.rtf**. Именно это имя файла по умолчанию пользователь будет видеть, если попытается сохранить файл из своего браузера.

После того как заголовки отправлены, мы открываем и считываем RTF-файл шаблона в переменную **$output** и используем функцию **str_replace()** для замещения заполнителей фактическими данными, которые должны появляться в файле. Строка

```
Soutput = str_replace( "<<Name>>", $name, $output );
```

заменит все случаи присутствия заполнителя «**Name**» содержимым переменной **$name**.

После выполнения всех подстановок остается только повторить вывод в окне браузера. Пример результата выполнения этого сценария показан на рис. 30.5.

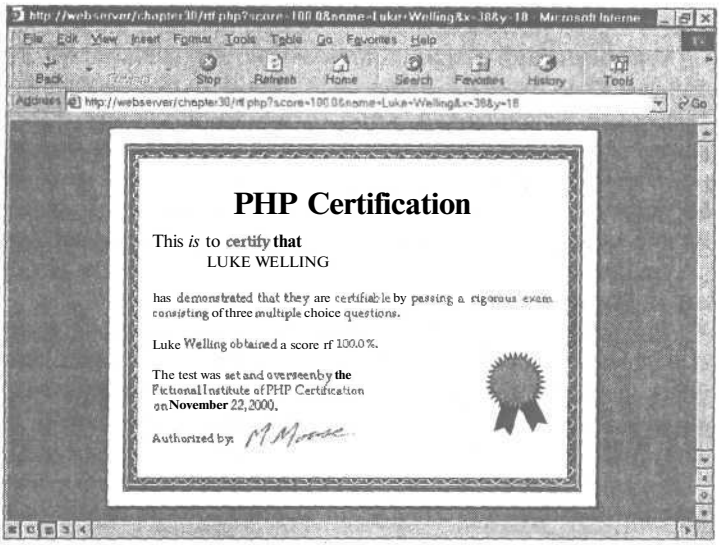


РИСУНОК 30.5 ***Сценарий rtf.php генерирует сертификат из шаблона в RTF-формате.***

Этот подход действует весьма успешно. Вызов функции **str_replace()** выполняется очень быстро, даже при том, что наш шаблон и, следовательно, содержимое переменной **$output** достаточно длинные. С точки зрения этого приложения основная проблема заключается в том, что для печати сертификата пользователю придется его загружать в текстовый процессор. Вероятно, это будет служить поощрением к модификации вывода. RTF-формат не позволяет создать документ "только для чтения".

Генерация PDF-сертификата из шаблона

Процесс генерации PDF-сертификата из шаблона во многом подобен описанному ранее. Основное различие состоит в том, что при создании PDF-файла некоторые заполнители могут быть спутаны с кодами форматирования. Например, если взглянуть на созданный шаблон сертификата, несложно заметить, что теперь заполнители выглядят следующим образом:

```
<<N)-13 (AME)-10 (>) -6 (>
<<Na)-9 (m)0 (e)-18 (>>
<) -11 (<) 1 <sc) -17 (or) -6 (в) -6 (>) -11 (>
<)-11(<)1 (т) -12 (т) 0 (/d) -6 (d)-19 (/) 1 (yy) -13 (yy) -13 (>>
```

Если вы просмотрите файл, то увидите, что в отличие от RTF, этот формат не является легко читабельным.

Существует несколько способов выхода из подобной ситуации.

Можно было бы просмотреть каждый из заполнителей и удалить коды форматирования. Это весьма незначительно сказалось бы на конечном виде документа, поскольку внедренные в предыдущий шаблон коды указывают, сколько места должно быть оставлено между буквами заполнителей, которые в любом случае будут замещаться. Однако при таком подходе придется вручную редактировать PDF-файл и повторять это при каждом его изменении или обновлении. Это не слишком трудно, когда приходится иметь дело только с четырьмя заполнителями, но превращается в настоящий кошмар, когда, например, имеется несколько документов со множеством заполнителей и требуется изменить заголовки во всех документах.

Этой проблемы можно избежать, прибегнув к другой технологии. Можно использовать Adobe Acrobat для создания PDF-формы, аналогичной HTML-форме с пустыми именованными полями. Затем можно воспользоваться PHP-сценарием для создания файла в формате FDF (Forms Data Format — формат данных форм), который представляет собой набор данных, объединяемых с шаблоном. FDF-файлы можно создать, используя библиотеку PHP-функций FDF: в частности, функцию **fdf_create()** для создания файла; функцию **fdf_set_value()** для определения значений полей; функцию **fdf_setfile()** для определения связанного с шаблоном файла формы. Затем этот файл можно передать обратно в браузер с соответствующим MIME-типом, в данном случае **vnd.fdf**, и подключаемый модуль браузера Acrobat Reader должен подставить данные в форму.

Это весьма искусный способ решения задачи, но он имеет два ограничения. Во-первых, предполагается, что у вас имеется копия Acrobat. Во-вторых, трудно заместить текст, который является встроенным, а не выглядящим как поле формы. Это может оказаться либо трудным, либо не очень, в зависимости от того, что вы пытаетесь сделать. Мы интенсивно использовали генерацию PDF-текста для создания писем, в которых очень многое должно замещаться непосредственно в тексте. FDF-файлы не очень хорошо подходят для этой цели. Однако, при интерактивном заполнении, например, налоговой формы, это не составляет особой проблемы.

Подробнее о FDF-формате можно прочесть на сайте компании Adobe:

```
http://www.adobe.com/support/techdocks/16196.htm
```

Если решено воспользоваться именно этим подходом, следует также просмотреть документацию по FDF в руководстве по PHP:

```
http://www.php.net/manual/ref.fdf.php
```

Теперь давайте обратимся к нашему PDF-решению ранее указанной проблемы.

Заполнители в PDF-файле все же можно найти и заменить, если известно, что коды дополнительного форматирования состоят исключительно из дефисов, цифр и круглых скобок и, следовательно, могут быть сопоставлены с применением регулярного выражения. Мы создали функцию **pdf_replace()** для автоматической генерации сопоставляющего регулярного выражения для заполнителя и для замены этого заполнителя соответствующим текстом.

За исключением этого добавления, код для генерации сертификата с помощью PDF-шаблона во многом подобен RTF-версии. Этот сценарий приведен в листинге 30.4.

Листинг 30.4 pdf.php — Сценарий для создания персонифицированного PDF-сертификата с помощью шаблона

```
<?
   set_time_limit ( 180 ); // этот сценарий может работать очень медленно
   function pdf_replace ( $pattern, $replacement, $string )
   {
      $len = strlen( $pattern );
      $regexp= '';
      for ( $i = 0; $i<$len; $i++ )
      {
         $regexp .= $pattern[$i];
         if ($i<$len-1)
            $regexp .= "(\)\-{0,1}[0-9]*\(){0,1}";
      }
      return ereg_replace ( $regexp, $replacement, $string );
   >
   if(!$name|| !$score)
   {
      echo "<h1>Error:</h1>This page was called incorrectly";
   }
   else
   {
      // генерация заголовков, помогающих браузеру
      // выбрать соответствующее приложение
      header ( "Content-Disposition: filename=cert.pdf");
      header ( "Content-type: application/pdf " );

      $date = date( "F d, Y" );

      // открытие файла шаблона
      $filename = "PHPCertification.pdf";
      $fp = fopen ( $filename, "r" );

      // считывание шаблона в переменную
      $output = fread( $fp, filesize ( $filename ) );

      fclose ( $fp );

      // замещение заполнителей в шаблоне данными
      $output = pdf_replace ( "<<NAME>>", strtoupper ( $name ) , $output );
      $output = pdf_replace ( "<<Name>>", $name, $output );
      $output = pdf_replace ( "<<score>>",$score, $output );
      $output = pdf_replace( "<<mm/dd/yyyy>>", $date, $output );

      // отправка сгенерированного документа в браузер
      echo $output;
   }
?>
```

Этот сценарий создает персонифицированную версию PDF-документа. Документ, показанный на рис. 30.6, будет надежно печататься в различных системах, и получателю трудно его модифицировать либо редактировать. Как видите, PDF-документ, показанный на рис. 30.6, выглядит почти так же, как RTF-документ, показанный на рис. 30.5

Единственная проблема при использовании этого подхода состоит в том, что код работает довольно-таки медленно, поскольку применяются регулярные выражения. Регулярные выражения обрабатываются значительно медленнее, чем функция **str_replace()**, которой можно было воспользоваться в RTF-версии.

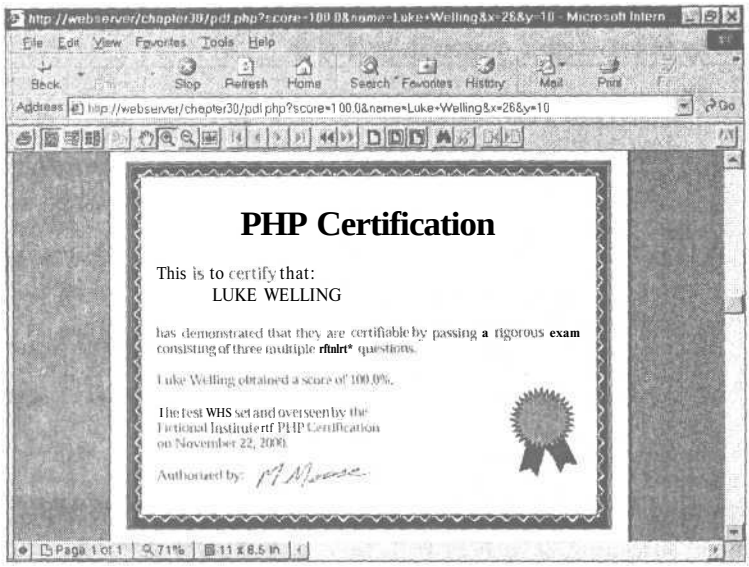


РИСУНОК 30.6 *Сценарии pdf.php генерирует сертификат из шаблона в PDF-формате.*

Если предстоит сопоставить большое количество заполнителей или нужно сгенерировать много таких документов на одном и том же сервере, возможно, стоит отдать предпочтение другим подходам. В случае более простого шаблона проблема стояла бы не так остро. А в этом файле значительная часть данных представляют изображения.

Генерация PDF-документа с использованием PDFlib

Библиотека функций PDFlib предназначена для генерации динамических PDF-документов из Web. Строго говоря, она является не частью PHP, но отдельной библиотекой, содержащей много функций, предназначенных для вызова из широкого числа языков программирования. Существуют привязки библиотеки к C, C++, Java, Perl, Python, Tcl и ActiveX/COM.

Интересно, что до сих пор нет официальной привязки PDFlib к PHP. Текущая привязка не задокументирована и не поддерживается PDFlib. В Web-сайте PDFlib говорится, что "Привязки к другим языкам, таким как PHP, будут поддерживаться в будущем", но это было сказано не менее года назад.

Хотя, возможно, официальная привязка к PHP и окажется лучше текущей, когда (или если) она появится, тем не менее, текущая привязка очень хороша. Единственная проблема заключается в том, что для ознакомления с подробным справочником по API потребуется прочесть документацию по PHP, которая доступна на сайте

```
http://www.php.net/manual/ref.pdf.php
```

дабы ознакомиться с обзором, инструкциями по установке и документацией PDFlib, доступной в виде файла **PDFlib-manual.pdf** в пакете PDFlib.

Сценарий *Hello World* для PDFlib

После того как PHP установлен с включенной поддержкой PDFlib, его можно протестировать с помощью простой программы наподобие примера Hello World, приведенного в листинге 30.5.

Листинг 30.5 testpdf.php — Классический пример Hello World, использующий PDFlib в среде PHP

```
<?
   // создание файла
   $fp = fopen("hello.pdf", "w") ;
   if(!$fp)
   {
      echo "Error: could not create the PDF file";
      exit;
   }

   // начало pdf-документа
   $pdf = pdf_open($fp) ;
   pdf_set_info($pdf, "Creator", "pdftest.php");
   pdf_set_info($pdf, "Author", "Luke Helling and Laura Thomson");
   pdf_set_info($pdf, "Title", "Hello World (PHP)");

   // печатный лист стандарта US letter имеет размеры 11" x 8.5"
   // и приблизительно 72 пункта на дюйм
   pdf_begin_page($pdf, 8.5*72, 11*72);
   pdf_add_outline($pdf, "Page 1");
   pdf_set_font($pdf, "Helvetica-Bold", 24, "host");
   pdf_set_text_pos($pdf, 50, 700);

   // написание текста
   pdf_show($pdf,"Hello,world!");
   pdf_continue_text($pdf,"(says PHP)");

   // конец документа
   pdf_end_page($pdf);
   pdf_close($pdf);
   fclose($fp);

   // отображение ссылки для загрузки
   echo "download the pdf <a href = 'hello.pdf'>here</a>";
?>
```

Наиболее вероятной ошибкой, которая может привести к неудачному выполнению этого сценария, является следующая:

```
Fatal error: Call to undefined function: pdf_open() in
/home/book/public_html/chapter30/pdftest.php on line 6
```

Это означает, что расширение PDFlib не было скомпилировано в PHP.

Установка достаточно проста, но некоторые нюансы могут различаться в зависимости от конкретных версий PHP и PDFlib. Для ознакомления с подробными предложениями по установке имеет смысл обратиться к замечаниям пользователей на странице PDFlib в упомянутом руководстве по PHP.

После установки и успешного выполнения этого сценария в своей системе можно приступить к его исследованию.

Первый раздел кода, включая строку

```
$fp = fopen("hell.pdf", "w");
```

создает доступный для записи файл. Стоит отметить, что в данном случае код выполняет запись непосредственно в текущий каталог несмотря на то, что мы уже приводили ряд причин, по которым не стоит разрешать PHP выполнять запись в пределах дерева Web-каталогов.

Строка

```
$pdf = pdf_open($fp);
```

инициализирует PDF-документ, используя уже открытый нами файл. Можно также вызвать функцию **pdf_open()** без параметров, чтобы создать документ в памяти с целью его вывода непосредственно в браузер. В любом случае потребуется перехватить возвращаемое значение из функции **pdf_open()**, поскольку его необходимо передавать в каждый последующий вызов любой функции PDF.

Функция **pdf_set_info()** позволяет снабдить документ дескриптором, содержащим тему, заголовок, имя создателя, автора, список ключевых слов и одно дополнительное поле, определяемое пользователем.

В приведенном ниже фрагменте кода мы указываем создателя документа, его автора и заголовок. Обратите внимание, что все шесть информационных полей являются необязательными.

```
pdf_set_info($pdf, "Creator", "pdftest.php");
pdf_set_info($pdf, "Author", "Luke Welling and Laura Thomson");
pdf_set_info($pdf, "Title", "Hello World (PHP)");
```

PDF-документ состоит из набора страниц. Чтобы начать новую страницу, следует вызвать функцию **pdf_begin_page()**. Как и идентификатору, возвращаемому функцией **pdf_open()**, функции **pdf_begin_page()** требуются размеры страницы. Каждая страница в документе может иметь свои размеры, но если только на то не существуют веские причины, следует использовать одинаковые размеры страниц.

И для размеров страниц, и для определения координат положений на странице в PDFlib используются пункты. Например, страница формата A4 имеет размеры, равные приблизительно 595 на 842 пунктов, а страница формата U.S. letter — 612 на 792 пунктов. Это означает, что строка

```
pdf_begin_page($pdf, 8.5*72, 11*72);
```

создает в документе страницу, размеры которой соответствуют формату U.S. letter.

PDF-документ не обязательно должен быть печатным. В документ могут быть внедрены многие свойства PDF, такие как гиперссылки и закладки. Функция **pdf_add_outline()** добавляет закладку в структуру документа. Закладки в документе будут отображаться в отдельной панели окна Acrobat Reader, позволяя переходить непосредственно к важным разделам.

Строка

```
pdf_add_outline($pdf, "Page 1");
```

добавляет запись структуры, помеченную меткой Page 1, которая будет ссылаться на текущую страницу.

Доступные в системе шрифты варьируются от одной операционной системы к другой, и даже от одного компьютера к другому. Набор основных шрифтов, работающих с любой программой чтения PDF-документов, обеспечит единообразие результатов. Вот 14 основных шрифтов:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique

- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats

В документы можно внедрять и другие шрифты, не входящие в этот набор, но это приведет к увеличению размера файла и может противоречить лицензионному соглашению на использование конкретного шрифта. Шрифт, его размер и кодировку символов можно выбрать следующим образом:

```
pdf_set_font($pdf, "Helvetica-Bold", 24, "host");
```

Размеры шрифтов указываются в пунктах. Нами была выбрана кодировка символов host. Допустимыми значениями этого параметра являются winansi, builtin, macroman, ebcdic или host. Они означают следующее:

- winansi — кодировка символов в соответствии со стандартом ISO 8859-1 плюс такие специальные символы, добавленных компанией Microsoft, как символ денежной единицы Евро.
- macroman — кодировка Mac Roman. Набор символов, устанавливаемый по умолчанию на компьютерах Macintosh.
- ebcdic — кодировка EBCDIC, используемая в системах AS/400 от IBM.
- builtin — использование кодировки, встроенной в шрифт. Обычно используется с шрифтами и символами, которые отличаются от латинских.
- host — автоматически выбирает macroman в системе Mac, ebcdic в системе, использующей кодировку EBCDIC, и winansi во всех остальных системах.

Если вставка специальных символов не требуется, выбор кодировки роли не играет.

PDF-документ не похож на HTML-документ или на документ текстового процессора. Текст не начинается просто с верхнего левого угла и не продолжается при необходимости в других строках. Нужно выбрать размещение каждой строки текста. Как уже упоминалось, для указания местоположения в PDF используются пункты. Начало координат (x, y [0, 0]) располагается в нижнем левом углу страницы.

Если используемая страница имеет размеры 612 на 792 пункта, то точка (50, 700) находится на расстоянии приблизительно двух третьих дюйма от левого края страницы и около одного и одной третьей дюйма от верхнего края. Чтобы расположить текст в этой позиции, используется строка

```
pdf_set_text_pos($pdf, 50, 700);
```

И, наконец, определив параметры страницы, можно поместить в нее некоторый текст. Для добавления текста в текущую позицию с использованием текущего шрифта применяется функция **pdf_show()**.

Строка

```
pdf_show($pdf, "Hello, World!");
```

добавляет в документ текст "Hello, World!"

Для перехода на следующую строку и записи нового текста используется функция **pdf_continue_text()**. Так, приведенный ниже код добавляет строку "(says PHP)":

```
pdf_continue_text($pdf, "(says PHP)");
```

Точное расположение этого текста будет зависеть от выбранных шрифта и размеров.

После завершения добавления элементов на страницу следует обратиться к функции **pdf_end_page()**:

```
pdf_end_page($pdf);
```

По завершении создания всего PDF-документа его нужно закрыть с помощью функции **pdf_close()**. После генерации файла его так же необходимо закрыть.

Строки

```
pdf_close($pdf);
fclose($fp);
```

завершают генерацию документа Hello World. Остается только обеспечить возможность его загрузки.

```
echo "download the pdf <a href = 'hello.pdf'>here</a>";
```

Этот пример был взят из примера для языка C, приведенного в документации PDFlib, и должен послужить полезной отправной точкой.

Документ, который требуется создать для сертификата, более сложен и содержит рамку, векторное и растровое изображения. При использовании двух других технологий мы добавили эти элементы с помощью текстового процессора. При использовании PDFlib их необходимо добавить вручную.

Генерация сертификата с помощью *PDFlib*

Чтобы использовать PDFlib, пришлось пойти на некоторые компромиссы. Хотя почти наверняка можно продублировать сертификат, который использовался ранее, для создания макета документа гораздо большие усилия потребовались бы при генерации и позиционировании каждого элемента вручную, чем при использовании такого инструмента, как Microsoft Word.

В этом случае применяется тот же текст, что и ранее, включая красную розетку и растровое изображение подписи, но мы не собираемся дублировать сложную рамку.

Полный исходный текст этого сценария приведен в листинге 30.6.

Листинг 30.6 pdflib.php — Генерация сертификата с помощью PDFlib

```
<?
   if(!$name||!$score)
   {
      echo "<h1>Error:</h1>This page was called incorrectly";
   }
   else
```

```
{
  //генерация заголовков, помогающих браузеру выбрать
  //соответствующее приложение
  header( "Content-type: application/pdf" );
  header( "Content-Disposition: filename=cert.pdf");

  $date = date( "F d, Y" );

  // создание pdf-документа в памяти
  $pdf = pdf_open();

  // определение размеров страницы в пунктах
  // страница US letter имеет размеры 11" x 8.5"
  // в дюйме приблизительно 72 пункта
  $width = 11*72;
  $height = 8.5*72;

  pdf_begin_page($pdf, $width, $height);

  // рисование рамок

  $inset = 20;   // расстояние между рамкой и краем страницы
  $border = 10;  // толщина линии основной райки
  $inner = 2;    // просвет внутри рамки

  //рисование внешней рамки
  pdf_rect($pdf, $inset-$inner,
                 $inset-$inner,
                 $width-2*($inset-$inner),
                 $height-2*($inset-$inner));
  pdf_stroke($pdf);

  //рисование основной рамки толщиной $border пунктов
  pdf_setlinewidth($pdf, $border);
  pdf_rect($pdf, $inset+$border/2,
                 $inset+$border/2,
                 $width-2*($inset+$border/2),
                 $height-2*($inset+$border/2));
  pdf_stroke($pdf);
  pdf_setlinewidth($pdf, 1.0);

  //рисование внутренней рамки
  pdf_rect($pdf, $inset+$border+$inner,
                 $inset+$border+$inner,
                 $width-2*($inset+$border+$inner),
                 $height-2*($inset+$border+$inner));
  pdf_stroke($pdf);

  //добавление текста
  pdf_set_font($pdf, "Times-Roman", 48, "host");

  $startx = ($width - pdf_stringwidth($pdf, "PHP Certification"))/2;
  pdf_show_xy($pdf, "PHP Certification", $startx, 490);

  pdf_set_font($pdf, "Times-Roman", 26, "host");
  $startx = 70;

  pdf_show_xy($pdf, "This is to certify that:", $startx, 430);
  pdf_show_xy($pdf, strtoupper($name), $startx+90, 391);

  pdf_set_font($pdf, "Times-Roman", 20, "host");

  pdf_show_xy($pdf, "has demonstrated that they are certifiable ".
                    "by passing a rigorous exam", $startx, 340);
  pdf_show_xy($pdf, "consisting of three multiple choice questions.",
                    $startx, 310);

  pdf_show_xy($pdf, "$name obtained a score of $score"."%.",
                    $startx, 260);
```

```
pdf_show_xy($pdf, "The test was set and overseen by the ",
                  $startx, 210);
pdf_show_xy($pdf, "Fictional Institute of PHP Certification",
                  $startx, 180);
pdf_show_xy($pdf, "on $date.", $startx, 150);

pdf_show_xy($pdf, "Authorised by:", $startx, 100);

// добавление растрового изображения подписи
$signature = pdf_open_image_file($pdf, "tiff",
                        "/htdocs/book/chapter30/signature.tif");
pdf_place_image($pdf, $signature, 200, 75, 1);
pdf_close_image($pdf, $signature);

pdf_setrgbcolor_fill ($pdf, 0, 0, .4); //dark blue

pdf_setrgbcolor_stroke($pdf, 0, 0, 0);  // black

// рисование ленточки 1
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_fill($pdf);

// рисование контура ленточки 1
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_stroke($pdf);

// рисование ленточки 2
pdf_moveto($pdf, 660, 150);
pdf_lineto($pdf, 680, 49);
pdf_lineto($pdf, 695, 69);
pdf_lineto($pdf, 716, 55);
pdf_lineto($pdf, 696, 150);
pdf_closepath($pdf);
pdf_fill($pdf);

// рисование контура ленточки 2
pdf_moveto($pdf, 660, 150);
pdf_lineto($pdf, 680, 49);
pdf_lineto($pdf, 695, 69);
pdf_lineto($pdf, 716, 55);
pdf_lineto($pdf, 696, 150);
pdf_closepath($pdf);
pdf_stroke($pdf);

pdf_setrgbcolor_fill($pdf, .8, 0, 0);  //red

//рисование розетки
draw_star(665, 175, 32, 57, 10, $pdf, true);

//рисование контура розетки
draw_star(665, 175, 32, 57, 10, $pdf, false);

pdf_end_page($pdf);
pdf_close($pdf);

}
```

```
function draw_star($centerx, $centery, $points, $radius,
                   $point_size, $pdf, $filled)
{
   $inner_radius = $radius-$point_size;
   for ($i = 0; $i<=$points*2; $i++ )
   {
      $angle= ($i*2*pi())/($points*2);
      if ($i%2)
      {
         $x = $radius*cos($angle) + $centerx;
         $y = $radius*sin($angle) + $centery;
      }
      else
      {
         $x = $inner_radius*cos($angle) + $centerx;
         $y = $inner_radius*sin($angle) + $centery;
      }
      if ($i=0)
        pdf_moveto($pdf, $x, $y) ;
      else if($i==$points*2)
        pdf_closepath($pdf);
      else
        pdf_lineto($pdf, $x, $y) ;
   }
   if ($filled)
     pdf_fill($pdf);
   pdf_stroke($pdf);
}
?>
```

Созданный этим сценарием сертификат показан на рис. 30.7. Как видите, он очень похож на созданные ранее сертификаты, за исключением того, что его рамка проще, а розетка выглядит несколько иначе. Это связано с тем, что они были нарисованы непосредственно в документе, а не с использованием существующего файла с рисунком.

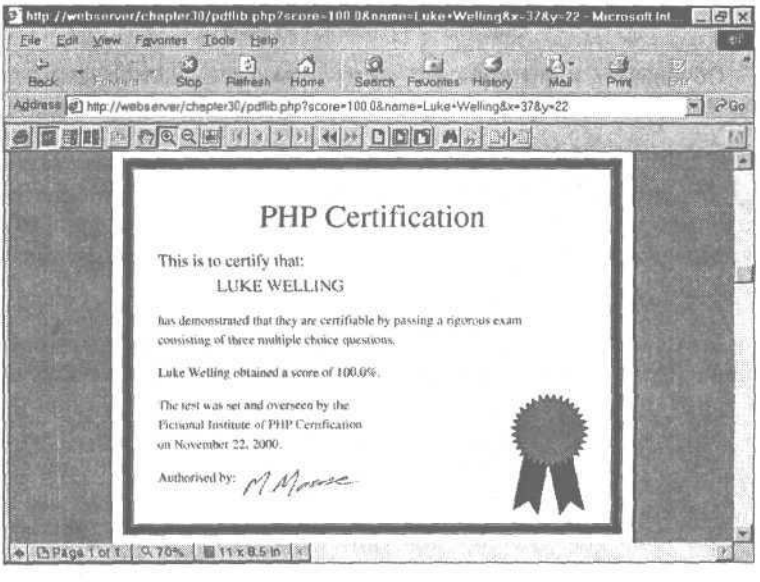


РИСУНОК 30.7 *Сценарий **pdflib.php** выполняет рисование графических элементов сертификата в PDF-документе.*

Мы рассмотрим некоторые части данного сценария, которые отличаются от приведенных ранее примеров.

Посетителям требуется, чтобы в сертификате отображались их персональные данные, поэтому документ будет создаваться в памяти, а не в файле. Если бы он был записан в файл, следовало бы позаботиться о механизмах создания уникальных имен файлов, предотвратить заглядывание в чужие сертификаты и определить способ удаления устаревших файлов сертификатов для освобождения объема жесткого диска сервера. Для создания документа в памяти вызывается функция **pdf_open()** без параметров:

```
spdf = pdf_open();
```

Упрощенная рамка будет состоять из трех рамок: жирной и двух тонких, одна из которых располагается снаружи основной, а другая — внутри. Все эти рамки рисуются как прямоугольники.

Для размещения рамок так, чтобы можно было легко изменять размеры страницы или внешний вид рамок, позиции всех рамок будут основаны на уже существующих переменных **$width** и **$height**, а также на нескольких других: **$inset**, **$border** и **$inner**. Переменная **$inset** будет использоваться для указания расстояния между рамкой и краем страницы в пунктах, **$border** — для указания толщины основной рамки, а **$inner** — для указания ширины просвета между основной и тонкими рамками.

Если читателям приходилось выполнять рисование с помощью другого графического API, рисование с помощью PDFlib преподнесет им несколько сюрпризов. Если вы не прочли главу 19, возможно, стоит сделать это, поскольку рисование изображений с помощью библиотеки **gd** весьма сходно с рисованием с помощью PDFlib.

Тонкие рамки не представляют особой сложности. Для создания прямоугольника мы используем функцию **pdf_rect()**, которая в качестве параметров требует передачи идентификатора PDF-документа, координат x и y нижнего левого угла прямоугольника и ширины и высоты прямоугольника. Поскольку требуется обеспечить гибкость макета, эти значения вычисляются из определенных нами переменных.

```
pdf_rect($pdf, $inset-$inner,
               $inset-$inner,
               $width-2*($inset-$inner),
               $height-2*($inset-$inner));
```

Обращение к функции **pdf_rect()** определяет контур формы прямоугольника. Чтобы вычертить эту форму, нужно вызвать функцию **pdf_stroke()**;

```
pdf_stroke($pdf);
```

Для рисования основной рамки потребуется указать толщину линии. По умолчанию толщина линии равна 1 пункту. Следующее обращение к функции **pdf_setlinewidth()** установит ее равной **$border** пунктам (в данном случае — 10):

```
pdf_setlinewidth($pdf, $border);
```

После установки толщины мы снова создаем прямоугольник с помощью функции **pdf_rect()** и вызываем функцию **pdf_stroke()**, чтобы его нарисовать.

```
pdf_rect($pdf, $inset+$border/2,
               $inset+$border/2,
               $width-2*($inset+$border/2),
               $height-2*($inset+$border/2));
pdf_stroke($pdf);
```

После того как толстая линия нарисована, нужно не забыть снова установить толщину линии равной 1 пункту:

```
pdf_setlinewidth($pdf, 1.0);
```

Мы собираемся использовать функцию **pdf_show_xy()** для размещения каждой строки текста в сертификате. Для большинства строк текста применяется настраиваемая левая граница (**$startx**) в качестве координаты x и выбранное на глаз значение в качестве координаты y. Поскольку требуется, чтобы заголовок располагался по центру относительно боковых сторон страницы, для размещения его левого края нужно знать ширину заголовка. Эту ширину можно получить, используя функцию **pdf_stringwidth()**. Вызов функции

```
pdf_stringwidth($pdf, "PHP Certification");
```

вернет ширину строки "PHP Certification" для текущего шрифта и размера.

Как это было сделано с другими версиями сертификата, подпись будет вставлена в него в виде сосканированного растрового изображения. Следующих три оператора

```
$signature = pdf_open_image_file($pdf, "tiff",
                                 "/htdocs/book/chapter30/signature.tif");
pdf_place_image($pdf, $signature, 200, 75, 1);
pdf_close_image($pdf, $signature);
```

откроют TIFF-файл, содержащий изображение подписи, добавят изображение в указанную позицию на странице и закроют TIFF-файл. Можно использовать также и другие типы файлов. Единственный параметр, который, возможно, непонятен — пятый параметр функции **pdf_place_image()**. Эта функция не ограничивается вставкой изображения с его исходными размерами. Пятый параметр является коэффициентом масштабирования. Мы решили отобразить изображение в масштабе 1:1 и поэтому использовали коэффициент масштабирования, равный 1, однако можно было бы выбрать и большее число с целью увеличения изображения или дробное число для его уменьшения.

Наибольшую трудность при вставке в сертификат с использованием библиотеки PDFlib представляет розетка. Мы не можем автоматически открыть и вставить в документ уже имеющийся Windows-метафайл с изображением розетки, но можем нарисовать любые формы.

Чтобы нарисовать закрашенную форму типа ленточек, можно записать следующий код.

В приведенных ниже строках мы устанавливаем цвет штриха (или линии) черным, а цвет заливки (или цвет внутренней части формы) темно-синим:

```
pdf_setrgbcolor_fill($pdf, 0, 0, .4);    // темно-синий
pdf_setrgbcolor_stroke($pdf, 0, 0, 0);    // черный
```

Затем мы создаем пятиугольник, который будет одной из ленточек, и выполняем его заливку:

```
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_fill($pdf);
```

Поскольку желательно, чтобы многоугольник имел четкий контур, нужно еще раз определить этот же путь, но на сей раз вызвать функцию **pdf_stroke()**, **а не pdf_fill()**.

Поскольку многолучевая звезда является сложной повторяющейся формой, мы создали функцию для вычисления точек вдоль пути. Эта функция называется **draw_star()** и требует передачи координат x и y центра, требуемого количества лучей, длины лучей, идентификатора PDF-документа, булевого значения для указания того, должна ли форма звезды быть залитой или просто контуром.

Функция **draw_star()** использует некоторые основные тригонометрические соотношения для вычисления расположения ряда лучей, образующих звезду. Для каждого луча мы определяем точку на радиусе звезды и точку на меньшей окружности, расположенной на расстоянии **$point_size** внутри внешней окружности, после чего рисуем линию между ними. Следует отметить, что тригонометрические функции PHP типа **cos()** и **sin()** работают с радианами, а не с градусами.

Используя эту функцию и некоторые математические соотношения, можно аккуратно сгенерировать сложную повторяющуюся форму. Если бы для рамки страницы требовался сложный узор, можно было бы воспользоваться аналогичным подходом.

После того как все элементы страницы сгенерированы, необходимо завершить страницу и документ.

Проблемы, связанные с заголовками

Один незначительный нюанс, который следует отметить во всех этих сценариях, заключается в том, что браузеру потребуется указать, какой тип данных мы собираемся ему отправить. Это делается через HTTP-заголовок **content-type:**

```
header( "Content-type: application/msword" );
```

ИЛИ

```
header( "Content-type: application/pdf" ) ;
```

Следует иметь в виду, что браузеры обрабатывают эти заголовки не слишком последовательно. В частности, Internet Explorer зачастую игнорирует MIME-тип и предпринимает попытку автоматического определения типа файла.

Некоторые заголовки, похоже, вступают в противоречие с заголовками управления сеансом. Для решения этой проблемы существует несколько путей. Мы установили, что использование параметров **GET** вместо параметров **POST** или параметров переменных сеанса позволяет избежать проблемы.

Еще одно решение — не использовать встроенный PDF-текст, а предоставлять пользователю загружать его в отличие от того, как показано в примере "Hello, World!" с использованием PDFlib.

Проблем можно также избежать, если написать две несколько различающиеся версии кода: одну для Netscape, а другую для Internet Explorer.

Существует также проблема с объединением PDF-файлов, динамически сгенерированных Internet Explorer, и подключаемой программой Acrobat Reader. Технические замечания Adobe по этому вопросу можно найти в документе

```
http://www.adobe.com/support/techdocs/3d76.htm
```

Хотя в этом документе говорится о генерировании PDF-документов с помощью Active Server Pages, проблема остается той же самой.

Расширение проекта

Добавление некоторых более реалистичных экзаменационных задач, очевидно, могло бы расширить этот проект, но он действительно задумывался только как пример доставкидокументов.

Персонифицированными документами, которые может потребоваться доставлять интерактивно, могут быть юридические документы, частично заполненные формы заказов или заявлений, или формы, требуемые правительственными учреждениями.

Дополнительная информация

Дополнительная информация, касающаяся форматов PDF (и FDF), может быть найдена на сайте компании Adobe по адресу:

```
http://www.adobe.com
```

Часть 6

Приложения

В этой части:

A Инсталляция **PHP 4 и MySQL**

B **Ресурсы Internet**

Приложение A

Инсталляция PHP 4 и MySQL

Существуют версии Apache, PHP и MySQL для многих операционных систем и Web-серверов. В этом приложении приводится методика установки Apache, PHP и MySQL на различных серверных платформах. Кроме того, рассматриваются наиболее распространенные опции для UNIX и Windows NT.

Тематика данного приложения включает в себя:

- Запуск PHP в качестве интерпретатора CGI или модуля
- Установка Apache, SSL, PHP и MySQL под UNIX
- Установка Apache, PHP и MySQL под Windows
- Тестирование работоспособности PHP: **phpinfo()**
- Присоединение PHP и MySQL к Internet Information Server
- Присоединение PHP и MySQL к Personal Web Server
- Исследование различных конфигураций

Целью этого приложения является предоставление руководства по инсталляции Web-сервера, полезного для тех, кто занимается хостингом Web-сайтов. Некоторые сайты, как было показано на примерах, требуют использования слоя безопасных сокетов (Secure Socket Layer, SSL) для проведения коммерческих операций. Большинством из них управляют сценарии, соединяющиеся с сервером баз данных (DB), запрашивающие и обрабатывающие данные. Авторы избрали для решения задач такого класса Apache, PHP и MySQL по таким показателям, как цена, надежность, производительность, простота интеграции и функциональность.

Запуск PHP в качестве интерпретатора CG1 или модуля

PHP — это простой, но мощный встраиваемый в HTML интерпретируемый язык серверной стороны, который позволяет обрабатывать файлы, запускать команды и открывать сетевые соединения. Интерпретатор можно запустить как модуль или как отдельный двоичный файл CGI. В большинстве случаев модульный вариант выбирают из соображений повышения производительности. CGI-версия позволяет пользователям Apache запускать PHP-сценарии с различных страниц под разными идентификаторами. Хотя последний вариант по умолчанию считается менее защищенным, PHP разработан так, что CGI-программы, написанные на нем, все же безопаснее аналогичных программ на языках Perl или C.

PHP имеет набор конфигурационных опций, предназначенных для подбора необходимой в каждом конкретном случае комбинации безопасности и удобства использования. Однако тем, кто предпочитает запускать PHP через CGI, следует ознакомиться с советом CERT CA-96.11 (CERT Advisory CA-96.11) по адресу:

```
http://www.cert.org/advisories/CA-96.11.interpreters_in_cgi_bin_dir.html
```

Установки по умолчанию для CGI требуют, чтобы выполняемый PHP-файл находился на Web-сервере в каталоге **cgi-bin**, тогда как CERT этот метод критикует. Причиной служит то, что большинство интерпретаторов (но не PHP), доступных в каталоге **cgi-bin,** позволяют удаленным пользователям запускать любые команды, которые сам интерпретатор способен выполнять на сервере. Тем не менее, PHP не позволяет взломщиками воспользоваться этой брешью.

Справиться с этой задачей PHP помогает следующее:

- PHP отказывается интерпретировать аргументы командной строки, если он используется через CGI.
- PHP предотвращает доступ к каким-либо Web-документам без проверки прав доступа, если он запускается как выполняемый файл CGI. Однако следует включить опцию **--enable-force-cgi-redirect** и установить конфигурационные директивы времени выполнения **doc_root и user_dir**.

Разумеется, речь идет о конфигурации по умолчанию.

Если используется метод CGI, можно выбрать еще более безопасную возможность — разместить выполняемый PHP-файл за пределами Web-дерева. Типичным положением файла может быть каталог **/usr/local/bin** под UNIX или **c:\ PHP-DIR** под Windows. Обратной стороной медали является то, что строку для запуска интерпретатора PHP потребуется помещать во все CGI-сценарии. Например, на UNIX-машине с интерпретатором PHP в каталоге **/usr/local/bin/php** первой строкой каждого сценария должна быть строка

```
#! /usr/local/bin/php
```

Кроме того, чтобы сделать сценарий запускаемым, необходимо установить соответствующие права доступа. Короче говоря, в случае CGI к PHP-сценарию следует относиться так же, как к сценарию, написанному на Perl.

В этом приложении рассказано о применении метода модуля для запуска PHP в среде UNIX и метода CGI для запуска в среде Windows.

Установка Apache, PHP и MySQL под UNIX

Начнем с установки Apache, PHP и MySQL в среде UNIX. Вначале необходимо решить, какие дополнительные модули могут потребоваться в дальнейшем. Поскольку некоторые из примеров в этой книге используют безопасный сервер для Web-транзакций, потребуется установить сервер с поддержкой слоя безопасных сокетов (Secure Socket Layer, SSL). Установка PHP здесь будет совпадать с настройкой по умолчанию, но будет рассказано и о подключении следующих дополнительных библиотек в PHP.

- **http://curl.haxx.se/**: Функции библиотеки Client URL
- **http://pspell.sourceforge.net/**: Переносимые библиотеки проверки правописания (Portable Spell Checker Libraries)
- **http://www.pdflib.com/pdflib/index.html**: Библиотека для создания документов PDF на лету

Это лишь три из великого множества доступных PHP-библиотек. На примере их подключения вполне можно понять, как следует добавлять дополнительные библиотеки в PHP. До компиляции модуля PHP рекомендуется установить на машине все необходимые библиотеки.

Загрузка и установка библиотечных программ уже обсуждались в соответствующих главах, и здесь этому внимание не уделяется. Здесь будет рассказано о том, как подключить их в PHP на этапе компиляции.

Инсталляция будет выполняться на сервере под управлением Red Hat Linux 6.2, однако схема является настоль общей, что ее можно применять под любым UNIX-сервером.

Итак, вначале перечислим средства, задействованные в процессе установки.

- Apache (**http://www.apache.org/**): Web-сервер
- Mod_SSL (**http://www.modssl.org/**): Модуль слоя безопасных сокетов
- OpenSSL (http://www.openssl.org/): Open Source Toolkit (необходимый для Mod_SSL)
- RSARef(**http://ftpsearch.lycos.com/**): Требуется только тем, кто находится в пределах США
- MySQL (**http://www.mysql.com/**): Реляционная база данных
- **PHP** (**http://www.php.net/**): Интерпретируемый язык серверной стороны

Далее будет предполагаться, что пользователь, производящий инсталляцию, имеет полный (root) доступ к серверу, на котором установлены следующие пакеты:

- Perl (желательно последней версии)
- gzip или gunzip
- gcc и GNU make

При отсутствии этих инструментальных средств их необходимо установить, иначе все последующие шаги потеряют смысл.

При отсутствии доступа root есть следующие возможности:

- Попросить системного администратора установить PHP
- Установить и запускать механизм PHP в собственном пользовательском каталоге **cgi-bin**
- Запустить собственный Web-сервер с поддержкой PHP на нестандартном порту

Однако, здесь рассматривается случай, когда пользователь имеет доступ **root.**

Когда все готово к инсталляции, следует загрузить все tar-файлы с исходными кодами во временный каталог. Здесь стоит убедиться, что в этом разделе имеется достаточно свободного дискового пространства. В данном случае временным каталогом является **/tmp/download**. Во избежание проблем с правами доступа к файлам, их необходимо загрузить от имени пользователя root.

Для установки пакетов следует выбрать такие каталоги:

- **/usr/local/apache**
- **/usr/local/mysql**
- **/usr/local/ssl**

Можно использовать и другие каталоги, изменив перед инсталляцией опцию prefix.

Итак, инсталляция начинается! Прежде всего, следует зарегистрироваться в системе как пользователь root

```
$ su
```

и ввести пароль. Затем перейти в каталог, содержащий файлы с исходным кодом, например

```
# cd /tmp/download/
```

Развернуть файлы с помощью команды:

```
#  gunzip -c mysql-3.22.xx.tar.gz | tar xvf -
```

Перейти в каталог, созданный при разархивировании:

```
#  cd mysql-3.22.xx
```

Теперь можно начать конфигурирование сервера MySQL. Утилита **configure** имеет немало опций, просмотреть которые позволяет команда **configure --help**. Сценарий **configure** проверяет версии компиляторов и другие системные настройки. Если при запуске он выдает ошибки, их можно просмотреть в файле **config.cache**.

```
t  ./configure --prefix=/usr/local/mysql
```

После окончания работы **configure** можно скомпилировать двоичные файлы, запустив команду (ее выполнение занимает некоторое время):

```
#  make
```

Теперь все готово для инсталляции двоичных файлов, которую можно произвести (каталог был задан опцией **configure** *--prefix*) командой:

```
#  make install
```

После этого самое время создать таблицы **mysql**, используемые для определения прав доступа. Строку **new-password** следует заменить чем-либо, иначе значением **new-password** будет текущий пароль пользователя root.

```
#  scripts/mysql_install_db
#  cd /usr/local/mysql/bin
# ./safe_mysqld &
# ./mysqladmin -u root password 'new-password'
```

Проверить работоспособность MySQL можно запуском нескольких простых тестов. При этом вывод на экран должен быть похож на показанный ниже:

```
#  /usr/local/mysql/bin/mysqlshow -p
```

Введите пароль.

```
+------------------+
I  Databases       I
+------------------+
I  mysql           |
+------------------+
```

При инсталляции пакета MySQL автоматически создаются две базы данных. Одна содержит таблицу, управляющую пользователями, хостами и правами доступа к базам данных на реальном сервере. Вторая является тестовой. Проверить базу данных можно, задав следующую командную строку:

***i* mysql -u root -p**

Введите пароль.

```
reysql> show databases;
+--------------------+
|  Database          I
+--------------------+
|  mysql             |
I  test              |
+--------------------+
2 rows in set (0.00 sec)
```

Теперь пришло время установить PHP. Здесь снова требуются права доступа пользователя root (можно, как и ранее, воспользоваться командой ***su***).

PHP требует предварительного конфигурирования Apache, чтобы местонахождение всех элементов было известным. К этому следует вернуться, что и произойдет в разделе, посвященном настройке сервера Apache. Далее необходимо перейти обратно в каталог, содержащий файлы с исходным кодом.

```
# cd /tmp/download
# gun zip -c apache_1 . 3 . x . tar . gz | tar xf -
# cd apache_1 .3.x
#  ./configure--pref ix=/usr/local/apache
# cd  ..
```

Итак, можно приступить к установке PHP. Разархивирование файлов с исходным кодом и переход в инсталляционный каталог достигаются командами:

```
# gunzip -c php-4.0.x.tar.gz | tar xvf -
f  cd php-4.0.x
```

Утилита **configure** пакета PHP также имеет опции, просмотреть которые можно командой **configure --help**. В данном случае используются те из них, которые добавляют поддержку MySQL, Apache, PDFLib, cURL и PSPELL.

Обратите внимание, что ниже приведена одна команда. Ее можно разместить в одной строке или, как показано в примере, используя символ продолжения (символ обратной косой черты (\)), набрать в нескольких строках, что улучшает читаемость.

```
f ./configure --with-mysql=/usr/local/mysql  \
     -- with-xml -- with-apache=. ./apache_1.3.x\
     -- with-curl=/usr/local/curl \
     -- with-pspell=/usr/local/pspell \
--enable-shared-pdflib --enable-track-vars
```

После этого (в случае отсутствия ошибок) производится компиляция и установка двоичных файлов:

```
# make
# make install
```

Конфигурационный ini-файл необходимо скопировать в каталог **lib:**

```
# cp php.ini-dist /usr/local/lib/php.ini
```

Этот файл можно редактировать, изменяя опции PHP. Например, увеличить время выполнения сценария можно, изменив значение директивы **max_execution_time** в файле **php.ini** следующим образом:

```
max_execution_time = 60;
```

Apache и mod_SSL

Пришло время сконфигурировать и установить модуль mod_SSL и Apache. Если вы находитесь в пределах США, вам потребуются файлы rsaref-2.0. К сожалению, поскольку их распространением больше не занимается RSA, они не имеют стабильной Web-страницы. Поэтому необходимо воспользоваться поисковым механизмом, например, Lycos

```
http://ftpsearch.lycos.com
```

или Google

```
http://www.google.com
```

для поиска файла **rsaref20.tar.Z**. Следует убедиться, что найден именно дистрибутив для UNIX.

Для извлечения файлов создается каталог **rsaref.** Здесь предполагается, что файл находится во временном каталоге, где хранится исходный код.

```
# mkdir rsaref-2.0
# cd rsaref-2.0
# gunzip -c ../rsaref20.tar.Z | tar xvf -
```

Теперь следует сконфигурировать и скомпилировать библиотеку OpenSSL, причем пользователям из США ее следует компилировать в сочетании с библиотекой RSAref.

```
# cd rsaref-2.0
# cp -rp install/unix local
f cd local
# make
# mv rsaref.a librsaref.a
# cd ../..
```

Далее потребуется настроить OpenSSL. Ниже приведен набор команд для создания временных сертификатов и CSR файлов. Опция **--prefix** задает каталог для инсталляции.

ПРИМЕЧАНИЕ

Пользователям в пределах США достаточно воспользоваться лишь командной строкой **с -L'pwd'/../rsaref-2.0/local/rsaref -fPIC.**

```
# gunzip c openssl-0.9.x.tar.gz | tar xvf -
# cd openssl-0.9.x
# ./config --prefix=/usr/local/ssl \
-L`pwd`/../rsaref-2.0/local/rsaref -fPIC
```

Компиляцию, тестирование и установку обеспечивает набор команд:

```
# make
# make test
# make install
# cd ..
```

Модуль mod_SSL конфигурируется как загружаемый модуль Apache.

```
# gunzip -c mod_ssl-2.6.x.tar.gz | tar xvf -
# od mod_ssl-2.5.x-1.3.x
# ./configure --with-apache=../apache_1.3.x
# od ..
```

Обратите внимание, что к исходному дереву Apache можно добавлять модули. Дополнительная опция **--enable-shared=ssl** позволяет скомпилировать mod_SSL как DSO (динамическую библиотеку разделенного доступа), **'libssl.so'**. Более подробная информация о поддержке DSO в Apache содержится в документах INSTALL и htdocs/manual/dso.html в дереве исходного кода Apache. Internet-провайдерам и разработчикам, поддерживающим пакет, рекомендуется использовать DSO для достижения максимальной гибкости от mod_SSL. Следует отметить, что Apache поддерживает DSO не на всех платформах.

```
# cd apache_1.3.x
# SSL_BASE=. /openssl-0.9.x \
RSA_BASE=../rsaref-2.0/local \
     ./configure \
     --enable-module=ssl \
     --activate-module=src/modules/php4/libphp4.a \
     --enable-module=php4 \
     --prefix=/usr/local/apache \
     --enable-shared=ssl
     [...здесь можно добавить и другие опции...]
```

(Переменные SSL_BASE и RSA_BASE можно установить как переменные окружения.)

В заключение выполняется компиляция Apache и сертификатов с последующей их установкой.

```
# make
```

Если все прошло успешно, на экран выводится следующее сообщение:

```
+-------------------------------------------------------------------+
| Before you install the package you now should prepare the SSL     |
| certificate system by running the 'make certificate' command.     |
| For different situations the following variants are provided:     |
|                                                                   |
| % make certificate TYPE=dummy (dummy self-signed Snake Oil cert)  |
| % make certificate TYPE=test (test cert signed by Snake Oil CA)   |
| % make certificate TYPE=custom (custom cert signed by own CA)     |
| % make certificate TYPE=existing (existing cert)                  |
| CRT=/path/to/your.crt [KEY=/path/to/your.key]                     |
| Use TYPE=dummy when you're a vendor package maintainer,           |
| the TYPE=test when you're an admin but want to do tests only,     |
| the TYPE=custom when you're an admin willing to run a real server |
| and TYPE=existing when you're an admin who upgrades a server.     |
| (The default is TYPE=test)                                        |
|                                                                   |
| Additionally add ALGO=RSA (default) or ALGO=DSA to select         |
| the signature algorithm used for the generated certificate.       |
```

```
I Use 'make certificate VIEW=1' to display the generated data.          |
| Thanks for using Apache & mod_ssl. Ralf S. Engelschall                I
| rse@engelschall.com   -   www.engelschall.com                         I
+-----------------------------------------------------------------------+
```

После этого можно создать индивидуальный сертификат. Эта опция запрашивает расположение, компанию и ряд других элементов.

```
# make certificate TYPE=custom
```

Установка Apache осуществляется командой:

```
# make install
```

Если вновь все прошло успешно, на экране появится сообщение:

```
+-----------------------------------------------------------------------+
| You now have successfully built and installed the                     |
| Apache 1.3 HTTP server. To verify that Apache actually                |
| works correctly you now should first check the                        |
|  (initially created or preserved) configuration files                 |
|                                                                       |
| /usr/local/apache/conf/httpd.conf                                     |
| and then you should be able to immediately fire up                    |
| Apache the first time by running:                                     |
|                                                                       |
| /usr/local/apache/bin/apachectl start                                 |
| Or when you want to run it with SSL enabled use:                      |
|                                                                       |
| /usr/local/apache/bin/apachectl staztssl                              |
| Thanks for using Apache. The Apache Group                             |
| http://www.apache.org/                                                |
+-----------------------------------------------------------------------+
```

Теперь необходимо проверить работоспособность Apache и PHP. Однако до этого следует внести изменения в файлы **httpd.conf и srm.conf,** добавив тип PHP к конфигурации системы.

После всех предыдущих шагов файл **httpd.conf** находится в каталоге **/usr/local/apache/conf**. В этом файле опция **addtype** для PHP 4 закомментирована. Комментарий из нее следует убрать.

Фрагменты файла httpd.conf

```
>
> # Для PHP 4.x нужно использовать:
> #
——> AddType application/x-httpd-php .php
——> AddType application/x-httpd-php-source .phps
>
>
```

Все готово к запуску и проверке сервера Apache. Вначале сервер запускается без поддержки SSL, чтобы проверить его работоспособность. После этого проверяется поддержка PHP, затем сервер останавливается и перезапускается уже с SSL.

Проверку правильности настройки полной конфигурации осуществляет утилита **configtest**:

```
# cd /usr/local/apache/bin
# ./apachectl configtest
Syntax OK
# ./apachectl start
    ./apachectl start: httpd started
```

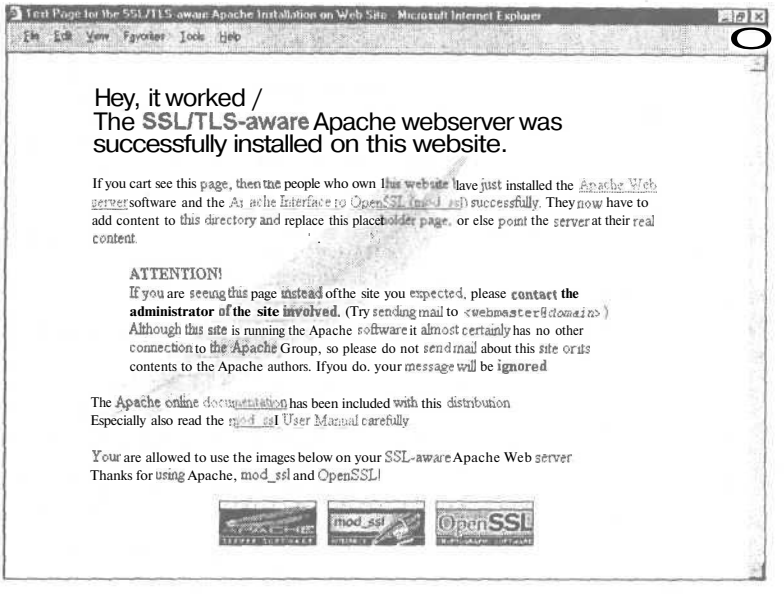


РИСУНОК А.1 *Тестовая страница по умолчанию, поставляемая вместе с Apache.*

Если все работает правильно, то при соединении с сервером в Web-браузере появляется содержимое, подобное показанному на рис. АЛ.

ПРИМЕЧАНИЕ

Указать сервер можно с помощью доменного имени или IP-адреса. Для проверки следует испробовать оба варианта.

Работает ли поддержка PHP?

Теперь необходимо проверить поддержку PHP. Для этого следует создать файл **test.php,** содержащий приведенный ниже код. Файл должен быть расположен в корневом каталоге документов — по умолчанию **/usr/local/apache/htdocs**. Обратите внимание, что в общем случае его нахождение определяется опцией prefix, использованной при конфигурировании. В процессе работы настройку можно изменить в файле **httpd.conf**.

```
<? phpinfo() ?>
```

Вывод на экран показан на рис. А.2.

Работает ли SSL?

После этого так же следует тестировать и SSL. Для этого необходимо остановить сервер и перезапустить его с включенной поддержкой SSL:

```
f /usr/local/apache/bin/apachectl stop
# /usr/local/apache/bin/apachectl startssl
```

Затем соединиться с сервером и выбрать в Web-браузере протокол **https:**

```
https://yourserver.yourdomain.com
```

ИЛИ

```
http://yoursever.yourdomain.com:443
```

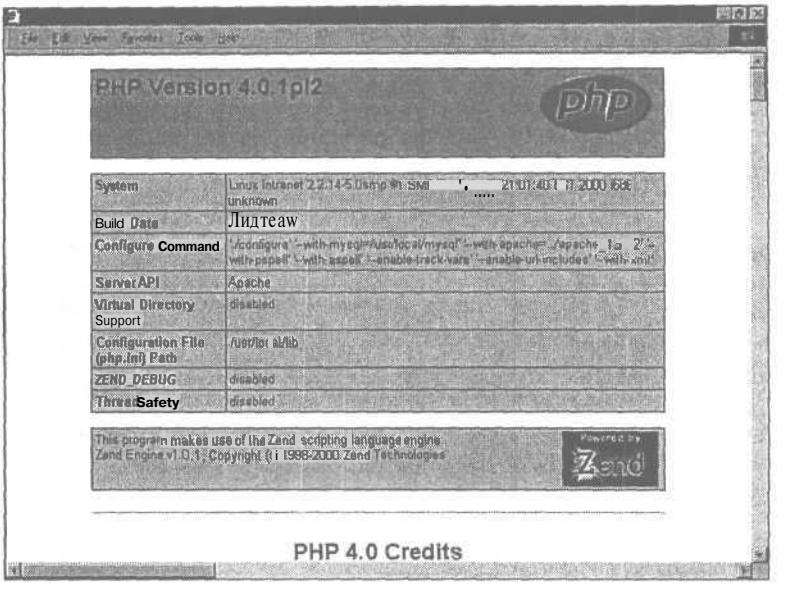


РИСУНОК А.2 *Функция phpinfo() возвращает полезную информацию о текущей конфигурации.*

Следует проверить и IP-адрес, например:

```
https://xxx.xxx.xxx.xxx
```

ИЛИ

```
http://xxx.xxx.xxx.xxx:443
```

Если все работает как положено, сервер перешлет сертификат браузеру для попытки установки безопасного соединения. Браузер запросит подтверждение на принятие самозакодированного сертификата. Если сертификат поступил от VeriSign или Thawte, браузер не будет запрашивать подтверждения, поскольку подобного рода сертификаты являются зарегистрированными органами сертификации (Certification Authority, CA). Здесь в примере используются собственные сертификаты, так как сначала нужно просто убедиться, что все работает как следует.



РИСУНОК А.3 *Web-браузеры выводят пиктограмму, указывающую, что просматриваемая страница получена через SSL-соединение.*

Если браузером является Internet Explorer или Netscape, то в строке состояния появится символ замка. Он показывает, что безопасное соединение успешно установлено. Пиктограмма, используемая в Netscape, показана на рис. А.3.

Установка Apache, PHP и MySQL под Windows

В среде Windows процесс установки является несколько иным, поскольку PHP можно настроить как сценарий CGI (php.exe) или как модуль ISAPI (php4isapi.dll). Однако Apache и MySQL устанавливаются таким же образом, как и в UNIX. До инсталляции пакетов под Windows следует убедиться, что уже установлены последние версии обновлений служб операционной системы.

Как и ранее, начинать нужно с загрузки всех файлов исходного кода во временный каталог на диске, имеющем достаточно свободного дискового пространства. Здесь временным каталогом для установки является **C:\TEMP\DOWNLOAD**.

Установка MySQL под Windows

Вначале будет произведена установка MySQL. После загрузки всех исходных файлов архив MySQL необходимо разархивировать во временный каталог и запустить программу *Setup.exe*. Обратите внимание, что по умолчанию MySQL устанавливается в каталог C:\mysql. При необходимости после инсталляции все его содержимое можно переместить в другой каталог.

Если перемещение MySQL произошло, программе **mysqld** с помощью опций необходимо указать, где находятся все элементы MySQL. Для вывода возможных опций достаточно воспользоваться командой **C:\mysql\bin\mysqld --help**. Например, если инсталляция MySQL находится в каталоге **'D:\programs\mysql'**, запускать **mysqld** следует командой **'D:\programs\mysql\bin\mysqld --basedir D:\programs\mysql'**.

В последних версиях MySQL можно создать файл **'C:\my.cnf'**, содержащий все настройки по умолчанию сервера MySQL. Файл **'C:\mysql\my-xxxxx.cnf'** необходимо скопировать под именем **'C:\my.cnf'** и изменить настройки согласно вашим требованиям.

Windows 95/98

Версия MySQL под Windows 95/98 содержит два различных сервера MySQL:

- **mysqld:** Скомпилирован с полной отладочной информацией и проверкой автоматического выделения памяти
- **mysqld-opt**: Оптимизирован под процессоры Pentium

Обе версии работают на текущих Intel x86 или более поздних процессорах.

Сервер **mysqld** можно запустить из командной строки Windows следующим образом:

```
C:\mysql\bin\mysqld-opt
```

Эта команда запускает сервер MySQL в фоновом режиме. Если сервер не запускается, следует, прежде всего, проверить наличие ошибок в файле **'\mysql\mysql.err'**, которые могут указывать, что именно не работает. Для останова сервера MySQL применяется команда

```
C:\mysql\bin\mysqladmin -u root shutdown
```

Windows NT/Win2000

Между запуском сервера MySQL на платформе NT и Windows 2000 существуют небольшие отличия. В инсталляции MySQL под NT/Win2000 сервер имеет имя **mysqld-nt** и обычно устанавливается как служба. Это достигается следующей командой:

```
C:\mysql\bin\mysqld-nt -install
```

После этого сервер MySQL как обычная служба может быть запущен или остановлен набором команд

```
NET START mysql
NET STOP mysql
```

ПРИМЕЧАНИЕ

Здесь используется именно имя **mysql, а не mysql-nt.**

После инсталляции сервер должен быть запущен утилитой диспетчера служб (Services Control Manager, SCM) (находящейся в панели управления (Control Panel)) или командой **NET START MySQL.** Применение SCM показано на рис. А.4. Все требуемые

опции необходимо указать как *параметры запуска (startup parameters)* утилиты SCM до запуска службы MySQL. Запущенную службу **mysqld-nt** можно остановить, используя **mysqladmin**, утилиту SCM или команду NET STOP **MySQL**.

Проверка работы MySQL осуществляется следующим набором команд:

```
C:\mysql\bin\mysqlshow
C:\mysql\bin\mysqlshow -u root mysql
C:\mysql\bin\mysqladmin version status proc
C:\mysql\bin\mysqladmin -и root shutdown
```

Эти команды выполняются одинаково под разными версиями операционной системы Windows.

MySQL создаст две базы данных: mysql и test. База данных **mysql** необходима для сохранения прав доступа к серверу. База данных test не является обязательной, однако позволяет проверить правильность работы сервера MySQL.

За более подробной информацией следует обратиться на Web-сайт MySQL по адресу http://www.mysql.com.

Теперь все готово к инсталляции Apache под Windows.

Установка *Apache под Windows*

Apache версии 1.3 и выше разработан так, чтобы запускаться под Windows NT 4.0 и Windows 2000. Программа установки работает только на семействе процессоров x86 (например, Intel). Apache может быть запущен под Windows 95/98, однако эта возможность не была проверена авторами. В любом случае, в системе должен быть установлен протокол TCP/IP. Под Win95 или Win98 необходимо убедиться в наличии библиотеки Winsock 2.

Начинающим (и тем, кто хочет отказаться от компиляции исходного кода) рекомендуется загрузить версию Apache для Windows в форме **.exe-файла**. Этот файл содержит готовый к установке дистрибутив сервера Apache.

Загруженный файл **apache_1_3_x_win32.exe** можно запустить двойным щелчком на его имени.

Процесс инсталляции похож на установку многих приложений Windows (как показано на рис. А.5) и поэтому должен выглядеть знакомым.

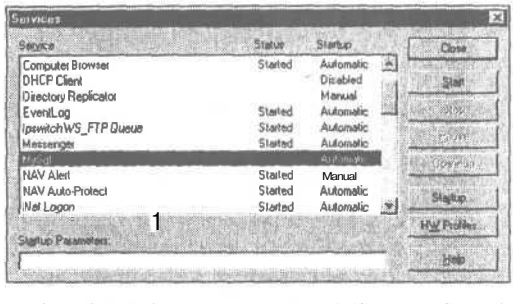


РИСУНОК А.4 *Диспетчер служб (Services Control Manager) позволяет конфигурировать службы, запущенные на данной машине.*

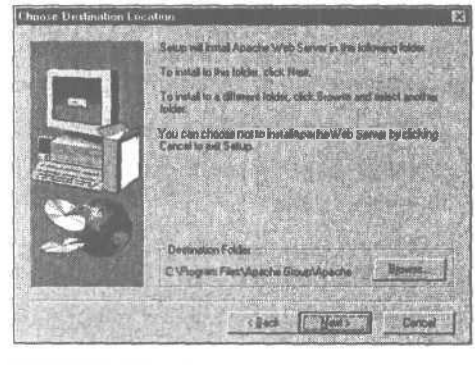


РИСУНОК А.5 *Простая в использовании программа инсталляции сервера Apache.*

Программа инсталляции запрашивает следующее:

- Каталог для установки Apache. (По умолчанию это **C:\Program Files\Apache Group\Apache**.)
- Имя меню в меню Start. (Apache Web Server по умолчанию.)
- Тип инсталляции. Опция Typical (Обычная) устанавливает все, кроме исходного кода. Опция Minimum (Минимальная) не устанавливает справочных руководств и исходного кода. Если требуется исходный код, следует выбрать опцию Custom (Выборочная).

После установки Apache может потребоваться внести изменения в конфигурационные файлы из каталога **conf**. О редактировании файла **httpd.conf** рассказано в разделе, посвященном установке PHP.

Запуск Apache под Windows

В общем случае существует два способа запуска Apache:

- Из окна консоли
- В виде службы Windows

Запуск сервера как службы используется обычно в Window NT и Windows 2000. Это позволяет запускать Apache автоматически при загрузке машины и останавливать при выходе из системы. Консольный вариант предназначен, в основном, для пользователей Windows 95 и Windows 98. Однако, в версии сервера 1.3.13 существует возможность пользователям Win 95 и Win 98 запускать Apache как службу. Следует отметить, что эта версия рассматривается разработчиками как сугубо экспериментальная.

В этом примере Apache будет установлен как служба лишь после того, как его работа будет успешно протестирована из окна консоли. Поэтому вначале рассматривается запуск Apache из окна консоли.

Запуск Apache из окна консоли

Чтобы запустить Apache из окна консоли, следует выбрать опцию Start Apache as console App (Запустить Apache как консольное приложение) из меню Start (Пуск). В результате будет открыто новое окно консоли, в котором запускается сервер. Оно будет оставаться активным до тех пор, пока сервер Apache не будет остановлен.

Для останова необходимо или выбрать опцию Shutdown Apache as Console App (Остановить Apache как консольное приложение) из меню Start (Пуск), или открыть другое окно и ввести в нем следующую команду (для версии 1.3.3 или ниже):

```
C:\Program Files\Apache Group\Apache> apache -k shutdown
```

В отличие от MySQL, Apache запускается не в фоновом режиме, поэтому его можно остановить, просто нажав комбинацию клавиш Control-C или Control-Break в окне консоли, либо закрыв это окно. (Эта возможность имеется только в версии 1.3.3 и предыдущих.)

Запуск Apache как службы

Чтобы запустить Apache как службу, его необходимо установить как службу. Помните, что на одной машине можно установить несколько служб Apache с различными именами и конфигурациями.

Для установки службы Apache по умолчанию (под именем Apache), следует выбрать опцию Install Apache as Service (NT only) (Установить Apache как службу (только NT))

из меню Start (Пуск). Затем открыть окно Services (Службы) (в Control Panel (Панели управления)), выбрать Apache и нажать Start (Пуск). Сервер Apache будет запущен в фоновом режиме. Остановить его можно, нажав Stop (Останов). Вместо применения окна Services (Службы) службу Apache можно запустить или остановить из командной строки, используя следующие команды

```
NET START apache
NET STOP apache
```

Обратите внимание, что они аналогичны командам сервера MySQL под NT и Windows 2000.

Apache в отличие от других приложений NT и Win2000 записывает все сообщения об ошибках в свой собственный журнальный файл **error.log**, находящийся в корневом каталоге этого сервера. Apache не использует стандартную систему журнализации Event Log (Журнал событий).

Как было отмечено ранее, можно установить и запустить в виде служб несколько копий Apache. Чтобы передать службе Apache сигнал к запуску, перезапуску или останову, необходимо указать имя службы в одной следующих команд:

```
apache -n "service name" -k start
apache -n "service name" -k restart
apache -n "service name" -k shutdown
```

Для службы по умолчанию все равно требуется опция **-n**, поскольку без нее команда -k будет обращена к серверу Apache, запущенному из окна консоли. Кавычки требуются лишь тогда, когда имя службы содержит пробелы.

После запуска Apache прослушивает порт 80 (если только в конфигурационных файлах не были изменены директивы Port (Порт), Listen (Прослушивать) или BindAddress (Привязка адреса)). Для соединения с сервером и получения страницы по умолчанию, потребуется запустить браузер и ввести в строке URL:

```
http://localhost/
```

В браузере должна появиться страница, показанная на рис. A.1, и ссылка на руководство по Apache. Если ничего не произошло или возникла ошибка, необходимо просмотреть файл **error.log** в каталоге **logs.** Если хост не соединен с Internet, можно использовать следующий URL:

```
http://127.0.0.1/
```

Это IP-адрес машины с именем localhost.

Если номер порта был изменен (и уже не равен 80), к URL нужно добавить параметр **:port_number**.

Обратите внимание, что Apache НЕ МОЖЕТ использовать один и тот же порт одновременно с другим приложением TCP/IP.

Различия между версиями **Apache** для **Windows** *и UNIX*

Ниже приведены основные различия между версиями Apache под Windows и UNIX:

- Сервер Apache для Windows является многозадачным, однако он не использует отдельные процессы для обработки каждого запроса, как это происходит для случая UNIX. Вместо этого обычно существуют два процесса Apache: родительский и дочерний, причем последний обрабатывает все запросы. Внутри дочернего процесса запросы обрабатываются отдельными процессами. Поэтому директивы управления процессами являются разными под двумя этими платформами.

- Директивы, аргументами которых являются имена файлов, теперь используют соглашения об именах файлов Windows, а не UNIX. Однако, поскольку внутренне Apache использует формат UNIX, необходимо применять символ прямой, а не обратной косой черты. Можно использовать имена разделов; если они опущены, то подразумевается раздел, на котором находится выполняемый файл Apache.
- Apache для Windows обладает возможностью загружать модули во время выполнения без перекомпиляции сервера. Если Apache скомпилирован обычным образом, набор дополнительных модулей находится в каталоге \modules. Для активации этих или каких-либо других модулей следует воспользоваться новой директивой LoadModule (Загрузить модуль). Например, для активации модуля состояния (status) применяется следующее (в дополнение к директивам в файле **access.conf**):

  ```
  LoadModule status_module modules/mod_status.so
  ```

  С подробностями можно ознакомиться в электронном руководстве по адресу

  http://httpd.apache.org/docs/mod/mod_so.html#loadmodule

- Все версии из набора 1.3 сервера Apache для Windows реализуют синхронные вызовы. Это представляет собой огромную проблему для авторов CGI-сценариев, которым не требуется, чтобы небуферизованный вывод пересылался немедленно в браузер. Это не является стандартным поведением CGI в Apache, а представляет собой сторонний эффект переноса сервера под Windows. Apache 2.0 продвигается вперед на пути к реализации требуемого несинхронного поведения. Это позволяет ожидать, что реализация NT/2000 позволит CGI работать согласно стандартным требованиям.

Если под Windows требуется сервер Apache с поддержкой SSL, то единственная возможность состоит в компиляции исходного кода. О том, как это сделать, можно узнать на сайтах Apache.org, OpenSSL.org и ModSSl.org, а также в предыдущем разделе, посвященном установке сервера под UNIX. Это не представляет сложностей, однако требует дополнительной работы.

Установка PHP под Windows

Система готова к установке PHP под Windows. До начала процесса установки необходимо остановить сервер Apache. Процесс чрезвычайно прост: в отличие от PHP 3, PHP 4 разделен на несколько компонент, которые требуют установки нескольких DLL-файлов. То есть PHP нельзя запустить в режиме CGI как отдельный выполняемый файл. До этого нужно убедиться, что требуемые DLL-файлы существуют в каталоге (любом), добавленном в путь поиска Windows. Простейший способ заключается в копировании этих файлов в каталог SYSTEM (Windows 9x) или SYSTEM32 (Windows NT), находящийся внутри каталога Windows. Необходимыми файлами являются MSVCRT.DLL (он может быть уже установлен ранее) и PHP4TS.DLL.

Далее приведена схема, которая может служить кратким пособием по инсталляции. Пользователи, следующие ему, не должны иметь проблем при установке и настройке PHP на Windows-машине.

1. В начале потребуется скопировать php.ini-dist в каталог **'%WINDOWS%'** и переименовать его в **'php.ini'**. Переменная **'%WINDOWS%'** обычно указывает на **C:\WINDOWS** под Windows 9x и на **C:\WINNT** под NT.
2. В файле **php.ini** опцию extension_dir необходимо установить равной имени каталога, содержащего DLL-модули расширений, а опцию doc_root — равной корневому каталогу документов Web-сервера (т.е. корневому каталогу сервера, который виден снаружи).
3. В файле **php.ini** убрать комментарии с имен модулей, которые нужно загружать при запуске PHP. За это отвечают строки вида **extension=php_*.dll**. Обратите внимание, что некоторые модули для корректной работы требуют наличия в системе дополнительных библиотек. Кроме того, поддержка MySQL теперь встроена в PHP 4 — она больше не требует подключения описанным выше методом.

Последнее, что осталось — внести изменения в файл **httpd.conf** из каталога **conf** дерева Apache. Это необходимо для настройки работы PHP через CGI. К конфигурационному файлу следует добавить такие директивы:

```
ScriptAlias /php/ "c:/путь-к-каталогу-php/"
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
Action application/x-httpd-php "/php/php.exe"
```

Обратите внимание, что директива AddType позволяет указать Apache, как следует обрабатывать файлы с различными расширениями. В приведенном выше случае Apache трактует все файлы с расширениями .php и .phtml как интерпретируемый PHP-код. Можно, например, и обычные файлы .htm и .html рассматривать как PHP-сценарии. За это отвечают директивы:

```
AddType application/x-httpd-php .html
AddType application/x-httpd-php .htm
```

Более подробную информацию о других директивах, которые можно установить в конфигурационном файле, можно найти на Web-сайте Apache **http://www.apache.org.**

Проверка результатов работы

После запуска Apache следует убедиться, что PHP работает правильно. Для этого потребуется создать сценарий **test.php**, содержащий строку:

```
<? phpinfo() ?>
```

Файл должен находиться в корневом каталоге документов Apache, тогда его можно запросить из браузера по адресу

```
http://localhost/test.php
```

ИЛИ

```
http://ваш-ip-адрес/test.php
```

О том, что Apache и PHP работают вместе, можно заключить, если на экране будет отображаться страница, приведенная на рис. А.2. Следует проверить и работу MySQL вместе с ними. Для этого необходимо написать простой PHP-сценарий, который соединяется с базой данных MySQL и помещает/извлекает данные из нее.

Присоединение PHP и MySQL к серверам Microsoft IIS и PWS

В этом разделе рассказано о том, как добавить поддержку PHP и MySQL в IIS в качестве модуля ISAPI (php4isapi.dll). Здесь подразумевается, что MySQL установлен так, как об этом было рассказано в одном из предыдущих разделов.

Первое, что следует сделать — установить DLL-файлы, упомянутые в прошлом разделе (MSVCRT.DLL и PHP4TS.DLL) в каталог **Windows.** Кроме того, нужно поместить в файл **test.php** команды, описанные выше.

Заметки по инсталляции для Microsoft IIS

Вот краткое пособие по инсталляции PHP в Microsoft Internet Information Server:

1. Скопировать либо файл **php.ini-dist,** либо файл **php.ini-optimized** в каталог Windows и переименовать его в **'php.ini'** Изменить настройки по умолчанию (если требуется) путем указания требуемых директив.
2. Запустить Microsoft Management Console (Консоль управления Microsoft) (она называется Internet Services Manager или в Windows NT 4.0 Option Pack или в Control Panel, Administrative Tools (Панель управления, Средства администрирования) в Windows 2000).
3. Щелкнуть правой кнопкой мыши на пиктограмме Web-сервера и выбрать Properties (Свойства). В пункте ISAPI Filters (Фильтры ISAPI) добавить новый фильтр ISAPI. Соответствующее диалоговое окно показано на рис. А.6. Затем добавить PHP как имя фильтра и указать путь к файлу php4isapi.dll, который включен в дистрибутив PHP 4.
4. Во вкладке Home Directory (Начальный каталог) нажать кнопку Configuration (Конфигурация) и добавить новую запись в Application Mappings (Соответствия приложений). Ввести путь к php4isapi.dll как к выполняемому файлу, указать расширение .php, оставить поле Method exclusions (Методы исключения) пустым и включить флажок Script engine (Механизм сценариев).
5. Полностью остановить сервер IIS, выполнив команду **'net stop iisadmin'** из командной строки.



РИСУНОК А.6 *Добавление интерпретатора PHP в качестве фильтра ISAPI.*

6. Перезапустить IIS командой **'net start w3svc'**.

7. Поместить файл **test.php** в корневой каталог документов Web-сервера. Файл **test.php,** как и ранее, содержит строку:

```
<?php phpinfo(); ?>
```

Если все работает успешно, на экране появится вывод, показанный на рис. А.2.

Замечания по инсталляции для Microsoft PWS

1. Установите файл **php.ini** и DLL-файлы так же, как в предыдущем случае.

2. Внесите изменения в файл **PWS-php4.reg,** указав местонахождение файла **php4isapi.dll**. Символы прямой косой черты необходимо предварить служебными косыми, например:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\
Script Map] ".php"="C:\\Program Files\ \PHP\\php4isapi.dll"
```

3. В PWS Manager (Диспетчер PWS) щелкните правой кнопкой мыши на каталоге, к которому нужно добавить поддержку PHP, и выберите Properties (Свойства). Установите флажок Execute (Запустить) и подтвердите изменения.

Вот и все! После этого PWS будет содержать встроенную поддержку PHP.

Другие конфигурации

PHP и MySQL можно настроить и под управлением других Web-серверов, включая Omni, HTTPD и Netscape Enterprise Server. Об этом в данном приложении не упоминается, поэтому за необходимой информацией следует обратиться на Web-сайты MySQL и PHP:

```
http://www.MySQL.com
```

и

```
http://www.php.net
```

Приложение B

Ресурсы Internet

В данном приложении перечислены некоторые из многих ресурсов в Internet, содержащие обучающие системы, статьи, новости и примеры PHP-кода. Это лишь их небольшая часть. Более того, каждый день возникают все новые и новые сайты, посвященные PHP и MySQL, — по мере того, как все большее число Web-разработчиков использует эти продукты.

Некоторые из приведенных сайтов содержат другие языки, например, немецкий или французский. Для перевода на другие языки можно воспользоваться переводчиком реального времени наподобие **http://www.systransoft.com**.

Ресурсы, посвященные PHP

PHP.Net — **http://www.php.net** — Основной сайт PHP. На нем можно найти полный исходный код PHP, а также справочное руководство по нему.

ZEND.Com — **http://www.zend.com** — Источник механизма ZEND, под управлением которого работает PHP 4.0. Этот сайт-портал содержит форумы, а также базу данных примеров классов и кода, которые можно свободно использовать. Рекомендуем его посмотреть.

PHPWizard.net — **http://www.phpwizard.net** — Источник многих хороших приложений PHP, например, phpMyAdmin, замечательный графический интерфейс для управления серверами MySQL. Кроме того, сайт содержит обучающие руководства по PHP.

PHPBuilder.com — **http://www.phpbuilder.com** — Портал обучающих курсов по PHP. На этом сайте можно найти ответы практически на любые вопросы. Кроме того, поддерживается форум и доска объявлений, где можно размещать свои вопросы.

DevShed.com — **http://www.devshed.com** — Сайт-портал, содержащий замечательные обучающие руководства по PHP, MySQL, Perl и другим языкам программирования. Очень полезен для начинающих.

PX-PHP Code Exchange — http://px.sklar.com — С него стоит начать. Здесь много примеров сценариев и полезных функций. Сайт содержит удобную поисковую систему.

The PHP4 Resource — **http://www.php-resource.de** — Удобный источник обучающих руководств, статей и сценариев. Единственная "проблема" заключается в языке — сайт на немецком. Для его просмотра авторы рекомендуют использовать систему перевода сайтов.

WeberDev.com — **http://www.WeberDev.com** — Известный ранее как "Berber's PHP sample page", этот сайт вырос буквально из ничего в место для обучающих руководств и примеров кода. Он предназначен для пользователей PHP и MySQL и покрывает вопросы безопасности, баз данных и NT. Однако этот сайт требует подписки. Тем не менее, он того стоит, учитывая информацию, которую на нем можно найти.

HotScripts.com — **http://www.hotscripts.com** — Набор сценариев, разбитый по категориям. Сайт содержит сценарии на таких языках, как PHP, ASP и Perl. На нем содержится замечательная коллекция PHP-сценариев. Сайт часто обновляется, поэтому его рекомендуется посещать тем, кто ищет примеры сценариев.

PHP Base Library — **http://phplib.netuse.de** — Сайт, использующийся разработчиками крупных проектов на PHP 3. Он содержит библиотеку большого числа средств для управления сеансами, которые теперь встроены в PHP 4, а также создания шаблонов и абстрактного слоя баз данных. Сайт полезен также тем, кто ищет обучающие руководства по PHP.

PHP Center — http://www.php-center.de — Еще один сайт-портал на немецком языке, содержащий обучающие руководства, сценарии, советы и многое другое.

PHPInfo.net — **http://www.phpinfo.net** — Этот французский сайт содержит немало информации о PHP и MySQL, а также ссылки, советы, статьи, ответы на часто задаваемые вопросы и многое другое.

PHP Homepage — **http://www.php-homepage.de** — Еще один немецкий сайт, посвященный PHP: сценарии, статьи, новости и многое другое. Есть раздел быстрых ссылок.

PHPIndex.com — **http://www.phpindex.com** — Удобный французский PHP-портал с огромным количеством материала, посвященного PHP. Он содержит новости, ответы на часто задаваемые вопросы, статьи, вакансии рабочих мест и многое другое.

WebMonkey.com — **http://www.webmonkey.com** — Портал с большим количеством Web-ресурсов, обучающие руководства, примеры кода и т.д. Сайт покрывает вопросы разработки, программирования, интерфейса к базам данных, мультимедиа и множество других.

The PHP Club — **http://www.phpclub.net** — PHP Club содержит много ресурсов для начинающих программистов на PHP: новости, обзоры книг, примеры кода, форумы, ответы на часто задаваемые вопросы, а также руководства для начинающих.

The PHP Classes Repository — **http://phpclasses.upperdesign.com** — Основной целью этого сайта является распространение бесплатных классов, написанных на PHP. Если вы разрабатываете код или ваш проект требует создания классов, обязательно посетите этот сайт. Он содержит удобную поисковую систему.

The PHP Resource Index — **http://php.resourceindex.com** — Сайт-портал сценариев, классов и документации. Он очень удобно разбит по категориям, что сокращает время поиска.

PHP Developer — **http://www.phpdeveloper.org** — Еще один PHP-портал: новости, статьи, обучающие руководства.

Evil Walrus — **http://www.evilwalrus.com** — Портал сценариев на PHP.

Oodie.com — **http://www.oodie.com** — Содержит список бесплатных провайдеров с PHP-хостингом, а также примеры сценариев на PHP.

e-gineer — **http://www.e-gineer.com** — Статьи, сценарии, а также основные вопросы по PHP с ответами на них.

Source Forge — **http://sourceforge.net** — Источник ресурсов с открытым исходным кодом. Source Forge не только позволяет отыскать требуемый код, но и предоставляет доступ к CVS, спискам рассылки и машинам разработчиков программного обеспечения с открытым исходным кодом.

Ресурсы, посвященные MySQL и SQL

The MySQL site — **http://www.mysql.com** — Официальный Web-сайт MySQL. Содержит отличную документацию, поддержку и немало информации. Рекомендуется всем, кто использует MySQL. Содержит исходный код сервера баз данных MySQL.

SQL Tutorial — **http://w3.one.net/~jhoffinan/sqltut.htm** — Исчерпывающее руководство по SQL для начинающих с примерами и упражнениями.

The SQL Course — **http://sqlcourse.com** — Вводный обучающий курс SQL с понятными инструкциями. Позволяет проверить изученный материал на встроенном интерпретаторе SQL. Расширенная версия находится по адресу **http://www.sqlcourse2.com**.

DatabaseCentral.com — **http://databasecentral.com** — Подробный портал с обилием информации о базах данных. Отличные обучающие руководства, советы, официальные издания, ответы на часто задаваемые вопросы, обзоры и т.д. Рекомендуем просмотреть!

The SQL **Pro — http://www.inquiry.com/techtips/thesqlpro** — Есть вопросы по программированию? Задайте их профессионалам! Поиск по базе данных вопросов и ответов, посвященных разработке баз данных и SQL.

Ресурсы, посвященные Apache

Apache Software — **http://www.apache.org** — Сайт, с которого следует начать, если вам нужны исходный код или двоичные файлы. Сайт содержит электронную документацию.

Apache Week — **http://www.apacheweek.com** — Электронный еженедельный журнал, полезный для тех, кто работает с сервером Apache или использует его службы.

Apache Today — **http://www.apachetoday.com** — Ежедневно обновляемый источник информации по Apache. Для отправки вопросов пользователям необходимо предварительно зарегистрироваться.

Разработка для Web

Путеводитель по Web-публикациям от Филиппа и Алекса (Philip and Alex's Guide to Web Publishing) — **http://www.arsdigita.com/books/panda/** — Остроумный путеводитель по применению методов разработки программного обеспечения для Internet. Одна из нескольких книг по данной теме в соавторстве с Samoyed.

Предметный указатель

Символы

-- 36
% 192
& 42
&& 44
. 35, 39
= 39
== 43
?: 45
@ 45, 326
`` 46
| 111
|| 44

А

Агрегирование данных 198
Алгоритм шифрования 256
Аномалия 163
  ввода 163
  модификации 163
  обновлений 163
  удаления 163
Анонимный вход 325
Аргументы 33
Арифметические операции 39
Архив 388
Архитектура Web-баз данных 166
Ассоциативность 47
Ассоциативные массивы 76
Атрибут 140, 141
Аудит 262
Аутентификация 247, 254, 265, 266, 411
  HTTP 274
  базовая 272
  дайджест 272
  пользователей 418

Б

База данных 211
  безопасность 221
  внесение новой информации в базу данных 211
  отсоединение от базы данных 211
  реализация 415
  создание и удаление баз данных 214
Базовая линия 352
Безопасность 246, 264
Библиотека GD 341
Блоки кода 51, 136
Блокирование файлов 72
Брандмауэр 221, 262

В

Ветвление 111
Возврат значений из функций 135
Восстановление баз данных MySQL 263
Встраивание PHP в HTML 29
Выражение 108
  регулярное 108

Г

Группа тематических дискуссий 571. *Си. также* Форум
Группировка данных 198

Д

Дата 338
  операции над датами 338
Двумерный файл 60
Декартово произведение таблиц 193
Декремент 42
Дескриптор 30, 140
  PHP 30
Дешифрация 256
Дискуссионная трибуна 571. *См, также* Форум
Документы RFC 317
Документирование проектов 390
Доступ 478
  FTP 478

З

Закладка 413
  добавление 432
  извлечение 432
  удаление 434
  хранение 432
Закрытие файла 66
Запись в файл 65, 400
Защита 254
Защита множества страниц 271
Знаки операций 39

И

Идентификатор 36
  MySQL 181
  сеанса 362
Идентификация пользователей 265
Индекс 76. *См. также* Ключ
Инкапсуляция 140
Инкремент 42
Интерфейс 140
  администрирования 469
Итерация 54

К

Каскадные таблицы стилей (CSS) 392
Класс 108, 140, 141
  атрибуты класса 143
  написание кода класса 149
  подкласс 141
  разработка классов 148
  структура класса 142
  суперкласс 141
  экземпляр класса 143
Класс treenode 582
Ключ 76, 160. *См. также* Индекс
  внешний 161
  первичный 160
Код 387
  выравнивание 386
  комментирование 385
  оптимизация 392
  стандарты написания 384
  фрагментирование 387
Команда
  CREATE TABLE 177
  DESCRIBE 180
  GRANT 172, 216
  mysql 170
Команда
  mysql_install_db 170
  REVOKE 175
  SHOW 180
Комментарии 32
Комментирование кода 385
Коммерческие Web-сайты 234
Компания Zend Technologies 393
Конец файла 68
Конкатенация строк 35
Константы 38
Конструктор 142

Л

Литерал 35
Логические операции 44

М

Массив 75
  ассоциативный 76
  доступ к элементам массива 78
  многомерный 80
  сортировка массивов 83
  численно индексированные массивы 76
  элемент массива 76
Менеджер списков рассылки 529
Метаданные 480
Множественное наследование 148
Модуль mod_auth_mysql 279

Н

Наборы символов 108
Наследование 141, 146, 148
  множественное 148

О

Область действия 39, 131
Объединение двух таблиц 193
  полное 193. *См. также* Декартово произведение таблиц
Объединение по равенству (equi-join) 193
Объединение трех и более таблиц 194
Объект 140
Объектно-ориентированное программирование 139
Операнд 39
Оператор
  break 58
  continue 58

Оператор
CREATE TABLE 178
DESCRIBE 225
describe user 217
do..while 57
each 79
else 51
elseif 52
EXPLAIN 225
for 56
if 50, 51
include() 123
INSERT 189
LOAD DATA INFILE 231
require() 118
return 135
SELECT 199
SHOW 224
SHOW GRANTS 224
switch 52
while 55
Операторы 31, 39
PHP 31
арифметические 39
, условные 50
Операции 40
арифметические 39
запуска на выполнение 46
логические 44
над датами 338
подавления ошибки 45
поразрядные 44
присваивания 40
сравнения 43
ссылки & (амперсанд) 42
строковые 40
тернарные 45
Определение конца файла 68
Оптимизация
кода 392
проектирования 229
таблиц 229
Освобождение ресурсов 214
Открытие удаленных файлов через FTP или HTTP 63
Открытие файла для чтения 68
Открытый интерфейс доступа к базам данных (ODBC) 215
Отладка 396
удаленная 410
переменных 404
Отношение 161
"многие ко многим" 161
"один к одному" 161
"один ко многим" 161
Отправка почты 317, 525
Отсоединение от базы данных 211
Отступы 51
Ошибки 396
времени выполнения 396, 398
генерация собственных ошибок 408
логические 396, 402
обработка ошибок 408
программные 396
синтаксические 396

П

Пакет GPG 293
Параметр 129
Пароль 222, 268, 427
переустановка забытых паролей 428
смена паролей 427
хранение 268
шифрование 270
Перегрузка функции 129
Передача по значению 133
Передача по ссылке 133
Перекрытие 146
Переменные 131
PHP 34
глобальные 131
локальные 131
переменных 38
типы переменных 36
формы 34
Персонализация пользователей 411
Подкласс 141
Подпись 258
цифровая 258
Подстрока 105
Поиск подстрок 112
Покупательская тележка 441
реализация 455
Полиморфизм 141
Получение почты 317

Поля привилегий 218
Поразрядные операции 44
Построчное считывание 68
Почтовые сообщения 516
Права доступа 229
Правила именования 384
Приведение типов 37
Привилегии 171, 216
  для администраторов 174
  для пользователей 174
  пользователей 222
    FILE 222
    PROCESS 222
    RELOAD 222
    SHUTDOWN 222
  специальные 175
Приложение
  Certification 605
  PHPBookmark 414
  Shopping Cart 445
  Warm Mail 503
Приоритет 47
Пробел 31
Программа Tera Term SSH 325
Программные ошибки 396
Протокол 288, 316, 361, 501
  FTP 316, 323
  HTTP 288, 316, 361
  IMAP 317, 501
  NNTP 501
  POP 317
  POP3 501
  Secure Socket Layer 260
  SMTP 317, 501
  TCP/IP 288
Прототип 391
Псевдоним 196

P

Реализация покупательской тележки 455
Реализация сайта 416
Регистрация 262
Регулярные выражения 108
Режим файла 61
Резервное копирование данных 263
Рекурсия 115, 137

C

Сеанс 361
  идентификатор сеанса 362
  управление 361
Сервер
  Apache 260, 274
  Microsoft IIS 260
Сериализация 376
Сертификат 259
Сетевые брошюры 234
Сетевые службы 401
Система Book-O-Rama 444
Система CVS 388
Система
  MySQL 171
  NASDAQ 318
  управления версиями 388
  управления реляционными базами данных 73
Слой безопасных сокетов (SSL) 287
Служба
  почтовая 500
Смена паролей 427
Создание
  Web-приложений 381
  изображений 343
  прототипов 391
  учетных записей 511
  форума 571
  холста 344
Специальные символы 111
Списки рассылки 529
Сравнение строк 104
Среда разработки (IDE) 389
Ссылки 42
Строка 33, 35, 97
  изменение регистра строки 100
  поиск подстрок 112
  сравнение строк 104
  усечение строк 97
  форматирование строк 97, 98
Строковые операции 40
Строковые типы 184
Суперкласс 141
СУРБД 74, 158
Схема 161

Сценарий 284, 378
checkout.php 463
delete_story.php 495
index.php 451
page.php 487
pdflib.php 621
PHP 284
publish.php 498
resize_image.php 482
search.php 496
show_book.php 454
show_cart.php 456, 457, 462
story.php 492
story_submit.php 494
view_post.php 589
определение владельца сценария 378
определение даты последнего изменения сценария 378
Считывание
всего файла 69
из каталогов 307
из файла 67
символа 70
строк произвольной длины 70

Т

Таблица 159, 165, 177
оптимизация 229
прав 217
создание 177
типы таблиц 165
Текст 352
вывод текста на кнопку 353
позиционирование текста 352
Тема дискуссии 571
Тестирование 394
Тип
BLOB 185
ENUM 186
SET 186
TEXT 185
Типы данных
даты и времени 183
с плавающей запятой 183
Типы переменных 36
приведение типов 37

У

Удаление файла 71
Управление сеансами 361
Управляющие структуры 50
Усечение строк 97
Условный оператор 50
Утилита myisamchk 229
Учетные записи 511

Ф

Файл 60, 302
блокирование файлов 72
двумерный 60
загрузка файлов на сервер 302
закрытие файла 66
запись в файл 65
конец файла 68
перемещение внутри файла 71
проверка существования файла 70
режим файла 61
считывание всего файла 69
считывание из файла 67
удаление файла 71
Файл
.htaccess 275
book_sc_fns.php 451
content.html 275, 277
creditcardnumbers.txt 291
footer.inc 121
header.inc 121
home.php 120
index.html 605
login.php 469
page.inc 151
Формат 598
ASCII 598
HTML 598
PostScript 600
RTF 599
переносимых документов (PDF) 596
Форматирование
вывода 480
строк 97, 98
Форматы изображений 341
GIF 342
JPEG 341
PNG 341
WBMP 342

Форматы файлов 65
Форум 571
Фрагментирование кода 387
Функции 125, 198
  PHP-MySQL 214
  возврат из функций 134
  для работы с переменными 48
  для работы с каталогами 307
  календарные 339
  обобщенные функции в MySQL 198
  перегрузка функции 129
  сетевого поиска 320
  хэш 259
Функция
  add_bm() 433
  add_quoting 592
  addslashes() 207, 290, 377
  array_count_values() 92
  array_reverse() 87
  array_walk() 91
  asort() 85
  basename($path) 309
  basename() 311
  calculate_items() 461
  calculate_price() 461
  change_password() 556
  check_auth_user() 510
  check_valid_user() 425, 426
  checkdate() 336
  chgrp() 312
  chmod() 312
  chown() 312
  copy() 312
  count() 92
  current() 90
  date() 33, 333
  DATE_FORMAT() 337
  db_connect() 423
  db_result_to_array() 452
  define() 38
  delete_account() 515
  delete_bm() 436
  dirname($path) 309
  display() 151
  display_account_form() 513
  display_account_select() 518
  display_account_setup() 512
  display_book_form() 473
  display_books() 454
  display_button() 509
Функция
  display_cart() 458
  display_categories() 451, 452
  display_form_button() 509
  display _items() 548
  display_list() 518
  display_list_form() 558
  display_mail_form() 560
  display_password_form() 555
  display_tree() 581
  dl() 378
  do.html_header() 417
  draw_star() 624
  each() 79, 81, 90
  empty() 49, 50
  end() 90
  ereg() 112, 113
  eregi() 112, 113, 319
  error_reporting() 407
  escapeshellcmd() 290
  eval() 375
  exec() 313
  expand_all() 580
  explode() 89, 102, 322
  extract() 93, 94
  fclose() 66
  feof() 68, 70
  fgetc() 70
  fgetcsv() 68, 69
  fgets() 68, 69, 70
  fgetss() 68
  (ile() 69
  fileatime() 311
  filegroup() 311
  filemtime() 311
  fileowner() 311
  fileperms() 311
  filesize() 71, 311
  filetype() 311
  filled_out() 421
  flock 72
  fopen() 61, 62, 63, 64, 65, 68, 69, 319
  fpassthru() 69
  fread() 70
  fseek() 72
  ftell() 71
  ftp_connect() 326
  ftp_fget() 328
  ftp_nlist() 330
  ftp_size() 329

Функция
fwrite() 65
get_account_list() 516
get_accounts{) 513
get.archive() 553
get_attribute() 144
get_categories() **451**, 452
get_category_name() 454
get_current_user() 378
get_email 546
get_extension_funcs() 377
get_loaded_extensions() 377, 378
get_magic_quotes_gpc() 375
get_post() 589
get_post_message() 592
get_random_word() 430
get_story_record() 493
get_unsubscribed_lists() 550
get_writer_record() 491
gethostbyname() 321, 322
getlastmod() 378
getmxrr{) 321
gettype() 48, 49
header() 346, 363
highlight_file() 379
highlight_string() 379
htmlspecialchars() 290
internet() 285
isset() 50
ksort() 85
ImageColorAllocate() 344
ImageCreate() 344
ImageDestroy() 347
ImageFill() 345
ImageFilledRectangle() 358
ImageGetTTFBBox() 351
ImageJPEG() 346
ImagePNG() 347
ImageString() 345
ini.get() 378, 379
ini_set() 378, 379
insert_order() 465
load_list_info() 552
login() 425, 545
lstat() 311
mail() 96
mkdir() 309
rnktimef) 335
myMultiply() 92
myPrint() 92

Функция
mysql_affected_rows() 214
mysql_connect() 209, 400
mysql_create_db() 214
mysql_db_query() 210
mysql_drop_db() 214
mysql_fetch_array() 210
mysql_fetch_object() 211
mysql_free_result() 214
mysql_num_rows() 214
mysql_numrows() 210
mysql_pconnect() 208, 213
mysql_query() 209, 210, 213
mysql_result() 211
mysql_select_db() 210
next() 90, 91
nl2br() 98
notify_password() 431
number() 47
number_of_accounts() 516
open_mailbox() 519
opendir() 308
parse_url() 322
passthru() 313
pos() 90
prev() 90
query_select() 494
readdir($dir) 308
readfile() 69
recommend_urls() 438
register() 422
rename() 312, 313
reset() 79, 90
reset_password() 429
retrieve_message() 522
rewind() 71, 72
rewinddir($dir) 308
rmdir() 309
send() 566
send_message() 526
serialize() 376
session_get_cookie_params() 363
session_register() 365
session_start() 364, 370
set_attribute() 144
settype() 49
show_source() 379
shuffle() 86
sort() 83, 85
srtok() 103

Функция
stat() 311
store_account() 543
store_account_settings() 514
store _new_post() 593
strip_tags() 290
strcasecmp() 104
strchr() 105
strcmp() 104
stripslashes() 101, 210
stristr() 106
strnatcm() 104
strnatcmp() 104
strore_list() 559
strpos() 106
strrchr() 106
strrpos() 106
strstr{} 105, 399
strtok() 102, 103
subscribe() 554
substr() 103, 107
system() 313
tempnam() 298
touch() 312
trim() 97, 207
umask() 309
UNIX_TIMESTAMP() 337
unlink() 71, 298, 312
unserialize() 376, 377
unsubscribe() 554
usort() 85, 86, 91
valid_email() 421

X

Холст 344, 350
Хранилище 388. *См. также* Архив
Хэш-функция 259

Ц

Цепочка 571. *См. также* Тема дискуссии
Цикл
do..while 57
for 56
while 55
Цифровая подпись 258
Цифровой сертификат 259

Ч

Чтение почты 516

Ш

Шифрование 256, 292
с закрытым ключом 257
с открытым ключом 258

Э

Экземпляр класса 143

Я

Язык 30
SGML 30
XML 30
манипулирования данными (Data Manipulation Language) 188
определения данных (Data Definition Language) 188
структурированных запросов (SQL) 187

Иностранные термины

Active Server Pages (ASP) 31
Cookie-набор 362
FTP-доступ 478
PHP 33
PHP-дескриптор 30
PHP-операторы 31
PHP-переменные 34
SGML (Standard Generalized Markup Language) 30
Web-форум 571
XML (Extensible Markup Language) 30



ISBN 966-7393-81-X,
384стр.

PHP. Руководство разработчика

Хьюгс Стерлинг и
Змиевский Андрей

Рекомендуемая цена	в Украине — 31 грн.
	в России — 155 руб.

Книга *PHP. Руководство разработчика* посвящена искусству программирования на языке PHP — серверном языке написания сценариев, внедряемых в HTML-страницы. Благодаря открытому коду, постоянному развитию и пополнению все новыми и новыми модулями, язык PHP превратился в один из наиболее полнофункциональных инструментальных средств для быстрого создания Web-сайтов, реализующих задачи электронной коммерции, баз данных для Web и различного рода порталов.

Книгу отличает ориентация на решение конкретных задач. Четкий и продуманный характер подобранных задач позволяет получить рецепты на все случаи повседневной практики программирования. Каждое решение подробно комментируется, что позволяет не только понять методологию программирования, но также и стимулирует нахождение наиболее эффективных способов решения сложных проектов.

Рекомендуется в качестве настольного руководства для разработчиков Internet-приложений любой квалификации.



ISBN 966-7393-80-1,
448 стр.

PHP. Справочник

ВайкАллен

Рекомендуемая цена	в Украине — 29 грн.
	в России — 150 руб.

Книга *PHP. Справочник* содержит наиболее полный материал по языку программирования PHP. Удобная форма представления материала, изобилие примеров применения, исчерпывающее описание функциональности превращают эту книгу в незаменимое руководство, которое должно находиться на столе каждого профессионального разработчика Web-приложений.

Справочник охватывает все аспекты применения языка PHP: базовый синтаксис, функции генерации содержимого Web-страниц, работа с Internet-протоколами, манипулирование документами, организация доступа к разнообразным базам данных, использование системных функций. Учитываются все нововведения последних версий PHP.

Справочник ориентирован на широкую категорию программистов и может рассматриваться в качестве эффективной замены официальной документации по языку PHP.



ISBN 966-7393-95-X,
464 стр.

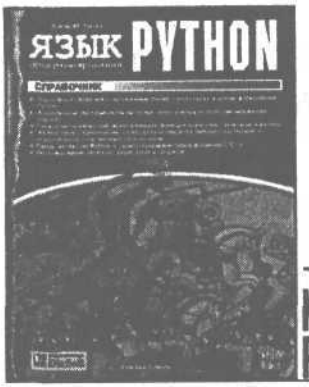
Динамический HTML. Руководство разработчика Web-сайтов

Вайнман Линда и Вильям

Книга *Динамический HTML Руководство разработчика Web-сайтов* является довольно необычной. Написанная Web-программистом и Web-дизайнером, она сочетает строгую логику программных разработок и неожиданные спецэффекты, характерные для дизайна. На протяжении всей книги читатель вместе с авторами принимает участие в разработке корпоративного Web-сайта Ducks In A Row (http://www.ducks.htmlbook.com). Процесс разработки сайта построен по принципу «от простого к сложному». Всесторонне рассматриваются возможности языка HTML 4, а также их практическое воплощение. Обилие иллюстраций и сопровождающий книгу CD-ROM значительно ускоряет и упрощает процесс обучения.

КРАТКОЕ ОГЛАВЛЕНИЕ

Глава 1. Начало
Глава 2. Первая страница
Глава 3. Быстрая обработка графики
Глава 4. Работа с Web-цветами
Глава 5. Могучая сила щелчков
Глава 6. Мозаичные фоновые изображения
Глава 7. Маркеры и линейки
Глава 8. Прозрачность
Глава 9. Выравнивание
Глава 10. Работа со шрифтами
Глава 11 Планирование
Глава 12. Организация
Глава 13. Таблицы стилей
Глава 14. Навигация
Глава 15. Интерактивные элементы управления
Глава 16. Формы
Глава 17. Анимация и звук
Глава 18. Включение в списки поисковых систем
Глава 19. Хороший стиль HTML-программирования
Глоссарий
Предметный указатель



Дэвид М. Бизли

Язык программирования Python. Справочник

ISBN 966-7393-54-2
175x250 мм, 336 с., мягк. переплет

Рекомендуемая цена | в Украине — 29 грн. | в России — 150 руб.

Первая книга на русском языке, посвященная интерпретируемому, интерактивному, объектно-ориентированному языку высокого уровня с открытым исходным кодом

Язык программирования Python получил широкую известность сравнительно недавно, но теперь его достоинства начинают признавать очень многие.

Этот язык привлекает программистов:

- Является интерпретируемым языком, поэтому он широко применяется для создания сценариев операционной системы (например, в Linux и UNIX). Сценарии, написанные на языке Python, практически не требуют отладки, поэтому он является повседневным инструментом системных администраторов. Он широко применяется для создания сценариев CGI (на основе модуля mod_python для сервера Apache). На этом языке пишутся сценарии для многих известных программных комплексов, например таких, как Delphi, COM+, MFC, Java, поскольку он значительно упрощает работу с этими программными комплексами.
- В программах, написанных на этом языке, можно легко разобраться, поскольку он имеет очень простой синтаксис. Строки документирования программ можно вызывать для справки непосредственно в интерпретаторе, строить на их основе документацию к программам.
- Это объектно-ориентированный язык, вплоть до уровня структур данных. В связи с этим работа программиста значительно упрощается. Модульные, не зависящие друг от друга программные компоненты можно легко отлаживать отдельно, а затем объединять в единый комплекс. Для любых, самых сложных структур данных предусмотрен оптимальный набор методов. Об их создании программисту заботиться не приходится.
- В языке Python структура программы обозначается отступами. Его программы эстетически привлекательны: никаких фигурных скобок, которые иногда так трудно согласовать между собой (C, C++), никаких режущих глаз знаков доллара, обратного слэша, амперсанда, в нагромождении которых просто теряешься (Perl).

Этот язык привлекает математиков:

- Поддерживает целые числа неограниченной длины (единственным ограничением является объем памяти компьютера). В нем есть поддержка комплексных чисел, встроены мощные функции работы с последовательностями (например, map, filter, reduce, zip), предусмотрена анонимная функция (lambda).
- На этом языке удобно проводить эксперименты со структурами данных, реализовать сложные алгоритмы, проверять гипотезы. Например, в модуле AVL реализована разработанная Адельсоном-Вельским структура, которая одновременно является сбалансированным деревом и списком. В модуле PLEX по исходным данным пользователя осуществляется синтез недетерминированного, а затем детерминированного конечного автомата, который применяется в качестве лексического анализатора.



Андре Дос-Сантос Лесса

Python. Энциклопедия программиста

ISBN 966-7393-97-6
175x250 мм, мягк. переплет

Python — мощный интерпретируемый объектно-ориентированный язык программирования высокого уровня, идеально приспособленный для Web. Он развивается в рамках проекта с открытым исходным ходом и предоставляется бесплатно для всеобщего пользования.

Среда программирования Python реализована на многих платформах, включая Microsoft Windows, UNIX/Linux и Macintosh. Исходный код интерпретатора и библиотек Python на языке C++ предоставлен для общего пользования.

Еще одна реализация этого языка, получившая название JPython и также общедоступная, написана на языке Java.

Благодаря этим особенностям язык программирования Python открывает путь ко многим программным технологиям, поэтому разработки на этом языке ведутся во многих направлениях. К наиболее интересным и перспективным из них относится:

- создание серверов и клиентов OLE, COM, DCOM и ActiveX для предоставления удобных сценарных средств в среде Windows (пакет win32com, проект Python for Delphi и др.);
- упрощенная поддержка распределенных объектов в рамках технологий CORBA, ILU, ADO, DCOM (проекты CORBA IDL Parser, Fnorb, OMF, Hector и др.);
- интерпретация исходного кода таких языков, как Java, которые в основной реализации поддерживают только компиляцию (проект JPython и др.);

Книга содержит все, что необходимо разработчику, использующему язык программирования Python

- разработка более дружественных, альтернативных интерфейсов к существующим библиотекам объектных модулей, таким как MFC, Tk, KDE, Motif и многим другим (проекты Pythonwin, wxPython, PyKDE, PyOpenGL и пр.);
- создание объектно-ориентированных интерфейсов к базам данных (проект Zope, MetaKit, модуль shelve);
- реализация сложных структур данных, включая многомерные массивы, поддержка графических и других двоичных файловых форматов;
- обработка данных, в том числе с использованием таких языков разметки, как XML, SGML, HTML, XDR.

Все эти направления разработки, а также многие другие подробно рассматриваются в книге "Python. Энциклопедия программиста". В ней даны многочисленные примеры, ссылки на все существующие источники в Web.

Книга содержит все, что необходимо разработчику, использующему язык программирования Python: дано описание синтаксиса, основных средств и модулей, среды разработки, способов оптимального использования мощных средств языка.



Аллен Вайк, Ричард Вагнер

JavaScript Энциклопедия пользователя

ISBN 966-7393-62-3
210x270 мм, 464 с., мягк. переплет

Рекомендуемая цена	в Украине —	55 грн.
	в России —	289 руб.

Методика использования JavaScript как на стороне клиента, так и на стороне сервера

Язык JavaScript за последние несколько лет превратился в одно из основных инструментальных средств разработки продукции для Web. С его помощью можно не только усовершенствовать статические Web-страницы, но и создавать полнофункциональные Web-приложения, в том числе и целые Web-сайты.

Книга *"JavaScript. Энциклопедия пользователя"* — это учебник и справочное руководство для широкого круга разработчиков Web-продукции. Традиционно JavaScript применялся для "оживления" Web-страниц за счет добавления разнообразных эффектов — от изображений, реагирующих на действия пользователей, до сложных алгоритмов верификации заполненных форм. Подобного рода функциональность на стороне клиента приводит к существенному уменьшению нагрузки на и без того загруженный сервер. Кроме того, с помощью JavaScript можно создавать серверные сценарии — реализовать, скажем, собственную систему бухгалтерского учета и тем самым сэкономить немалую сумму денег. А что скажете по поводу создания системы электронных заказов, полностью размещенной на сервере? Конечно, существует множество других возможностей сделать это, например, использовать языки Perl или Python. Однако не забывайте, что на освоение нового языка уходит немалое время, да и грешно не воспользоваться собственным опытом разработки на JavaScript.

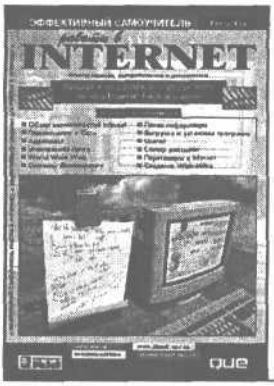
Особенности этой книги

Вот лишь небольшой список особенностей, выгодно отличающих ее от других книг подобной тематики.

- Книга имеет четкую практическую направленность, т.е. она написана *программистами* для *программистов*.
- Изобилие примеров *учебного* характера, которые помогают точно и быстро ухватить суть того или иного аспекта языка, а также решений *реальных* бизнес-задач позволит немедленно приступить к написанию собственных приложений.
- Хорошо организованный и удачно преподнесенный справочный материал по базовым объектам JavaScript позволяет использовать книгу и в качестве удобного справочника.
- В книге подробно анализируется множество современных технологий программирования на динамическом HTML, знание которых — ключ к успешной разработке удачных, эффективных и, самое главное, эксклюзивных Web-сайтов.
- Методика использования JavaScript как на стороне клиента, так и на стороне сервера, позволяет максимально адекватно и эффективно построить собственную технологию Web-дизайна для широкого спектра применений.

Подробно рассматриваются

- Основы JavaScript и его взаимодействие с HTML.
- Использование операций, выражений, операторов, управляющих структур и функций.
- Особенности реализации технологии клиент/сервер в JavaScript.
- Фундаментальные основы объектной модели документа.
- Наиболее распространенные технологии создания динамического содержимого Web-страниц.
- Типовые приемы эффективного решения реальных бизнес-задач.



Харли Хан

Эффективный самоучитель работы в Internet

ISBN 966-7393-70-4
150x210 мм, 448 с., мягк. переплет

PUE

Рекомендуемая цена	в Украине —	16 грн.
	в России —	125 руб.

Internet — обширная информационная система, которая стала очень важным изобретением в истории человечества.

Построенная на основе компьютеров, программ и линий связи, сеть Internet в действительности представляет собой систему взаимодействия людей и информации. Мы создали Internet во имя удовлетворения своих потребностей в общении и приобретении знаний.

Особенности книги

Почему эта книга мало напоминает традиционные самоучители по использованию Internet? Прежде всего тем, что ее написал всемирно известный автор, аналитик и консультант. Кто не слышал (не видел, не использовал) знаменитые "Желтые страницы" Харли Хана, с завидной периодичностью выходящие и отражающие актуальное состояние ресурсов Internet? Кроме того, Харли Хан гарантирует решение, пожалуй, наиболее сложных проблем, возникающих у каждого, кто сталкивается с Internet.

- Осветит все, что требуется для начала работы: как выбрать компьютер, как подключиться к Internet и как отыскать нужные программы.
- Введет читателей в мир технических терминов, которые встречаются в литературе, посвященной Internet. Будет поясняться каждое слово и указываться способ его корректного применения.
- Пояснит основные концепции, на которых построена работа в Internet, а также использование программ в соответствие с этими концепциями.
- И самое главное, автор дает читателям глубокое понимание того, что они делают и как это соответствует Internet в целом. Он поможет осознать возможности и красоту того, что можно наблюдать ежедневно и ежечасно в Internet.

> Усилиями автора читатели получают глубокое понимание того, *что* они делают и как это соответствует Internet в целом

Книга оформлена так, что наиболее важные термины, утверждения и аксиомы представляются в доступном для понимания виде.

Чтение книги не только обогатит полезной информацией, но и доставит удовольствие — это не просто книга, посвященная компьютерам.

Автору хотелось бы заставить читателей обдумать важные концепции. Каждый человек наделен разумом, творческим потенциалом и любознательностью. Многим никогда не удается реализовать свой потенциал, поскольку они позволяют себе поддаться лени или нерешительности. На жизненном пути такие люди нередко утрачивают (или не приобретают) уверенность в собственных силах. Они боятся новых идей и в результате обречены на невозможность понять и овладеть новой технологией. Автору хочется думать, что этого не произойдет.

В ходе изложения материала демонстрируются не только возможности Internet. Будет показано, как работать и как изящно и профессионально выстраивать свои мысли, как открывать те двери, о существовании которых не было известно.

Стать частью Internet — биологический удел всего человечества. Возможно, в это трудно поверить, тем более сразу, однако эта книга может полностью изменить ваше мировоззрение.



Кен Мильберн, Джон Крото

Внутренний мир Flash 5 для дизайнера

ISBN 966-7393-61-5
175x250 мм, 496 с., мягк. переплет

Рекомендуемая цена	в Украине —	29 грн.
	в России —	150 руб.

Своей популярностью в качестве мультимедийного приложения Flash обязана компактности, быстродействию, интерактивности и практически неограниченным возможностям оформления фильмов. Усовершенствованный интерфейс, более высокая интеграция с другими программными продуктами, внедрение многих новых свойств, в том числе встроенного языка создания сценариев ActionScript, — все это в Flash 5 позволяет создавать качественную мультимедийную продукцию, включающую в себя графику, анимацию, звук, текст и интерактивные элементы, как для Web, так и для автономного применения.

Книга "Внутренний мир Flash 5 для дизайнера" дает возможность углубленно изучить новые свойства Flash 5, касающиеся графического оформления, ввода и редактирования текста, анимации, звукового сопровождения, интерактивности и автоматизированной публикации. Особое внимание в книге уделяется нововведениям в 5-й версии, взаимодействию Flash с другими программными продуктами, в том числе и с программами компании Macromedia, а также использованию сценариев, написанных на языке ActionScript, для автоматизации рабочего процесса во Flash с помощью . Удобным для читателя окажется и приведенный в книге краткий справочник по Flash 5, который позволяет быстро найти описание конкретных свойств, функций и команд и соответствующих им оперативных клавиш. Книга также содержит подробную техническую информацию по рассматриваемой программе, многочисленные упражнения, иллюстрируемые графикой, способной удовлетворить разные вкусы, а также полезные советы и рекомендации авторов. Кроме того, в книге приведены многочисленные ссылки на Web-сайты учебного материала по Flash, разработанные одним из авторов книги Джоном Крото, который считается всемирно известным специалистом по Flash.

Отличительные особенности книги

- Книга написана для тех, кто делает свои первые шаги во Flash, в том числе и тех, кто не умеет толком рисовать, но горит желанием создать привлекательный Web-сайт, хотя пользу от этой книги, несомненно, извлекут и опытные Web-дизайнеры и художники-оформители.
- В книге рассматриваются приемы и методы работы не только во Flash, но и в таких графических и мультимедийных программных продуктах компании Macromedia, как FreeHand, Fireworks, Dreamweaver, Generator и Director. Подобная взаимосвязь программ подчеркивается на протяжении всей книги, что позволяет в полной мере оценить возможности Web-дизайна и подготовки автономных презентаций во Flash совместно с упомянутыми программами.
- Сценарии, написанные на языке ActionScript, в значительной степени повышают интерактивность Flash-содержимого, хотя и требуют некоторых усилий при создании. Тем не менее приведенный в книге материал может утешить тех читателей, которые напуганы перспективой освоения программирования, поскольку возможности Flash позволяют создавать сценарии и без особых навыков в программировании.
- Многочисленные ссылки на сайты учебного материала по Flash, созданные одним из авторов этой книги, Джоном Крото, послужат, наряду со множеством полезных приемов и советов, отличным подспорьем в оперативном освоении Flash и углубленном изучении ее возможностей.



Девис Чепмен и др.

Разработка Internet-приложений в Delphi 2

ISBN 966-7033-20-1
150x210 мм, 640 с., тв. переплет

PUB

Рекомендуемая цена	в Украине —	9 грн.
	в России —	45 руб.

Эта книга — учебное пособие, посвященное разработке приложений для Internet в среде Delphi. В ней описаны протоколы и службы Internet, алгоритмы защиты сетевой информации, электронная почта и группы новостей.

Авторы нашли оптимальное равновесие между изложением основных принципов программирования в Internet и практическими примерами того, как эти принципы воплощаются в реальность.

Читателю предоставляется возможность испробовать все — от создания приложений клиента до расширения серверов и Web-броузеров.

Прочитав книгу, вы узнаете все секреты протоколов Internet, научитесь применять CGI для организации взаимодействия с Web-серверами, сможете разрабатывать высокоэффективные приложения для Internet

Вы узнаете все секреты протоколов Internet, научитесь применять CGI для организации взаимодействия с Web-серверами, сможете разрабатывать **высокоэффективные** приложения для Internet

Пособие рассчитано на широкий круг пользователей, владеющих азами программирования. В настоящее время книгу можно приобрести за сравнительно небольшую цену, а рассматриваемые вопросы применимы и для новейших версий Delphi.

CD-ROM содержит исходные коды всех примеров, рассмотренных в книге, наборы полезных компонент и утилит.



Роберт Седжвик

Фундаментальные алгоритмы на C++
Анализ/Структуры данных/
Сортировка/Поиск

ISBN966-7393-89-5

Рекомендуемая цена	в Украине — 37 грн.
	в России — 195 руб.

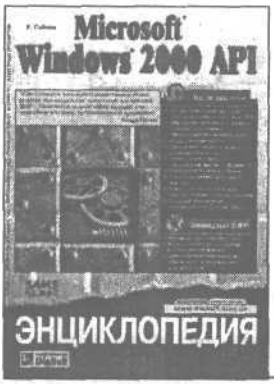
Эта книга посвящена глубокому исследованию основополагающих концепций и алгоритмов, которые можно отнести к разряду "вечных". Изучив их, вы получите знания, которые никогда не устареют и которыми вы будете пользоваться всегда

Когда человек берет в руки книгу компьютерной тематики, он наверняка задастся такими вопросами: насколь актуальна информация, изложенная в книге? Когда была издана книга? Знакома ли фамилия автора? На какой период книга сохранит свою актуальность? Вопросы далеко не праздные: приобретение книги связано с тратой определенной суммы денег, поэтому человек вправе ожидать как можно большего эффекта и желательно как можно дольше.

Не секрет, что в последние годы программирование, перейдя "на поток", перестало быть искусством, а стало ремеслом. Проникновение компьютеров практически во все сферы повседневной жизни привело к существенному увеличению спектра задач, решаемых программистами. Вполне естественно появление и постоянное совершенствование инструментов разработки, позволяющих даже начинающим без особых затрат времени и усилий решать свои маленькие (и не только) задачи. Представьте себе создание приложения для пополнения, сортировки и поиска в большой информационной системе по каким-то товарам. Вы создаете на экране окошко (естественно, красивое), перетаскиваете туда элементы "Таблица", "Сортировка", "Поиск" и так далее. Затем среди свойств элемента "Сортировка" выбираете "Быстрая" (разумеется, ведь информации у вас немерянно!)... Звучит заманчиво, не правда ли? А что же такое сортировка, к тому же быстрая? А как, собственно, организуется поиск (кроме способа, который первым приходит в голову, - искать простым перебором и сравнением информационных элементов)? Как вообще обстоят дела с алгоритмами в настоящий момент?

А ситуация такова: структурное (алгоритмическое) программирование не забыто, да и никогда не будет забыто просто потому, что сами по себе алгоритмы гораздо старее программирования. Ими пользовались еще до появления не только компьютеров, но даже и мыслей подобного рода. Это классические понятия, которые просто скрылись под разного вида оболочками и надстройками.

И эта книга как раз и посвящена глубокому исследованию всех основополагающих концепций и алгоритмов, которые можно отнести к разряду "вечных". Изучив их, вы получите знания, которые *никогда* не устареют и которыми вы будете пользоваться *всегда.* Ибо они - фундаментальная основа успешного Программирования (именно так, с большой буквы).



Ричард Саймон

Windows 2000 API. Энциклопедия программиста

ISBN 966-7393-74-7
175x250 мм, 1088, мягк. переплет

SAMS

Рекомендуемая цена в Украине — 64 грн.
в России — 320 руб.

Система Windows является непревзойденной платформой для разработки приложений на основе окон (эта операционная система неслучайно носит такое название, ведь "windows" в переводе с английского означает "окна"). В системе Windows в виде окон выполнены все элементы пользовательского интерфейса: кнопки, линейки прокрутки, меню, пиктограммы, поля, списки и, естественно, сами окна (главное окно приложения, диалоговые окна, дочерние окна, модальные окна, всплывающие окна).

Окна могут принимать и обрабатывать сообщения, возникающие в результате действий пользователя с клавиатурой и мышью, передавать сообщения самим себе, своим дочерним окнам, другим окнам и самой системе, ставить сообщения г очередь и извлекать их из очереди. Очереди сообщений позволяют выполнять в системе сразу несколько приложений, координировать их работу и создавать в приложениях подпроцессы для лучшего использования выделенных ресурсов.

Сообщения вырабатываются приложением или возникают в ответ на событие. Событием может являться любое действие, выполненное пользователем (щелчок или двойной щелчок кнопкой мыши, ведение курсора над определенной областью окна, нажатие клавиши на клавиатуре), а также любое изменение состояния системы.

Это — стройная, глубоко продуманная программная технология, которая годами шлифовалась и доводилась до совершенства. Она позволяет легко и быстро воплотить в жизнь любой замысел — при одном условии: разработчик должен уметь пользоваться интерфейсом прикладного программирования (API — application programming interface) системы Windows и иметь под рукой описания всех функций этого интерфейса. Незаменимым источником этой информации является книга "Windows 2000 API. Энциклопедия программиста".

Содержится полное описание всех функций с подробным перечнем их параметров и приведены примеры, позволяющие понять назначение функций

Назначение книги

Издание позволит программистам сэкономить время при разработке приложений для всех современных версий системы, начиная с Windows 95. Новые API и стандарты, предусмотренные в Windows 2000, стали еще более сложными и развитыми, что требует от разработчика дополнительных усилий по их освоению. В книге не только подробно описаны новые средства Windows, но и включены основные разделы API, применяемые большинством программистов.

Книга предназначена для использования в качестве полного справочника по функциям и средствам API Win32, которые поддерживаются в системах Windows 2000 и Windows 98 компании Microsoft. Описана область применения каждого API, что позволит разработчикам создавать приложения, применимые в обеих операционных системах. В данной книге в основном рассматриваются API Win32, поддерживаемые в обеих операционных системах, а не вся совокупность API Win32, поскольку для большинства приложений не требуются API, предназначенные только для Windows 2000.



Ральф Вебер

Сборка, конфигурирование, настройка, модернизация и разгон ПК. Энциклопедия пользователя

ISBN 966-7393-45-3
150x210 мм, мягк. переплет

Книга будет полезна не только любителям-конструкторам компьютерных систем, но может служить справочным пособием для опытных пользователей!

Назначение книги

Открыв свежий сборник прайс-листов по компьютерной литературе, вы найдете там более четырех десятков названий книг, в которых так или иначе описываются аппаратные компоненты ПК, а также вопросы его сборки, настройки и модернизации. Однако развитие компьютерной техники происходит настолько быстро, что любая книга, посвященная этой тематике, неизбежно быстро устаревает. Поэтому новый перевод книги Ральфа Вебера (предыдущий вышел в 1998 году в издательстве "МИР"), дополненный последними новостями об аппаратно-программной реализации современного компьютера, представляется вполне обоснованным.

Книга представляет собой руководство по аппаратным средствам и программному обеспечению персональных компьютеров на базе наиболее популярных процессоров фирм Intel, AMD, Cyrix и других. Редакторы этого перевода добавили новые данные, связанные с рассматриваемыми в книге вопросами, в результате чего в ней объединены традиционные, в основном неизменные сведения об аппаратном и программном обеспечении компьютера с новыми данными по современным процессорам, видеокартам, чипсетам. Дополнительный материал касается также вопросов разгона процессоров, обеспечивающего повышение мощности вашего компьютера. Таким образом, эта книга будет полезной не только любителям-конструкторам компьютерных систем, но может служить справочным пособием для опытных пользователей и конструкторов персональных компьютеров.

Отличительные особенности *книги*

Эту книгу можно рассматривать как универсальное руководство по аппаратно-программной реализации современного ПК на базе новейших процессоров фирм Intel, AMD, Cyrix и других. Здесь вы найдете подробное описание устройства и различных характеристик процессоров — от Intel 8088 и AMD486DX до Pentium 4, Tualantin и Athlon, материнских и системных плат, видеокарт, чипсетов, шин и интерфейсов. Это поможет вам сделать обоснованный выбор элементов для правильного конфигурирования всего аппаратного обеспечения ПК.

В книге прекрасно освещены проблемы модернизации ПК и все связанные с ней вопросы. Кроме того, здесь описана настройка BIOS и программного обеспечения ПК, а также использование современных операционных сметем — от Windows 95 до Millenium 2000.

Особое место в книге занимает рассмотрение разгона процессоров. Вас не оставит равнодушным возможность придания дополнительной мощности ПК за счет его разгона. Здесь даны практические рекомендации по разгону процессоров, мерам предосторожности и устранению возможных проблем.

Наконец, полезные технические данные и таблицы, приведенные в заключительной главе книги, вы можете использовать для правильного выбора конфигурации и параметров настройки своего ПК.



Ричард Лейнекер

COM+. Энциклопедия программиста

ISBN уточняется
175x270мм, мягк. переплет

Предлагается эффективное решение проблем, возникающих при работе с недостаточно подробной и трудной в изучении документации по COM+

Технология COM+ превратилась в составную часть DNA - архитектуры распределенных межсетевых приложений, весьма широко применяемой разработчиками программного обеспечения. COM+ предоставила инфраструктуру, которая обеспечивает быстрое, простое и эффективное создание многоуровневых приложений практически без затрат на кодирование.

Назначение книги

Автор поставил перед собой благородную цель восполнить пробелы, связанные с недостаточно подробной и трудной в изучении документации по COM+. Другая цель - обзор разнообразных сфер применения COM+, среди которых самые непредсказуемые. Именно в этой книге развеяна завеса таинственности над Windows DNA и соответствующими серверными продуктами.

Подробное рассмотрение COM+ заняло бы несколько томов. Тем не менее существуют определенные базовые, основополагающие аспекты, без которых знание, а тем более применение этой технологии попросту невозможно. Именно они подробнейшим образом описываются в книге. Таким образом, книга как бы закладывает фундамент для дальнейшего совершенствования мастерства. Любая сложная технология, коей может считаться и COM+, имеет множество ловушек и подводных камней. Каково, например, тому, кто часами корпел над кодом и затем во время демонстрации заказчику получил столько сообщений об ошибках, что хватило бы на всю оставшуюся жизнь? А проблема-то заключалась вовсе не в неправильном кодировании, а в том, что начинающий программист не исследовал все "узкие места" COM+. Где же ему отыскать информацию, если в документации ее попросту нет? Да, именно в этой книге! Имея за плечами огромный опыт использования COM+, Ричард Лейнекер любезно делится им с читателями, и все что им остается, - лишь внимательно прочитать книгу и не наступать на те же грабли, на которые наступили до них другие программисты.

Особенности этой книги

Книга состоит из четырех взаимосвязанных частей. В первой части, "Windows DNA и COM+", рассматривается объектная модель компонентов (COM) и ее связь со средой Microsoft DNA. Основные продукты серверной части и средства разработки исследуются в контексте DNA. Помимо прочего, в этой части представлены принципы дизайна систем с многоуровневой архитектурой.

Вторая часть, "Усовершенствованные технологии программирования COM+", расширяет базовый набор знаний за счет глубокого исследования концепций долговременного хранения, событий, имен, потоков. Немалое внимание уделяется скрытым свойствам COM+.

В третьей части, "Управление компонентами и транзакции", детально описывается сервер транзакций Microsoft Transcation Server и службы COM+. Эти ресурсы и службы оказываются более мощными, чем транзакции, осуществляемые через брокеров.

Четвертая часть, "Асинхронное компонентное программирование", посвящена рассмотрению MSMQ и компонент с реализацией очередей COM+, которые вместе обеспечивают асинхронную связь в современных информационных средах, являясь залогом высокой степени отказоустойчивости при взаимодействии множества приложений.

КНИЖНЫЙ КЛУБ «КОМП@С»
(ОФИЦИАЛЬНЫЙ ПАРТНЕР ТОРГОВО-ИЗДАТЕЛЬСКОГО ДОМА «ДС»)

Приглашаем всех желающих вступить в книжный клуб «Комп@с». Вступив в клуб, Вы получите скидки, содействие в поиске и покупке необходимых Вам книг. Мы своевременно реагируем на запросы читателей и своей издательской деятельностью способствуем их профессиональному росту. Вы раньше других будете иметь возможность познакомиться с новинками, получая наш прайс-лист, иллюстрированный каталог по почте либо обратившись на наш сайт www.diasoft.kiev.ua.

Вы сможете: пользоваться системой скидок при покупке книг;
высказать свое мнение, пожелания и замечания относительно книг издательства;
найти новых друзей и партнеров по интересам;
заочно встретиться и пообщаться с авторами книг;
участвовать в розыгрыше призов бесплатной лотереи.

И главное, впервые членам клуба будет предоставляться **бесплатное** комплексное программное обеспечение, позволяющее осуществлять связь с книжной базой данных, получать оперативные изменения в базе, компоновать заказы, получать информацию о книгах издательств - объем, содержание (краткое или глобальное), аннотация, текст одной из глав, внешний вид обложки и т.д. С более подробной информацией Вы можете ознакомиться на сайте www.diasoft.kiev.ua

ПРИЕМ В КЛУБ

Для того чтобы стать членом книжного клуба, достаточно купить одну из книг нашего издательства в любой торговой точке, заполнить анкету, помещенную в книге, и отправить ее по адресу издательства либо купить три любых книги на одном из торговых мест ТИД «ДС».

Все члены клуба получают карточку с индивидуальным номером, дающую право на скидку.

Карточка выдается сразу же при покупке на торговом месте ТИД «ДС» или у партнеров ТИД «ДС» (адреса смотри на следующей странице). В остальных случаях (заказ по почте, через электронный магазин, а также покупка у независимых торговых организаций книги ТИД «ДС») Вам необходимо заполнить анкету, помещенную в книге, и отправить ее в издательство. Действие карточки распространяется на все торговые точки, перечисленные в этом пункте, кроме независимых торговых организаций.

СИСТЕМА СКИДОК

Став членом книжного клуба, Вы получаете первоначальную скидку в размере 5% на весь ассортимент книг *(книги ТИД «ДС» + более 1000 наименований книг издательств Украины и стран СНГ).*

Карточка действительна на протяжении года со дня выдачи; по истечении срока подлежит замене.

При покупке в течение квартала пяти и более книг карточка заменяется на новую с 10% скидкой.

Более подробную информацию о системе скидок можно получить, обратившись на наш сайт по адресу www.diasoft.kiev.ua.

АНКЕТА

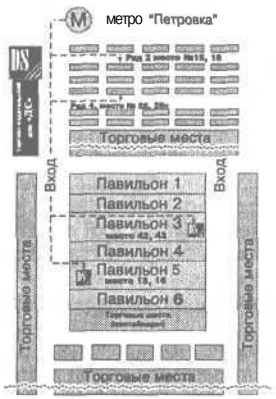
1. Название книги _____
2. Где, по какой цене, когда (с точностью до месяца) Вы приобрели эту книгу? _____
3. Ф.И.О. _____
4. Кем и где работаете _____
5. Параметры Вашего ПК _____
6. Адрес, телефон, e-mail, web-узел _____

Дата заполнения "__" _____ _____ г.

Вышлите анкету по адресу: Украина, 03055, Киев-55, а/я 100, «Издательство «ДиаСофт».

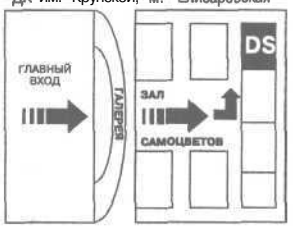
ТОРГОВЫЕ **МЕСТА ТОРГОВО-ИЗДАТЕЛЬСКОГО ДОМА «ДС»**

КИЕВ



САНКТ-ПЕТЕРБУРГ

Пр-т Обуховской обороны, д. 105
ДК им. Крупской, м. "Елизаровская"



ХАРЬКОВ



ДНЕПРОПЕТРОВСК



ОДЕССА



ПАРТНЕРЫ ТИД «ДС»

Киев
т./ф. 212-12-54,216-35-64
Книга-почтой: 03055, а\я 100
e-mail: bo Dks@diasoft.kiev.ua
e-mail: bss@diasoft.kiev.ua
e-mail: stepanb@akcecc.kiev.ua

Днепропетровск
т.(0562)33-27-74
Книга-почтой: 49008, а\я 466
e-mail: diasoff@mail.dnepr.net

Харьков
т.(0572)47-20-67
e-mail: books@rail.kharkov.com

Львов
т./ф. (0322)39-87-08

Одесса
т.(0482) 68-73-99
e-mail: kvant@eurocom.od.ua
e-mail: kvant@tekom.odessa.ua

Москва
т.(095) 726-80-67
e-mail: diasoft_msk@rosmail.ru

Санкт-Петербург
т.(812) 317-97-56
e-mail: diasoft_spt@mail.convey.ru

АДРЕСА МАГАЗИНОВ, ГДЕ МОЖНО КУПИТЬ КНИГИ ТИД «ДС»

КИЕВ
"Знання",ул.Крещатик,46
т.224-22-91
Техническая книга",
ул.Красноармейская,51
т.227-25-86
"Сучасник", пр-т Победы, 29
т.274-52-35
"Книжковый світ",пр-т Победы,2
т.219-26-17

ДНЕПРОПЕТРОВСК
Магазин "Технічна книга"
пл.Островского, 1
телефон (0562) 33-09-55
Магазин Техническая книга"
пр.К.Маркса 40
телефон (0562)744-86-72

ДОНЕЦК
"Дом книги"ул.Артема 147 а
телефон (0622) 55-74-49

КРИВОЙ РОГ
"Букинист-Солон"
пл.Освобождения 1
телефон (0564) 29-81-21

ЛЬВОВ
Техническая книга", пл.Рынок,10
т.72-54-06
"Влас", Университет Львовский
политехник,корпуса 4,5, т.39-8708

ЛУГАНСК
"Книги",ул.Советская, 58
т.53-62-30

ХАРЬКОВ
"Вища школа" ,ул.Петровского,6
т.47-80-20
"Книги 3", ул.Полтавский шлях,37,
т.12-55-27

МОСКВА
"Академкнига" ,ул.Вавилова,55
т.124-92-02
"Библио-Глобус",ул.Мясницкая,6
тел.928-87-44
"Дом технической книги",
Ленинский пр-т,40,т.137-60-38
"Московский дом книги",
ул.Новый Арбат, 8
"Мир",Ленинградский пр-т,78
т.152-45-11
"Энергия", ул.В.Кожиной,10
т.145-52-00
"Мир печати" ул.2-я Тверская Ямская,54
т.978-50-47, 978-55-07
"Молодая гвардия", ул.Большая Полянка, 2S
т.238-11-44, 238-00-32

САНКТ-ПЕТЕРБУРГ
"Дом книги",Невский пр-т,28
т.318-64-16
Техническая книга",
ул.Пушкинская,2,т.164-65-65
"Энергия", Московский пр-т,189
т.443-01-47

МИНСК
"Книга XXI"
пр-т Ф.Скорины, 92,
ст.м. Московская,
т.64-31-05, 64-27-97

Как сделать заказ и получить книги

Для организаций

1) Получите полный прайс-лист по адресу, указанному ниже
2) Аккуратно заполните бланк заказа
3) Отправьте его нам по факсу, почте или e-mail
4) В течении одного рабочего дня в Ваш адрес будет направлен счет по факсу или e-mail. В течении срока действия счета мы гарантируем наличие книг и их цену
5) После оплаты счета Ваш заказ, а также оригинал счета, расходная и налоговая накладные будут высланы Вам посылкой

Для частных лиц

1) Получите полный прайс-лист по адресу, указанному ниже
2) Аккуратно заполните бланк заказа
3) Отправьте его нам по факсу, почте или e-mail
4) В течении одного рабочего дня в Ваш адрес будет направлен счет по факсу или e-mail. В течении срока Действия счета мы гарантируем наличие книг и их цену.
   Внимание! Не оплачивайте покупку до получения счета
5) Оплатите счет в любом коммерческом банке Деньги можно отправить на наш расчетный счет и почтовым переводом, но это несколько дороже, чем через банк.

Заказы поступившие по электронной почте обрабатываются в первую очередь.
Заказы «наложенным платежом» — не принимаются.
Цены на книги не учитывают стоимость доставки, коммисионные банка или почтового перевода.
Стоимость доставки по Украине или курьером по Киеву — 7 грн.
Стоимость доставки по России — в зависимости от региона.

Заказ

от «_____» _____ 2001 г

Название организации: _____
Для организаций

Фамилия Имя Отчество: _____
Для частных лиц

Адрес доставки: _____
Почтовый индекс

ИНН: _____ № св-ва плательщика НДС: _____
Только для организаций

Фамилия Имя Отчество: _____
Контактного лица

Телефон: /_____/_____ Факс: _____

E-mail: _____

Код	Название книги	К-во

Получайте полный прайс-лист на все книги и направляйте заказы по адресу:
г.Киев : 03055 а/я 100, тел./факс (044) 212-1254, 216-3564 e-mail: books@diasoftkiev.ua
stepanb@akcecc.kiev.ua

ПРАЙС-ЛИСТ ИЗДАТЕЛЬСТВА "ДИАСОФТ" ОТДЕЛА "КНИГА-ПОЧТОЙ"

Код	Автор	Название	Цена в грн.	Цена в руб.	Стр.	Формат	Заказ
		Использование ПК в целом					
4756	Морзе Н.В.	Основи інформатики Екзаменаційні білети:запитання т	5		160	84x108/16	
5616	Жалдак, Мор	Інформатика-7. Експеременталний навчальний посібн	9		208	70x100/16	
6700	Клименко, Н	Эффективный самоучитель работы на ПК. 2-е изд.	15,9	79	672	70x100/16	
4043	Нортон П., Г	Работа на персональном компьютере. Самоучитель	19	95	584	84x108/16	
5611	Вебер Ральс	Сборка, конфигурирование, настройка, модернизация	30		544	70x100/16	
5164	Анонимный і	Максимальная безопасность в Linux	39	195	400	84x108/16	
4683	Канер Сэм и	Тестирование программного обеспечения	25	150	544	60x84/16	
5228	Митчелл Ш.	Толковый словарь компьютерных технологий	29	195	720	70x100/16	
4020	Нортон П., Г	Внутренний мир персональных компьютеров, 8е издан	34	169	548	84x108/16	
		Операционные системы					
9	Питрек	Внутренний мир Windows	4		0		
5612	Чуприн А., Т	Эффективный самоучитель в Windows Me	16	79	304	70x100/16	
2759	Бойс Д.	Внутренний мир Windows 98.	25	150	608	84x108\16	
306	Коварт Р.	Windows NT 3.51. Энциклопедия пользователя (+ CD-ROM	5	135	592	60x84/8	
5511	Кессел Пол	Microsoft Windows 2000 Professional. Энциклопедия пользоват	49	245	832	70x100/16	
5003	Вильяме М.	Программирование в Windows 2000. Энциклопедия поль:	75	395	640	84x108/16	
44	Кепли М.	Ответы на актуальные вопросы по OS/2 Warp	5	28	352	60x84/16	
		Linux, Unix					
4653	Скловская С	Red Hat Linux 6.0. Учебник	19	95	401	70x100/16	
5453	Сэри П.	Сервер Red Hat Linux для Windows	24	120	400	70x100/16	
5758	Сэтчэлл Сте	Linux IP Stacks в комментариях (+ CD-ROM)	29	170	288	70x100/16	
3926	Паркер Тим	Linux 5.2. Энциклопедия пользователя + 2 CD-ROM	29	145	688	84x108/16	
5164	Анонимный а	Максимальная безопасность в Linux	39	195	400	84x108/16	
5641	Скловская С	Команды Unux. Справочник.	39	195	688	70x100/16	
4844	Максвелл Ск	Ядро Linux в комментариях (+ CD)	39	215	488	84x108/16	
5717	Шенк Т.	Red Hat Linux для системных администраторов. Энцикло	59	290	672	84x108/16	
936	Хейр К. и др.	Внутренний мир Unix (+ дискета).	40	250	830	84x108/16	
1557	Бурк Р., Хорв	Unix для системных администраторов. Энциклопедия (	80	470	864	60x84/8	
4318	Бурк Робин,Х	UNIX для Internet . Энциклопедия пользователя (+ CD-I	95	480	496	84x108/16	
		Программирование					
5462	Прата С.	Язык программирования С++. Лекции и упражнения	29	14S	656	84x108/16	
4982	Прата Стиве	Язык программирования С. Лекции и упражнения	29	179	432	84x108/16	
4754	Либерти ,Дж	С++ Энциклопедия пользователя с приложением	53	275	584	84x108/16	
5804	Хэзфилд Р.	Искусство программирования на С. Фундаментальные	59	295	736	84x108/16	
3850	Эйткен Пите	Программирование на Visual Basic 6. Этюды профессион	29	165	480	84x108/16	
5063	Гилберт С.	Самоучитель Visual C++ 6 в примерах. (+ CD-ROM)	23	150	496	70x100/16	
5056	Олафсен Ю.	Visual C ++ 6 и MFC. Энциклопедия пользователя. (+ CD-	75	390	720	84x108/16	
3621	Калверт Ч.	Delphi 4. Самоучитель (без приложения)	9	45	192	84x108\16	
5845	Кандзюба С.	Delphi 6. Базы данных и приложения. Лекции и упражне	19	95	576	70x100/16	
5419	Кандзюба С.	Delphi 5. Базы данных и приложения. Лекции и упражне	25		592	70x100/16	
2757	Калверт Ч.	Delphi 4. Энциклопедия пользователя (+ CD-ROM)	29	145	400	84x108\16	
242	Калверт Ч.	Delphi 2. Энциклопедия пользователя (+ CD-ROM).	35	120	736	60x84/8	
93	Конопка Р.	Создание оригинальных компонент в среде Delphi	49		512	60x84/16	
2758	Калверт Ч.	Borland C++ Builder 3. Энциклопедия пользователя (+ CD-	45	225	800	84x108\16	
5799	Клименко	Kylix 1.0. Базы данных и приложения. Лекции и упражне	19	95	288	70x100/16	
1057	Галлагер С.,	Power Builder 6.0. Энциклопедия пользователя (+ CD-ROM	35	180	816	60x84/8	
5571	Саймон Р.	Windows 2000 API. Энциклопедия программиста (+ CD-RO	64	320	1088	70x100/16	
650	Чепмен Д.	Разработка Internet-приложений в Delphi 2 (+ CD-ROM).	9	45	640	60x84/16	
5720	Вайк А.	PHP. Справочник	29	150	448	70x100/16	
5149	Бизли Девид	Язык программирования Python. Справочник	29	150	336	70x100/16	
6721	Хьюгс С., Зм	PHP. Руководство разработчика	31	155	384	70x100/16	
5258	Холден Грег,	Apache Server в комментариях.(+ CD-ROM)	49	269	480	84x108/16	
5165	Вайк Аллен	JavaScript в примерах	19	95	304	70x100/16	
5355	Вайк А., Ван	JavaScript. Энциклопедия пользователя (+ CD-ROM j	55	289	464	84x108\16	
		Графика и анимация, CAD-пакеты.					
5512		Adobe Photoshop 6.0. Полный справочник-путеводитель пс	4	20	32	84x108/16	
2472	Боутон Г. и Д	Внутренний мир Adobe Photoshop 5 (+ CD-ROM)	19	95	496	84x108\16	
5803	Кишик А.Н.	Adobe Photoshop 6.0. Эффективный самоучитель	19	96	304	70x100/16	

5587	Компания Add	Adobe Illustrator 9.0. Учебник (+ CD-ROM)	36	180	368	70x100/16	
169	Боутон Г.	CorelDraw! 6 для экспертов (+ CD-ROM)	10	100	608	60x84/16	
551S	Ковтанюк Ю.	CorelDRAW 10 для дизайнера	29	170	880	70x100/16	
5002	Тёмин Г.	3D Studio MAX 3. Учебник (+ CD-ROM)	19	95	480	70x100/16	
4634	Бордмэн	Внутренний мир 3D Studio MAX 3 : Моделирование, мате	19	95	456	84x108/16	
5802	Тёмин Г.	3D Studio MAX 4. Эффективный самоучитель (+ CD-ROM	29	150	480	70x100/16	
3925	Маэстри Джо	Внутренний мир 3D Studio MAX 2. Том 3: Анимация (+ CC	29	145	408	84x108/16	
2860	Бордмэн Т.,	Внутренний мир 3D Studio MAX 2. Том 2: Моделировани	29	145	368	84x108\16	
5080	Миллер Ф.	Внутренний мир 3D Studio MAX 3: моделирование,матери	29	145	720	84x108/16	
649	Эспиноза Д.	3D Studio MAX..Внутренний мир. Том 2-й (+ CD-ROM)	37		432	84x108/16	
570	Эллиотт С.	Внутренний мир 3D Studio MAX. Том 1 (+ CD-ROM)	60		752	84x108/8	
2471	Эллиотт С.,	Внутренний мир 3D Studio MAX 2. Том 1 (+ CD-ROM).	61		848	84x108/16	
5069	Мильберн Ке	Внутренний мир Flash 4 для дизайнера. (+ CD-ROM)	19	180	448	70x100/16	
5029	Мортиер Р.	Внутренний мир Bryce 4 для дизайнера (+ CD-ROM)	23	150	336	70x100/16	
5262	Мильберн К.	Внутренний мир FLASH 5 для дизайнера + CD-ROM)	29	150	496	70x100/16	
2101	Браун Дейв	Adobe Web-дизайн и публикация. Энциклопедия пользов	39	190	650	60x84/8	
4763	Харрел Виль	Внутрений мир QuarkXPress 4 с приложением.	39	215	424	84x108/16	
5163	Хансен Хан	Разработка сценариев для PageMaker (+ CD-ROM)	29	150	704	60x84/16	
5064	Чуприн А.	AutoCAD 2000. Лекции и упражнения (+ дискета)	29	180	784	70x100/16	
4753	Барчард Бил	Внутренний мир AutoCAD 2000. (+ CD-ROM)	34	169	688	84x108/16	
3924	Барчард Б. и	Внутренний мир AutoCAD 14. Новые возможности (+ CD	49		767	84x108/16	
		Офисные пакеты и другие книги					
4303	Нортон Пите	Microsoft Office 2000.Избранное от Питера Нортона	29	145	552	84x108/16	
5139	Линд Д., Кер	Lotus Notes и Domino R5. Энциклопедия пользователя (+	79	390	656	84x108/16	
5096	Форт С.,Хоуи	Программирование в среде Access 2000. Энциклопедия	75	390	544	84x108/16	
942	Сингх Л.	Oracle 7.3. Пособие разработчика (+ CD-ROM)	42	195	736	84x108/16	
4925	Бэлсон Дон	Внутренний мир Oracle8. Проектирование и настройка.(	75	470	800	84x108/16	
5057	Грин Джо	Oracle 8/8i Server. Энциклопедия пользователя.	75	390	576	84x108/16	
1493	AIS	Oracle 8. Энциклопедия пользователя (+ CD-ROM)	160		864	60x84/8	
5640	Бьелетич Ша	MS SQL Server 2000.	49	245	688	70x100/16	
2010	Мак-Налли Д	Informix. Энциклопедия пользователя	60	300	800	60x84/8	
3947	Спортак М. И	Компьютерные сети. Книга 1: High-Performance Networking.	30	180	432	60x84/8	
3927	Спортак М. И	Компьютерные сети. Книга 2: Networking Essentials. Энцикл	65	250	432	84x108/16	
1295	Грин Д. и др.	Microsoft BackOffice 2. Энциклопедия пользователя (+ CD-	9	45	800	60x84/8	
1032	Оливер Дик,	Популярные Web-броузеры. Энциклопедия пользовате	9	45	460	60x84/8	
5510	Хан Харли	Эффективный самоучитель работы в Internet	16	125	448	60x84/16	
40	Левин Я., Ле	Ответы на актуальные вопросы по Internet	5	28	384	60x84/16	
802	Акоста Н. и Д	Внутренний мир World Wide Web.	9	45	544	84x108/8	
314	Скибб Л., Хе	Оптимизация мультимедиа ПК (+ CD-ROM)	15		352	60x84/16	
4849	Устинова Г.М	Информационные системы менеджмента	45	215	368	60x90/8	

Файл взят с сайта - http://www.natahaus.ru/

где есть ещё множество интересных и редких книг,
программ и прочих вещей.
Данный файл представлен исключительно в
ознакомительных целях.
Уважаемый читатель!
Если вы скопируете его,
Вы должны незамедлительно удалить его
 сразу после ознакомления с содержанием.
Копируя и сохраняя его Вы принимаете на себя всю
ответственность, согласно действующему
международному законодательству .
Все авторские права на данный файл
сохраняются за правообладателем.
Любое коммерческое и иное использование
кроме предварительного ознакомления запрещено.
Публикация данного документа не преследует за
собой никакой коммерческой выгоды. Но такие документы
способствуют быстрейшему профессиональному и
духовному росту читателей и являются рекламой
бумажных изданий таких документов.

Все авторские права сохраняются за правообладателем.
Если Вы являетесь автором данного документа и хотите
дополнить его или изменить, уточнить реквизиты автора
или опубликовать другие документы, пожалуйста,
свяжитесь с нами по e-mail - мы будем рады услышать ваши
пожелания.