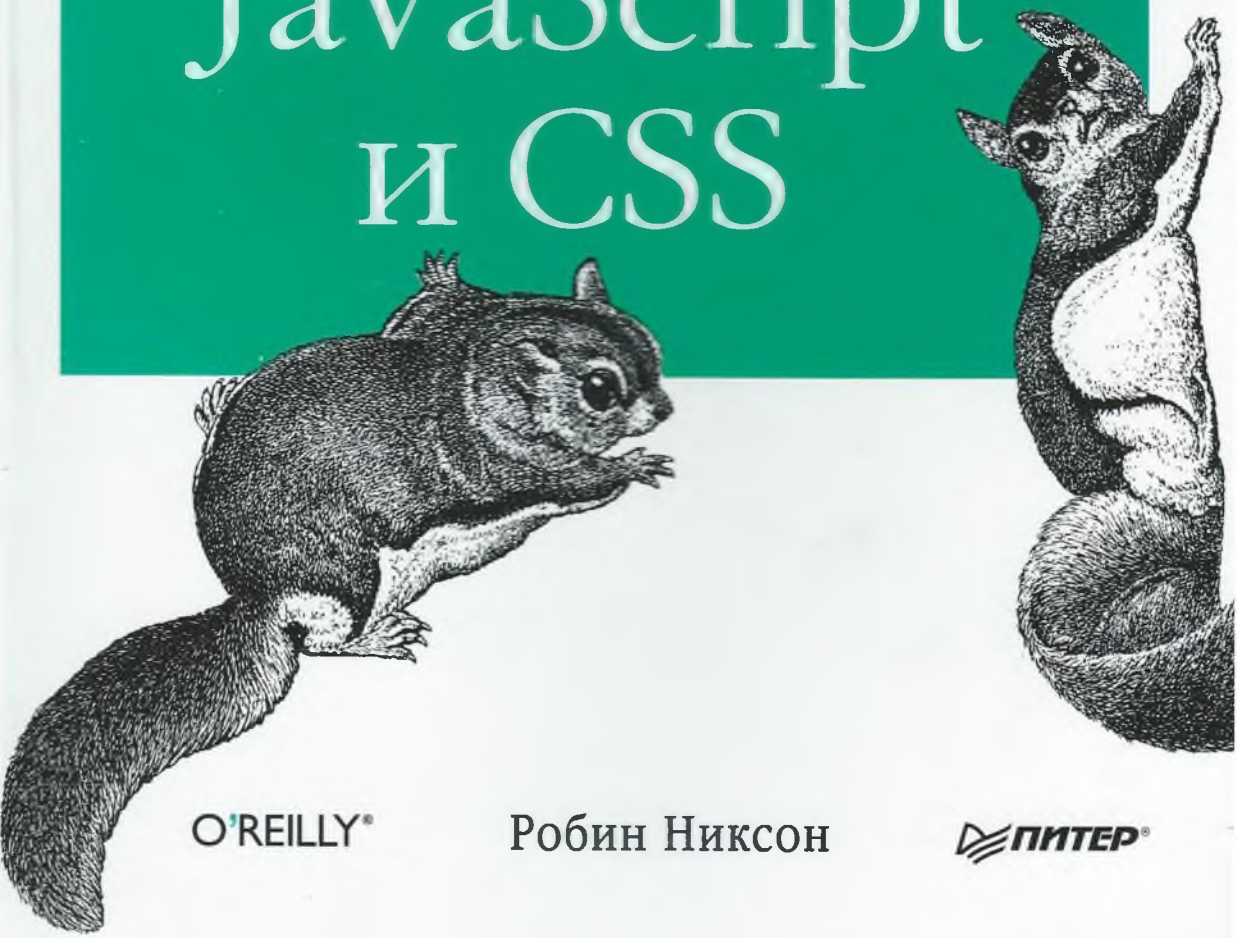
2-Е ИЗДАНИЕ

СОЗДАЕМ ДИНАМИЧЕСКИЕ ВЕБ-САЙТЫ С ПОМОЩЬЮ

# PHP, MySQL JavaScript и CSS



Робин Никсон

O'REILLY®

ПИТЕР®

SECOND EDITION

# Learning PHP, MySQL, JavaScript, and CSS

Robin Nixon

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

ВТОРОЕ ИЗДАНИЕ

# СОЗДАЕМ ДИНАМИЧЕСКИЕ ВЕБ-САЙТЫ С ПОМОЩЬЮ PHP, MySQL JavaScript и CSS

Робин Никсон

ПИТЕР®

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2013

*Никсон Р.*

# Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS

## 2-е издание

*Серия «Бестселлеры O'Reilly»*

Перевел с английского *Н. Вильчинский*


| | |
|---|---|
| Заведующий редакцией | *Д. Виницкий* |
| Ведущий редактор | *Е. Каляева* |
| Научный редактор | *Н. Вильчинский* |
| Литературный редактор | *Е. Каляева* |
| Художник | *Л. Адуевская* |
| Корректоры | *Е. Павлович* |
| Верстка | *А. Барцевич* |

ББК 32.988.02-018 УДК 004.738.5

**Никсон Р.**

Н64 Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS. 2-е изд. — СПб.: Питер, 2013. — 560 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-496-00187-8



Научитесь создавать современные динамические веб-сайты, даже если у вас нет опыта в программировании! Если вы умеете писать статические сайты на HTML, то с помощью этого руководства вы освоите динамическое веб-программирование и изучите современные технологии с открытым кодом: PHP, MySQL, JavaScript и CSS.

В данном руководстве каждая технология рассматривается отдельно и показывается, как их объединить в одно целое, дается представление о самых современных концепциях веб-программирования. С помощью подробно разобранных примеров и контрольных вопросов, приводимых в каждой главе, вы сможете закрепить изученный материал на практике.



12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1449319267 англ.





ISBN 978-5-496-00187-8





ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 04.04.13. Формат 70×100/16. Усл. п. л. 45,150. Тираж 2000. Заказ № 296.
Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ГППО «Псковская областная типография». 180004, Псков, ул. Ротная, 34.

# Краткое содержание

**Предисловие** . . . . . 21

**От издательства** . . . . . 25

**Глава 1.** Введение в динамическое содержимое веб-страницы. . . 26

**Глава 2.** Установка сервера, предназначенного для разработки . . . . . 39

**Глава 3.** Введение в PHP . . . . . 62

**Глава 4.** Выражения и управление процессом выполнения программы в PHP . . . . . 90

**Глава 5.** Функции и объекты PHP . . . . . 118

**Глава 6.** Массивы в PHP . . . . . 145

**Глава 7.** Практикум по программированию на PHP . . . . . 160

**Глава 8.** Введение в MySQL . . . . . 188

**Глава 9.** Освоение MySQL . . . . . 230

**Глава 10.** Доступ к MySQL с использованием PHP . . . . . 254

**Глава 11.** Обработка форм . . . . . 281

**Глава 12.** Cookie, сессии и аутентификация . . . . . 298

**Глава 13.** Изучение JavaScript . . . . . 318

**Глава 14.** Выражения и управление процессом выполнения сценариев в JavaScript . . . . . 338

**Глава 15.** Функции, объекты и массивы JavaScript . . . . . 356

**Глава 16.** Проверка данных и обработка ошибок в JavaScript и PHP . . . . . 374

**Глава 17.** Использование технологии Ajax. . . . . 397

**Глава 18.** Введение в CSS . . . . . 413

**Глава 19.** Расширение CSS с помощью CSS3 . . . . . 452

**Глава 20.** Доступ к CSS из JavaScript. . . . . 477

**Глава 21.** Объединение технологий . . . . . 496

**Приложение А.** Ответы на контрольные вопросы. . . . . 527

**Приложение Б.** Интернет-ресурсы . . . . . 543

**Приложение В.** MySQL's FULLTEXT Stopwords. . . . . 546

**Приложение Г.** Функции MySQL . . . . . 549

# Оглавление

**Предисловие** . . . . . 21
Для кого предназначена эта книга . . . . . 21
Предположения, допущенные в данной книге . . . . . 21
Как устроена книга . . . . . 22
Дополнительная литература . . . . . 22
Соглашения, использованные в данной книге . . . . . 23
Использование примеров кода . . . . . 24
Благодарности . . . . . 24

**От издательства** . . . . . 25

**Глава 1.** Введение в динамическое содержимое веб-страницы . . . 26
HTTP и HTML: основы, заложенные Бернерсом-Ли . . . . . 27
Процедура «запрос — ответ» . . . . . 27
Преимущества использования PHP, MySQL, JavaScript и CSS . . . . . 29
Использование PHP . . . . . 31
Использование MySQL . . . . . 32
Использование JavaScript . . . . . 33
Использование CSS . . . . . 34
Веб-сервер Apache . . . . . 35
Несколько слов о программах с открытым исходным кодом . . . . . 36
А теперь все это, вместе взятое . . . . . 36
Проверьте ваши знания . . . . . 38

**Глава 2.** Установка сервера, предназначенного для разработки . . . . . 39
Что такое WAMP, MAMP и LAMP . . . . . 40
Установка WAMP на систему Windows . . . . . 40
Тестирование установки . . . . . 47
Другие системы WAMP . . . . . 49

Установка MAMP на систему Mac OS X . . . . . 50
Настройка MySQL . . . . . 53
Обеспечение запуска MySQL при загрузке системы . . . . . 54
Проверка установки . . . . . 54
Установка LAMP на Linux . . . . . 56
Работа в удаленном режиме . . . . . 57
Вход в систему . . . . . 57
Использование FTP . . . . . 57
Использование редактора программ . . . . . 59
Использование IDE . . . . . 60
Проверьте ваши знания . . . . . 61

**Глава 3.** Введение в PHP . . . . . 62
Включение PHP в HTML . . . . . 62
Вызов парсера PHP . . . . . 63
Примеры, приводимые в этой книге . . . . . 64
Структура PHP . . . . . 65
Использование комментариев . . . . . 65
Основной синтаксис . . . . . 66
Осмысление переменных . . . . . 67
Операторы . . . . . 71
Присваивание значений переменным . . . . . 74
Многострочные команды . . . . . 77
Типы переменных . . . . . 79
Константы . . . . . 80
Различие между командами echo и print . . . . . 81
Функции . . . . . 82
Область видимости переменной . . . . . 83
Проверьте ваши знания . . . . . 88

**Глава 4.** Выражения и управление процессом выполнения программы в PHP . . . . . 90
Выражения . . . . . 90
Литералы и переменные . . . . . 92
Операторы . . . . . 93
Приоритетность операторов . . . . . 93
Взаимосвязанность операторов . . . . . 95
Операторы отношения . . . . . 96
Условия . . . . . 100
Инструкция if . . . . . 100
Инструкция else . . . . . 102
Инструкция elseif . . . . . 103

Инструкция switch . . . . . 104
Оператор ? . . . . . 107
Организация циклов . . . . . 108
Циклы while . . . . . 109
Циклы do...while . . . . . 110
Циклы for . . . . . 111
Прекращение работы цикла . . . . . 113
Инструкция continue . . . . . 114
Неявное и явное преобразование типов . . . . . 114
Динамическое связывание в PHP . . . . . 115
Динамическое связывание в действии . . . . . 116
Проверьте ваши знания . . . . . 117

**Глава 5.** Функции и объекты PHP . . . . . 118
Функции PHP . . . . . 119
Определение функции . . . . . 120
Возвращение значения . . . . . 121
Возвращение массива . . . . . 123
Передача по ссылке . . . . . 123
Возвращение глобальных переменных . . . . . 125
И еще раз об области видимости переменных . . . . . 125
Включение и запрос файлов . . . . . 126
Инструкция include . . . . . 126
Использование инструкции include_once . . . . . 126
Использование инструкций require и require_once . . . . . 127
Совместимость версий PHP . . . . . 127
Объекты PHP . . . . . 128
Терминология . . . . . 129
Объявление класса . . . . . 130
Создание объекта . . . . . 131
Доступ к объектам . . . . . 131
Конструкторы . . . . . 134
Написание методов . . . . . 135
Объявление свойств . . . . . 136
Объявление констант . . . . . 137
Область видимости свойств и методов в PHP 5 . . . . . 138
Наследование . . . . . 140
Проверьте ваши знания . . . . . 143

**Глава 6.** Массивы в PHP . . . . . 145
Основные подходы к массивам . . . . . 145
Массивы с числовой индексацией . . . . . 145

Ассоциативные массивы . . . . . 147
Присваивание, использующее ключевое слово array . . . . . 148
Цикл foreach...as . . . . . 149
Многомерные массивы . . . . . 151
Использование функций для работы с массивами . . . . . 154
is_array . . . . . 154
count . . . . . 154
sort . . . . . 154
shuffle . . . . . 155
explode . . . . . 155
extract . . . . . 156
compact . . . . . 157
reset . . . . . 158
end . . . . . 158
Проверьте ваши знания . . . . . 159

**Глава 7.** Практикум по программированию на PHP . . . . . 160
Использование функции printf . . . . . 160
Настройка представления данных . . . . . 162
Дополнение строк . . . . . 163
Использование функции sprintf . . . . . 165
Функции даты и времени . . . . . 165
Константы, связанные с датами . . . . . 167
Использование функции checkdate . . . . . 167
Работа с файлами . . . . . 168
Проверка существования файла . . . . . 168
Создание файла . . . . . 168
Чтение из файлов . . . . . 170
Копирование файлов . . . . . 171
Перемещение файла . . . . . 171
Удаление файла . . . . . 172
Обновление файлов . . . . . 172
Блокирование файлов при коллективном доступе . . . . . 173
Чтение всего файла целиком . . . . . 175
Загрузка файлов на веб-сервер . . . . . 175
Системные вызовы . . . . . 181
XHTML . . . . . 182
Преимущества XHTML . . . . . 183
Версии XHTML . . . . . 183
Отличие XHTML от HTML . . . . . 183
Типы документов HTML 4.01 . . . . . 185

Типы документов HTML5 . . . 185
Типы документов XHTML 1.0 . . . 185
Проверка правильности XHTML . . . 186
Проверьте ваши знания . . . 187

**Глава 8.** Введение в MySQL . . . 188
Основные характеристики MySQL . . . 188
Сводка понятий, используемых в базах данных . . . 189
Доступ к MySQL из командной строки . . . 189
Начало работы с интерфейсом командной строки . . . 190
Использование интерфейса командной строки . . . 193
Команды MySQL . . . 195
Типы данных . . . 199
Индексы . . . 208
Создание индекса . . . 209
Создание запросов к базе данных MySQL . . . 214
Объединение таблиц . . . 223
Использование логических операторов . . . 225
Функции MySQL . . . 226
Работа с MySQL через phpMyAdmin . . . 226
Для пользователей Windows . . . 226
Для пользователей Mac OS X . . . 226
Для пользователей Linux . . . 228
Использование phpMyAdmin . . . 228
Проверьте ваши знания . . . 228

**Глава 9.** Освоение MySQL . . . 230
Проектирование базы данных . . . 230
Первичные ключи: ключи к реляционным базам данных . . . 231
Нормализация . . . 232
Первая нормальная форма . . . 233
Вторая нормальная форма . . . 235
Третья нормальная форма . . . 238
Когда не следует проводить нормализацию . . . 240
Отношения . . . 241
«Один к одному» . . . 241
«Один ко многим» . . . 241
«Многие ко многим» . . . 242
Базы данных и анонимность . . . 244
Транзакции . . . 244
Ядра (механизмы хранения) транзакций . . . 245
Использование команды BEGIN . . . 246

Использование команды COMMIT . . . . . 246
Использование команды ROLLBACK . . . . . 246
Использование команды EXPLAIN . . . . . 247
Резервное копирование и восстановление данных . . . . . 248
Использование команды mysqldump . . . . . 249
Создание файла резервной копии . . . . . 250
Восстановление данных из файла резервной копии . . . . . 252
Выгрузка данных в файлы формата CSV . . . . . 252
Планирование резервного копирования . . . . . 253
Проверьте ваши знания . . . . . 253

**Глава 10.** Доступ к MySQL с использованием PHP . . . . . 254
Запросы к базе данных MySQL с помощью PHP . . . . . 254
Процесс . . . . . 254
Создание файла регистрации . . . . . 255
Подключение к MySQL . . . . . 256
Практический пример . . . . . 261
Массив $_POST . . . . . 264
Удаление записи . . . . . 264
Отображение формы . . . . . 265
Запросы к базе данных . . . . . 266
Запуск программы . . . . . 266
Практическая работа с MySQL . . . . . 267
Создание таблицы . . . . . 268
Описание таблицы . . . . . 268
Удаление таблицы . . . . . 269
Добавление данных . . . . . 270
Извлечение данных . . . . . 270
Обновление данных . . . . . 271
Удаление данных . . . . . 272
Использование свойства AUTO_INCREMENT . . . . . 272
Выполнение дополнительных запросов . . . . . 274
Предотвращение внедрения SQL-кода . . . . . 275
Предотвращение внедрения HTML-кода . . . . . 279
Проверьте ваши знания . . . . . 280

**Глава 11.** Обработка форм . . . . . 281
Создание форм . . . . . 281
Извлечение отправленных данных . . . . . 282
register_globals: склонность к использованию устаревших решений . . . . . 284
Значения по умолчанию . . . . . 284

Типы элементов ввода данных . . . . . 286
Обезвреживание введенных данных . . . . . 293
Пример программы . . . . . 294
Проверьте ваши знания . . . . . 297

**Глава 12.** Cookie, сессии и аутентификация . . . . . 298
Использование cookie в PHP . . . . . 298
Установка cookie . . . . . 300
Доступ к cookie . . . . . 301
Удаление cookie . . . . . 301
HTTP-аутентификация . . . . . 301
Сохранение имен пользователей и паролей . . . . . 304
Добавление произвольных данных . . . . . 305
Использование сессий . . . . . 308
Начало сессии . . . . . 309
Завершение сессии . . . . . 312
Безопасность сессии . . . . . 313
Проверьте ваши знания . . . . . 317

**Глава 13.** Изучение JavaScript . . . . . 318
JavaScript и текст HTML . . . . . 318
Использование сценариев в заголовке документа . . . . . 320
Устаревшие и нестандартные браузеры . . . . . 320
Включение файлов JavaScript . . . . . 321
Отладка кода JavaScript . . . . . 322
Использование комментариев . . . . . 324
Точка с запятой . . . . . 324
Переменные . . . . . 325
Строковые переменные . . . . . 325
Числовые переменные . . . . . 326
Массивы . . . . . 326
Операторы . . . . . 327
Арифметические операторы . . . . . 327
Операторы присваивания . . . . . 328
Операторы сравнения . . . . . 328
Логические операторы . . . . . 329
Инкремент и декремент переменной . . . . . 329
Объединение строк . . . . . 329
Управляющие символы . . . . . 329
Типизация переменных . . . . . 330
Функции . . . . . 331
Глобальные переменные . . . . . 331

Локальные переменные . . . . . 332
Объектная модель документа . . . . . 333
Но не все так просто . . . . . 335
Использование DOM . . . . . 336
Проверьте ваши знания . . . . . 337

**Глава 14.** Выражения и управление процессом выполнения сценариев в JavaScript . . . . . 338
Выражения . . . . . 338
Литералы и переменные . . . . . 339
Операторы . . . . . 340
Приоритетность операторов . . . . . 341
Взаимосвязанность . . . . . 341
Операторы отношения . . . . . 342
Инструкция with . . . . . 345
Использование события onerror . . . . . 345
Использование конструкции try...catch . . . . . 346
Условия . . . . . 347
Инструкция if . . . . . 348
Инструкция switch . . . . . 349
Оператор ? . . . . . 350
Циклы . . . . . 351
Циклы while . . . . . 351
Циклы do...while . . . . . 351
Циклы for . . . . . 352
Прекращение работы цикла . . . . . 353
Инструкция continue . . . . . 353
Явное преобразование типов . . . . . 354
Проверьте ваши знания . . . . . 355

**Глава 15.** Функции, объекты и массивы JavaScript . . . . . 356
Функции JavaScript . . . . . 356
Определение функции . . . . . 356
Возвращение значения . . . . . 358
Возвращение массива . . . . . 360
Объекты JavaScript . . . . . 361
Объявление класса . . . . . 361
Создание объекта . . . . . 362
Доступ к объектам . . . . . 363
Ключевое слово prototype . . . . . 363
Массивы в JavaScript . . . . . 365
Числовые массивы . . . . . 366

Ассоциативные массивы . . . . . 367
Многомерные массивы . . . . . 368
Использование методов массивов . . . . . 369
Проверьте ваши знания . . . . . 373

**Глава 16.** Проверка данных и обработка ошибок в JavaScript и PHP . . . . . 374
Проверка данных, введенных пользователем, средствами JavaScript. . . . 374
Документ validate.html (часть первая) . . . . . 375
Документ validate.html (часть вторая) . . . . . 377
Регулярные выражения . . . . . 380
Соответствие, закладываемое в метасимволы . . . . . 380
Нестрогое символьное соответствие . . . . . 381
Группировка с помощью скобок . . . . . 382
Символьный класс . . . . . 383
Более сложные примеры. . . . . 384
Сводная таблица метасимволов . . . . . 387
Общие модификаторы . . . . . 388
Использование регулярных выражений в JavaScript. . . . . 389
Использование регулярных выражений в PHP . . . . . 389
Повторное отображение формы после проверки данных PHP-программой . . . . . 390
Проверьте ваши знания . . . . . 395

**Глава 17.** Использование технологии Ajax. . . . . 397
Что такое Ajax? . . . . . 398
Использование XMLHttpRequest . . . . . 398
Реализация Ajax с помощью POST-запросов . . . . . 400
Свойство readyState . . . . . 402
Серверная половина Ajax-процесса . . . . . 403
Использование GET вместо POST . . . . . 405
Отправка XML-запросов . . . . . 407
Несколько слов о XML. . . . . 409
А зачем вообще использовать XML? . . . . . 410
Использование для Ajax специальной среды . . . . . 411
Проверьте ваши знания . . . . . 412

**Глава 18.** Введение в CSS . . . . . 413
Импортирование таблицы стилей. . . . . 414
Встроенные настройки стиля . . . . . 415
Использование идентификаторов (ID). . . . . 415
Использование классов. . . . . 415

Правила CSS . . . . . 416
Использование точек с запятой . . . . . 416
Множественные задания стиля . . . . . 416
Использование комментариев . . . . . 417
Типы стилей . . . . . 418
Исходные стили . . . . . 418
Пользовательские стили . . . . . 418
Внешние таблицы стилей . . . . . 419
Внутренние стили . . . . . 419
Внедренные стили . . . . . 420
Селекторы CSS . . . . . 420
Селектор типа . . . . . 420
Селектор потомков . . . . . 420
Селектор дочерних элементов . . . . . 421
Селектор смежных элементов . . . . . 422
Селектор элементов, имеющих идентификатор . . . . . 423
Селектор класса . . . . . 424
Селектор атрибутов . . . . . 424
Универсальный селектор . . . . . 425
Групповая селекция . . . . . 426
Каскадность CSS . . . . . 426
Создатель таблиц стилей . . . . . 427
Методы создания таблиц стилей . . . . . 427
Селекторы таблиц стилей . . . . . 428
Разница между  и  . . . . . 430
Измерения . . . . . 431
Шрифты и оформление . . . . . 433
font-family . . . . . 433
font-style . . . . . 434
font-size . . . . . 434
font-weight . . . . . 435
Управление стилями текста . . . . . 435
Оформление . . . . . 436
Разрядка . . . . . 436
Выравнивание . . . . . 437
Преобразование . . . . . 437
Отступы . . . . . 437
Цвета CSS . . . . . 438
Сокращенные цветовые строки . . . . . 438
Градиенты . . . . . 439

Позиционирование элементов . . . . . 440
Абсолютное позиционирование . . . . . 440
Относительное позиционирование . . . . . 441
Фиксированное позиционирование . . . . . 441
Сравнение типов позиционирования . . . . . 441
Псевдоклассы . . . . . 443
Псевдоэлементы . . . . . 445
Сокращенная запись правил . . . . . 445
Модель блока и макет страницы . . . . . 446
Установка полей . . . . . 446
Применение границ . . . . . 448
Настройка отступов . . . . . 449
Содержимое объекта . . . . . 451
Проверьте ваши знания . . . . . 451

**Глава 19.** Расширение CSS с помощью CSS3 . . . . . 452
Селекторы атрибутов . . . . . 452
Соответствующие части строк . . . . . 453
Свойство box-sizing . . . . . 454
Создание фона в CSS3 . . . . . 454
Свойство background-clip . . . . . 455
Свойство background-origin . . . . . 455
Свойство background-size . . . . . 457
Использование нескольких фонов . . . . . 457
Границы CSS3 . . . . . 459
Свойство border-color . . . . . 459
Свойство border-radius . . . . . 460
Прямоугольные тени . . . . . 463
Выход элемента за пределы размеров . . . . . 463
Разметка с использованием нескольких колонок . . . . . 464
Цвета и непрозрачность . . . . . 465
Цвета HSL . . . . . 465
Цвета HSLA . . . . . 466
Цвета RGB . . . . . 466
Цвета RGBA . . . . . 467
Свойство opacity . . . . . 467
Эффекты, применяемые к тексту . . . . . 468
Свойство text-shadow . . . . . 468
Свойство text-overflow . . . . . 468
Свойство word-wrap . . . . . 469

Веб-шрифты . . . . . 469
Веб-шрифты Google . . . . . 470
Трансформации . . . . . 470
Переходы . . . . . 472
Свойства, применяемые к переходам . . . . . 473
Продолжительность перехода . . . . . 473
Задержка перехода . . . . . 473
Задание скорости перехода . . . . . 473
Сокращенный синтаксис . . . . . 474
Проверьте ваши знания . . . . . 476

**Глава 20.** Доступ к CSS из JavaScript . . . . . 477
Еще одно обращение к функции getElementById . . . . . 477
Функция O . . . . . 477
Функция S . . . . . 478
Функция C . . . . . 479
Включение функций . . . . . 480
Обращение к свойствам CSS из JavaScript . . . . . 481
Некоторые общие свойства . . . . . 482
Другие свойства . . . . . 482
Встроенный JavaScript . . . . . 485
Ключевое слово this . . . . . 485
Привязка событий к объектам в сценарии . . . . . 486
Прикрепление к другим событиям . . . . . 486
Добавление новых элементов . . . . . 487
Удаление элементов . . . . . 489
Альтернативы добавлению и удалению элементов . . . . . 489
Использование прерываний . . . . . 490
Использование функции setTimeout . . . . . 490
Отмена тайм-аута . . . . . 491
Использование функции setInterval . . . . . 491
Использование прерываний для анимации . . . . . 493
Проверьте ваши знания . . . . . 495

**Глава 21.** Объединение технологий . . . . . 496
Проектирование сайта социальной сети . . . . . 496
Информация на веб-сайте . . . . . 497
Файл functions.php . . . . . 497
Функции . . . . . 497
Файл header.php . . . . . 499
Файл setup.php . . . . . 501
Файл index.php . . . . . 502

Файл signup.php . . . . . 503
Проверка возможности использования желаемого имени пользователя . . . . . 503
Файл checkuser.php . . . . . 506
Файл login.php . . . . . 507
Файл profile.php . . . . . 509
Добавление текста в поле About Me (Обо мне) . . . . . 510
Добавление изображения профиля . . . . . 510
Обработка изображения . . . . . 511
Отображение текущего профиля . . . . . 511
Файл members.php . . . . . 513
Просмотр профилей пользователей . . . . . 513
Добавление и удаление друзей . . . . . 513
Вывод списка всех участников . . . . . 514
Файл friends.php . . . . . 517
Файл messages.php . . . . . 520
Файл logout.php . . . . . 523
Файл styles.css . . . . . 524

**Приложение А.** Ответы на контрольные вопросы . . . . . 527

**Приложение Б.** Интернет-ресурсы . . . . . 543

**Приложение В.** MySQL's FULLTEXT Stopwords . . . . . 546

**Приложение Г.** Функции MySQL . . . . . 549

*Юлии*

# Предисловие

Сочетание PHP и MySQL является одним из самых удобных подходов к динамическому веб-конструированию, основанному на использовании базы данных. Этот подход удерживает свои позиции перед лицом вызовов, брошенных интегрированными средами разработки, такими как Ruby on Rails, освоение работы с которыми дается значительно труднее. Благодаря открытости исходных кодов (в отличие от конкурирующей технологии Microsoft .NET framework) это технологическое сочетание можно использовать совершенно бесплатно, и поэтому оно приобрело очень большую популярность у веб-разработчиков.

Любой претендующий на результативность разработчик, использующий платформу Unix/Linux или даже Windows/Apache, нуждается в серьезном освоении этих технологий. В то же время большую важность приобретает изучение языка JavaScript, поскольку он обеспечивает браузерную динамическую функциональность, а также посредством Ajax скрытую связь с веб-сервером, позволяющую создавать плавно меняющиеся интерфейсы.

## Для кого предназначена эта книга

Эта книга предназначена для тех, кто хочет изучить способы создания эффективных и динамичных веб-сайтов. К их числу можно отнести веб-мастеров или специалистов по графическому дизайну, которым уже приходилось создавать статические веб-сайты и у которых есть желание вывести свое мастерство на следующий уровень, а также студентов вузов и колледжей, недавних выпускников этих учебных заведений и просто самоучек.

Фактически любой человек, стремящийся изучить основные принципы, заложенные в основу технологии Web 2.0, известной как Ajax, сможет получить весьма обстоятельные сведения обо всех трех основных технологиях: PHP, MySQL, JavaScript и CSS.

## Предположения, допущенные в данной книге

Материал данной книги изложен в предположении, что читатель уже имеет элементарные понятия об HTML и способен как минимум скомпоновать простой

статический веб-сайт. Но при этом не предполагается наличие у читателя каких-либо знаний в области PHP, MySQL, JavaScript и CSS. Тем не менее, если такие знания имеются, изучение книги будет происходить значительно быстрее.

# Как устроена книга

Главы книги расположены в определенном порядке. Сначала идет представление всех основных технологий, рассматриваемых в книге, а затем описывается процесс их установки на сервер, предназначенный для разработки веб-приложений, для того чтобы подготовить читателя к практической работе с примерами.

В следующей части книги преподносятся основы языка программирования PHP, охватывающие основы синтаксиса, массивов, функций и объектно-ориентированного программирования.

Затем, после усвоения основ PHP, можно переходить к введению в систему управления базами данных MySQL, рассмотрение которой начинается со структуры базы данных MySQL и заканчивается составлением сложных запросов.

После этого рассказывается о том, как воспользоваться сочетанием PHP и MySQL, чтобы приступить к созданию собственных динамических веб-страниц путем интегрирования в это сочетание форм и других функциональных возможностей HTML.

В следующих трех главах рассматриваются подробности практических аспектов разработки на PHP и MySQL, включая описание различных полезных функций и способов работы с cookie и сессиями, а также способов поддержания высокого уровня безопасности.

В следующих четырех главах излагаются основы JavaScript, начиная с простых функций и обработки событий и заканчивая доступом к объектной модели документа (DOM), проверкой введенных данных и обработкой ошибок в браузере.

После рассмотрения всех трех основных технологий излагаются способы создания фоновых Ajax-вызовов и превращения веб-сайтов в высокодинамичную среду.

И наконец, вам предстоит освоить еще две главы, изучая все, касающееся использования CSS для стилевого оформления и подбора формата ваших веб-страниц, а затем собрать ранее изученное в полноценный набор PHP-программ, в совокупности создающий работоспособный веб-сайт социальной сети.

По мере изложения материала дается большое количество указаний и советов по выработке хорошего стиля программирования, а также подсказок, которые помогут читателям обнаружить и устранить скрытые ошибки программирования. Также делается много ссылок на веб-сайты, содержащие дополнительные материалы, относящиеся к рассматриваемым темам.

# Дополнительная литература

Приступив к изучению разработки программных продуктов с помощью PHP, MySQL и JavaScript, вы наверняка будете готовы к переводу своего мастерства на новый уровень, если используете следующие книги:

- «Dynamic HTML: The Definitive Reference» (http://oreil.ly/dynamic_html) Денни Гудмана (Danny Goodman), издательство O'Reilly;
- «PHP in a Nutshell» (http://oreil.ly/PHP_nutshell) Пола Хадсона (Paul Hudson), издательство O'Reilly;
- «MySQL in a Nutshell» (http://oreil.ly/MySQL_nutshell) Рассела Дайера (Russell Dyer), издательство O'Reilly;
- «JavaScript: The Definitive Guide» (http://oreil.ly/JS_Definitive) Дэвида Фланагана (David Flanagan), издательство O'Reilly;
- «CSS: The Definitive Guide» (http://oreil.ly/CSS_Definitive) Эрика А. Майера (Eric A. Myer), издательство O'Reilly.

# Соглашения, использованные в данной книге

Здесь приводится список соглашений, принятых в данной книге.

**Шрифт для названий**

Применяется для отображения URL, а также названий папок и выводимой на экран информации.

`Шрифт для команд`

Используется для имен файлов, названий путей, имен переменных и команд. Для примера, путь будет выглядеть так: `/Developer/Applications`.

```
Шрифт с постоянной шириной
```

Применяется для отображения примеров исходного кода и содержимого файлов.

**`Полужирный шрифт с фиксированной шириной`**

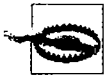
Обозначает текст, который должен быть введен пользователем дословно. Кроме того, данный шрифт иногда используется в примерах для создания логического ударения, например, чтобы выделить важную строку кода в большом примере.

*`Курсивный шрифт с фиксированной шириной`*

Обозначает код, который должен быть заменен подходящим значением (например, *`имя_пользователя`*).

Вам следует обращать особое внимание на специальные врезки, выделенные с помощью следующих рисунков.

Это подсказка, пожелание, заметка общего типа. Содержит полезную прикладную информацию по рассматриваемой теме.



Это предостережение или указание, говорящее о том, что вам необходимо быть внимательным.

# Использование примеров кода

Эта книга предназначена для оказания помощи в выполнении поставленных перед вами задач. Вы можете использовать код, приведенный в ней, в своих программах и документации. Вам не нужно обращаться к нам за разрешением до тех пор, пока вы не станете копировать значительную часть кода. Например, использование при написании программы нескольких фрагментов кода, взятых из данной книги, не требует специального разрешения. А вот продажа и распространение компакт-диска с примерами из книг издательства O'Reilly — требует. Ответы на вопросы, в которых упоминаются материалы этой книги, и цитирование приведенных в ней примеров не требуют разрешения. А включение существенного объема примеров кода, приводимых в данной книге, в документацию по вашему собственному продукту *требует* получения разрешения.

Ссылки на источник приветствуются, но не обязательны. В такие ссылки обычно включаются название книги, имя ее автора, название издательства и номер ISBN. Например: «Learning PHP, MySQL, JavaScript and CSS», второе издание, автор Робин Никсон (Robin Nixon). Copyright 2012 Robin Nixon, 978-1-4493-1926-7».

При любых сомнениях относительно превышения разрешенного объема использования примеров кода, приведенных в данной книге, можете свободно обращаться к нам по адресу permissions@oreilly.com.

# Благодарности

Огромное спасибо моему редактору Энди Ораму, у которого всегда находились дельные мысли о том, как лучше объяснить сложные вопросы, Рэйчел Хэд за отличную работу по приведению в порядок моей рукописи, Айрис Фебрес и Рэйчел Стили, которые тщательно курировали создание книги, Роберту Романо за прекрасные иллюстрации в обоих изданиях, Карен Монтгомери за великолепного летающего сахарного поссума на обложке книги, Дэвида Футато за понятный и легко читаемый внутренний дизайн, и всем другим преданным своему делу работникам издательства O'Reilly, упорно трудившимся над этой книгой. Без них она никогда не была бы написана. Хотелось бы также поблагодарить моих технических рецензентов, работавших над первым изданием, Дерека Дехарта, Кристофера Дорна, Томислава Дуганджика, Беска Моргана, Гарри Никсона, Алана Солиса и Демиана Тернера, а также Альберта Виаша за его неоценимый вклад и советы по новому для этого издания разделу, посвященному CSS. И наконец, спасибо читателям первого издания книги, заметившим опечатки, всем, кто помог появиться этому исправленному, обновленному и улучшенному руководству по веб-разработке.

# От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты vinitski@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

# 1 Введение в динамическое содержимое веб-страницы

Всемирная паутина — это непрерывно развивающаяся сеть, ушедшая далеко вперед от своей концепции ранних 1990-х, когда ее создание было обусловлено решением вполне конкретных задач. Высокотехнологичные эксперименты в ЦЕРНе (Европейском центре физики высоких энергий, известном в наши дни в качестве обладателя большого адронного коллайдера) выдавали невероятно большой объем данных, который был слишком велик для распространения среди участвующих в экспериментах ученых, разбросанных по всему миру.

К тому времени Интернет уже существовал и к нему было подключено несколько сотен тысяч компьютеров, поэтому Тим Бернерс-Ли (Tim Berners-Lee) (специалист ЦЕРНа) придумал способ навигации между ними с использованием среды гиперссылок — так называемого протокола передачи гиперссылок (Hyper Text Transfer Protocol (HTTP)). Он также создал специальный язык разметки, названный языком гипертекстовой разметки (Hyper Text Markup Language (HTML)). Для того чтобы собрать все это воедино, он создал первые веб-браузер и веб-сервер.

Теперь мы воспринимаем эти инструменты как должное, но в то время эта концепция носила революционный характер. До этого основной объем соединений приходился на пользователей домашних модемов, дозванивавшихся и подключавшихся к электронным доскам объявлений, которые базировались на отдельном компьютере и позволяли общаться и обмениваться данными только с другими пользователями этой службы. Следовательно, для эффективного электронного общения с коллегами и друзьями нужно было становиться участником многих электронных досок объявлений (bulletin board systems).

Однако Бернерс-Ли изменил все это одним махом, и к середине 1990-х годов уже существовали три основных конкурирующих друг с другом графических веб-браузера, пользовавшихся вниманием 5 млн пользователей. Но вскоре стало очевидно, что кое-что было упущено. Конечно, текстовые и графические страницы, имеющие гиперссылки для перехода на другие страницы, были блестящей концепцией, но результаты не отражали текущий потенциал компьютеров и Интернета по удовлетворению насущных потребностей пользователей в динамическом изменении

контекста. Всемирная паутина оставляла весьма невыразительное впечатление, даже при использовании прокрутки текста и анимированных GIF-картинок.

Корзины покупателей, поисковые машины и социальные сети внесли существенные коррективы в порядок использования Всемирной паутины. В этой главе будет дан краткий обзор различных компонентов, формирующих ее облик, и программного обеспечения, способствующего обогащению и оживлению наших впечатлений от ее использования.

Пришло время использовать ряд акронимов. Прежде чем делать это, я постарался дать им четкое объяснение. Но, если сразу не удастся понять, какое именно понятие они замещают или что означают, переживать не стоит, поскольку все подробности прояснятся по мере чтения материала.

# HTTP и HTML: основы, заложенные Бернерсом-Ли

HTTP является стандартом взаимодействия, регулирующим порядок направления запросов и получения ответов — процесса, происходящего между браузером, запущенным на компьютере конечного пользователя, и веб-сервером. Задача сервера состоит в том, чтобы принять запрос от клиента и попытаться дать на него содержательный ответ, обычно передавая ему запрошенную веб-страницу. Именно поэтому и используется термин *сервер* (обслуживающий). Партнером, взаимодействующим с сервером, является *клиент*, поэтому данное понятие применяется как к веб-браузеру, так и к компьютеру, на котором он работает.

Между клиентом и сервером может располагаться ряд других устройств, например маршрутизаторы, модули доступа, шлюзы и т. д. Они выполняют различные задачи по обеспечению безошибочного перемещения запросов и ответов между клиентом и сервером. Как правило, для отправки этой информации ими используется Интернет.

Обычно веб-сервер может обрабатывать сразу несколько подключений, а при отсутствии связи с клиентом находится в режиме ожидания входящего запроса на подключение. При поступлении подобного запроса сервер подтверждает его получение отправкой ответа.

## Процедура «запрос — ответ»

В наиболее общем виде процесс «запрос — ответ» состоит из просьбы веб-браузера к веб-серверу отправить ему веб-страницу и выполнения веб-браузером этой просьбы. После этого браузер занимается отображением страницы (рис. 1.1).

При этом соблюдается данная пошаговая последовательность.

1. Вы вводите в адресную строку браузера http://server.com.
2. Ваш браузер ищет IP-адрес, соответствующий доменному имени server.com.
3. Браузер посылает запрос на главную страницу server.com.

4. Запрос проходит по сети Интернет и поступает на веб-сервер server.com.
5. Веб-сервер, получивший запрос, ищет веб-страницу на своем жестком диске.
6. Сервер извлекает веб-страницу и отправляет ее по обратному маршруту в адрес браузера.
7. Браузер отображает веб-страницу.

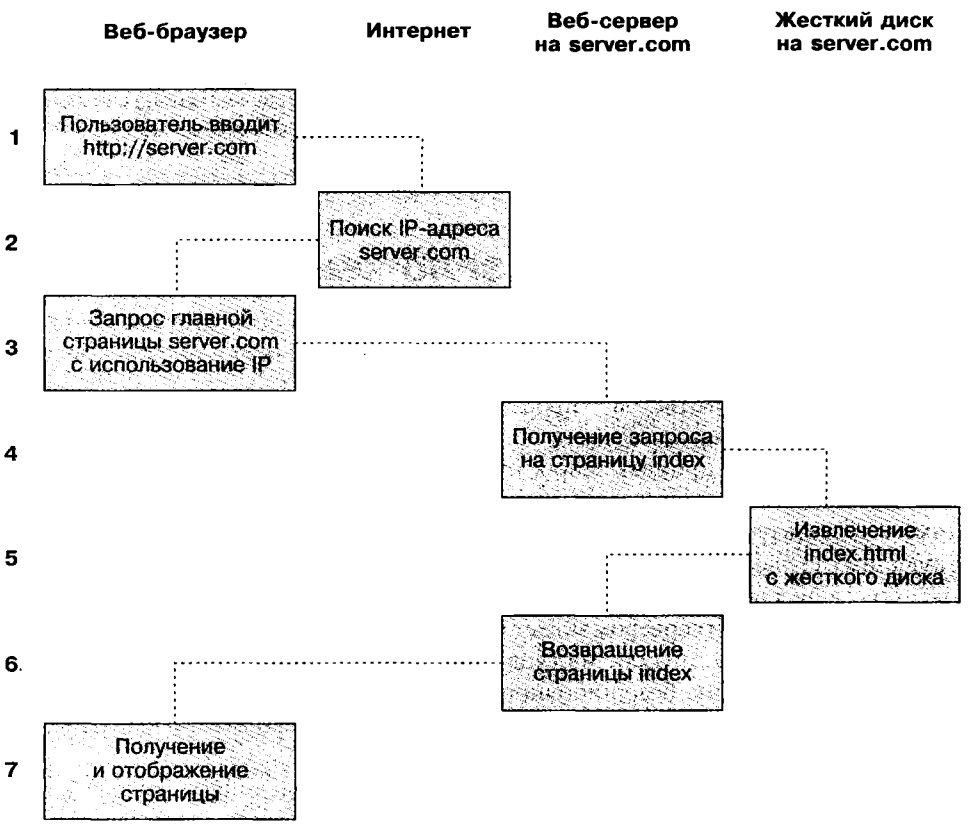



**Рис. 1.1.** Основная последовательность процесса «запрос — ответ» между клиентом и сервером

При передаче типовой веб-страницы этот процесс осуществляется для каждого имеющегося на ней объекта: элемента графики, встроенного видео- или Flash-ролика и даже шаблона CSS.

Обратите внимание на то, что на шаге 2 браузер ищет IP-адрес, принадлежащий доменному имени server.com. У каждой машины, подключенной к Интернету, включая и ваш компьютер, есть свой IP-адрес. Но, как правило, доступ к веб-серверам осуществляется по именам, таким как google.com. Вам, должно быть, известно, что браузер обращается к вспомогательной интернет-службе, так называемой службе доменных имен (Domain Name Service (DNS)), для того чтобы найти связанный с сервером IP-адрес, а затем воспользоваться им для связи с компьютером.

При передаче динамических веб-страниц процедура состоит из несколько большего количества действий, поскольку к ней могут привлекаться как PHP, так и MySQL (рис. 1.2).

1. Вы вводите в адресную строку браузера http://server.com.
2. Ваш браузер ищет IP-адрес, соответствующий доменному имени server.com.
3. Браузер посылает запрос на главную страницу server.com.
4. Запрос проходит по сети Интернет и поступает на веб-сервер server.com.
5. Веб-сервер, получивший запрос, ищет веб-страницу на своем жестком диске.
6. Теперь, когда главная страница размещена в его памяти, веб-сервер замечает, что она представлена файлом, включающим в себя PHP-сценарии, и передает страницу интерпретатору PHP.
7. Интерпретатор PHP выполняет PHP-код.
8. Некоторые фрагменты кода PHP содержат MySQL-инструкции, которые интерпретатор PHP, в свою очередь, передает процессору базы данных MySQL.
9. База данных MySQL возвращает результаты выполнения инструкции интерпретатору PHP.
10. Интерпретатор PHP возвращает веб-серверу результаты выполнения кода PHP, а также результаты, полученные от базы данных MySQL.
11. Веб-сервер возвращает страницу выдавшему запрос клиенту, который отображает эту страницу на экране.

Конечно, ознакомиться с этим процессом и узнать о совместной работе трех элементов не помешает, но на практике эти подробности не понадобятся, поскольку все происходит в автоматическом режиме.

В каждом из примеров возвращенные браузеру HTML-страницы могут содержать также код JavaScript, интерпретируемый локально на машине клиента. Этот код может инициировать еще один запрос, точно так же запрос может быть инициирован встроенными объектами, например изображениями.

# Преимущества использования PHP, MySQL, JavaScript и CSS

В начале этой главы был представлен мир технологии Web 1.0, но рывок к созданию технологии Web 1.1, вместе с которой были разработаны такие браузерные расширения, как Java, JavaScript, JScript (несколько иной вариант JavaScript от корпорации Microsoft) и ActiveX, не заставил себя долго ждать. На серверной стороне прогресс был обеспечен за счет общего шлюзового интерфейса (Common Gateway Interface (CGI)), использования таких языков сценариев, как Perl (альтернатива языку PHP), и выполнения сценариев на стороне сервера (динамическая вставка содержимого одного файла или выходных данных системного вызова в другой файл).

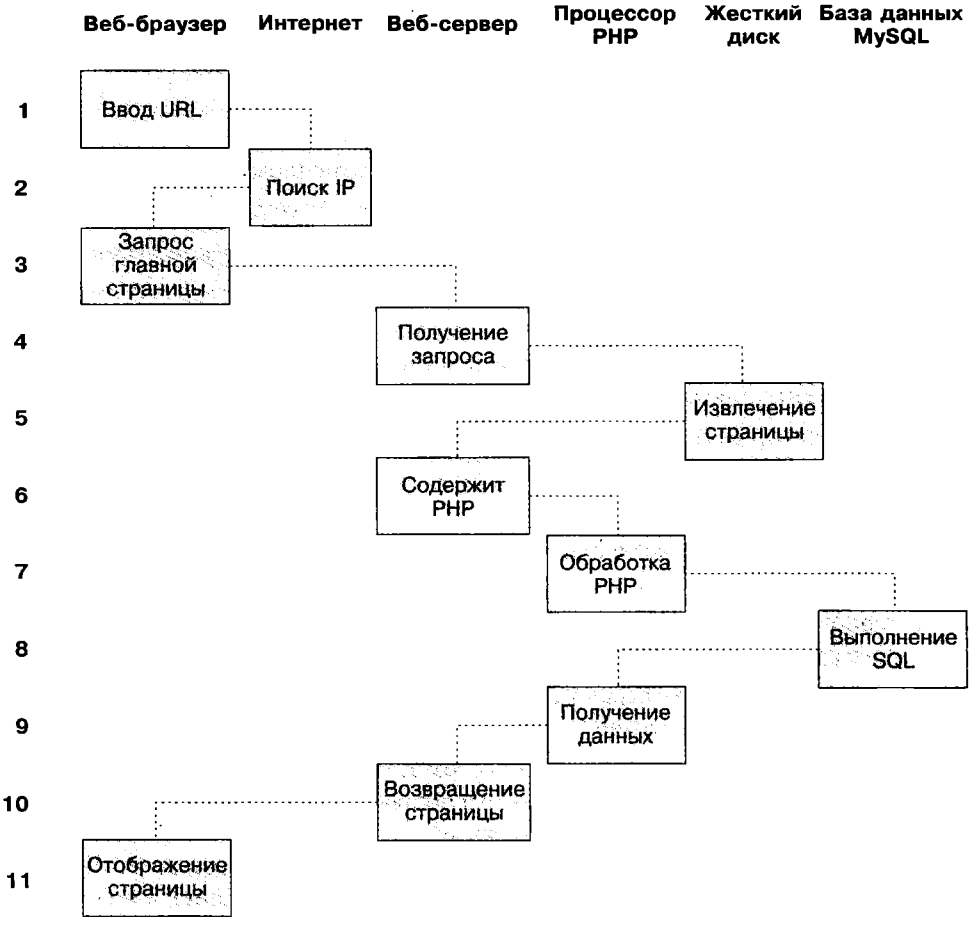



**Рис. 1.2.** Динамическая последовательность процесса «запрос — ответ», выполняемого клиентом и сервером

Когда ситуация окончательно прояснилась, на передовых позициях остались три основные технологии. Несмотря на то что язык сценариев Perl силами своих стойких приверженцев сохранил популярность, простота PHP и допустимость использования в нем встроенных ссылок на программу базы данных MySQL обеспечили этому языку более чем двойное превосходство по количеству пользователей. A JavaScript, ставший важнейшей составной частью уравнения, используемого для динамического манипулирования каскадными таблицами стилей (Cascading Style Sheets (CSS)), в настоящее время берет на себя наиболее трудоемкие задачи осуществления Ajax-процесса на стороне клиента. Благодаря технологии Ajax (которая рассматривается в разделе «Использование JavaScript») веб-страницы обрабатывают данные и отправляют запросы веб-серверу в фоновом режиме, не оповещая пользователя о происходящем.

Несомненно, своеобразный симбиоз PHP и MySQL способствует их продвижению, но что привлекает к ним разработчиков в первую очередь? На это следует дать простой

ответ: та легкость, с которой эти технологии можно использовать для быстрого создания на веб-сайтах динамических элементов. MySQL является быстродействующей и мощной, но при этом простой в использовании системой базы данных, предлагающей веб-сайту практически все необходимое для поиска и обработки данных, предназначаемых для браузеров. Когда PHP для хранения и извлечения этих данных выступает в союзе с MySQL, вы получаете основные составляющие, необходимые для разработки сайтов социальных сетей и для перехода к технологии Web 2.0.

И когда вы также соедините вместе JavaScript и CSS, у вас появится рецепт для создания высокодинамичных и интерактивных веб-сайтов.

## Использование PHP

Использование PHP существенно упрощает встраивание средств, придающих веб-страницам динамические свойства. Когда страницам присваивается расширение PHP, у них появляется прямой доступ к языку сценариев. Разработчику нужно лишь написать код, похожий на этот:

```
<?php
echo "Hello World. Today is ".date("l").". ":
?>

How are you?
```

Открывающий тег <?php дает веб-серверу разрешение на интерпретацию всего последующего кода вплоть до команды ?>. Все, что находится за пределами этой конструкции, отправляется клиенту в виде простого HTML. Поэтому текст «How are you?» просто выводится в браузер. А внутри PHP-тегов встроенная функция date отображает текущий день недели, соответствующий системному времени сервера.

В итоге на выходе их этих двух частей получается примерно следующее:

```
Hello World. Today is Wednesday. How are you?
```

PHP — довольно гибкий язык, и некоторые разработчики предпочитают помещать PHP-конструкцию непосредственно рядом с кодом PHP, как в этом примере:

```
Hello World. Today is <?php echo date("l"); ?>. How are you?
```

Существуют также другие способы форматирования и вывода информации, которые будут рассмотрены в главе, посвященной PHP. Важно усвоить то, что, используя PHP, веб-разработчики получают язык сценариев, который хотя и не обладает быстротой кода, скомпилированного на C или ему подобных языках, но все же работает невероятно быстро и к тому же очень хорошо вписывается в код HTML.

Если вы собираетесь набирать встречающиеся в этой книге примеры на PHP, чтобы работать параллельно с моим повествованием, не забывайте предварять их тегом <?php, а в конце ставить тег ?>, для того чтобы обеспечить их обработку интерпретатором PHP. Для упрощения этой задачи можно заранее подготовить файл example.php, содержащий эти теги.

Используя PHP, вы получаете средство управления своим веб-сервером с неограниченными возможностями. Если понадобится на лету внести изменения в HTML, обработать данные кредитной карты, добавить сведения о пользователе в базу данных или извлечь информацию из стороннего веб-сайта, все это можно будет сделать из тех же самых PHP-файлов, в которых находится и сам код HTML.

## Использование MySQL

Разумеется, без средств отслеживания тех изменений, которые пользователь вносит по мере использования вашего веб-сайта, нельзя в полной мере говорить о возможностях динамического изменения выходного кода HTML. На заре создания Всемирной паутины многие сайты использовали для хранения таких данных, как имена пользователей и пароли, неструктурированные текстовые файлы. Но такой подход мог вызвать ряд проблем, если файл не был надежно заблокирован от повреждений, возникающих при одновременном доступе к нему множества пользователей. К тому же неструктурированный файл мог разрастаться до таких размеров, что с ним непросто было работать, не говоря уже о трудностях, связанных с попытками объединения файлов и осуществления в них сложных поисковых операций за какое-нибудь мало-мальски приемлемое время.

Именно в таких случаях приобретает большое значение использование реляционных баз данных со структурированной системой запросов. И MySQL, будучи совершенно бесплатной и установленной на огромном количестве веб-серверов Интернета системой, оказывается как нельзя кстати. Она представляет собой надежную и исключительно быстродействующую систему управления базами данных, использующую команды, похожие на простые английские слова.

Высшим уровнем структуры MySQL является база данных, внутри которой можно иметь одну или несколько таблиц, содержащих ваши данные. Предположим, например, что вы работаете над таблицей под названием **users** (пользователи), внутри которой были созданы графы для фамилий — **surname**, имен — **firstname** и адресов электронной почты — **email**, и теперь нужно добавить еще одного пользователя. Одна из команд, которую можно использовать для этого, выглядит следующим образом:

```
INSERT INTO users VALUES('Smith', 'John', 'jsmith@mysite.com');
```

Разумеется, как упоминалось ранее, для создания базы данных и таблицы и настройки всех нужных полей понадобится выдать и другие команды, но используемая здесь команда `INSERT` демонстрирует простоту добавления в базу данных новой информации. Команда `INSERT` является примером структурированного языка запросов (Structured Query Language (SQL)), разработанного в начале 1970-х годов и напоминающего один из старейших языков программирования — COBOL. Тем не менее он хорошо подходит для запросов к базе данных, что и предопределило его использование в течение столь длительного времени.

Так же просто выполняется и поиск данных. Предположим, что имеется адрес электронной почты пользователя и нужно найти имя его владельца. Для этого можно воспользоваться следующим запросом MySQL:

```
SELECT surname,firstname FROM users WHERE email='jsmith@mysite.com';
```

После чего MySQL вернет Smith, John и любые другие пары имен, которые могут быть связаны в базе данных с адресом электронной почты.

Нетрудно предположить, что возможности MySQL простираются значительно дальше выполнения простых команд вставки и выбора INSERT и SELECT. Например, можно объединить несколько таблиц в соответствии с множеством различных критериев, запросить результаты, выбрав порядок их выдачи из множества вариантов, найти частичные совпадения, если известна только часть искомой строки, вернуть только конкретно заданное количество результатов и сделать многое другое.

При использовании PHP все эти вызовы можно направлять непосредственно к MySQL, без необходимости запуска самой программы MySQL или использования ее интерфейса командной строки. Это значит, что для того, чтобы докопаться до нужного вам элемента данных, вы можете сохранять результаты в массивах для их обработки и осуществления множества поисковых операций, каждая из которых зависит от результатов, возвращенных предыдущими операциями.

Далее будет показано, что для придания еще большей мощности прямо в MySQL встроен ряд дополнительных функций, которые можно вызвать для наиболее часто встречающихся операций и повышения скорости обработки данных.

## Использование JavaScript

Самая старая из основных технологий, рассматриваемых в данной книге, JavaScript, была создана с целью получения доступа из сценариев ко всем элементам HTML-документа. Иными словами, она предоставляет средства для динамического взаимодействия с пользователем, например для проверки приемлемости адресов электронной почты в формах ввода данных, отображения подсказок наподобие «Вы действительно подразумевали именно это?» и т. д. (хотя с точки зрения безопасности, которая всегда должна реализовываться на веб-сервере, на эту технологию положиться нельзя).

В сочетании с CSS (смотрите следующий раздел) JavaScript закладывает основу мощности динамических веб-страниц, которые изменяются буквально на глазах, в отличие от технологии, при которой сервер возвращает новую страницу.

Тем не менее с использованием JavaScript могут возникнуть осложнения, обусловленные некоторыми существенными различиями в способах реализации этого языка, выбранных разными разработчиками браузеров. В основном эти различия возникают, когда некоторые производители пытаются придать своим браузерам дополнительные функциональные возможности, не обращая внимания на совместимость с продуктами своих конкурентов.

К счастью, разработчики в большинстве своем уже взялись за ум и осознали необходимость полной совместимости своих продуктов, для того чтобы разработчикам веб-продуктов не приходилось создавать код с множеством исключений. Но остаются миллионы экземпляров устаревших браузеров, которыми будут пользоваться на протяжении еще многих лет. Тем не менее и для них существуют решения проблем несовместимости, и позже в этой книге будут рассмотрены технологии, позволяющие без каких-либо опасений проигнорировать существующие различия.

А сейчас давайте взглянем на то, как можно воспользоваться обычным JavaScript, воспринимаемым всеми браузерами:

```
<script type="text/javascript">
  document.write("Hello World. Today is " + Date() );
</script>
```

Этот фрагмент кода предписывает веб-браузеру интерпретировать все, что находится внутри тегов script, в качестве кода JavaScript, который браузер затем интерпретирует путем записи в текущий документ текста «Hello World. Today is», а также даты, полученной за счет использования принадлежащей JavaScript функции Date. В результате получится нечто подобное следующему:

```
Hello World. Today is Sun Jan 01 2012 14:14:00
```

Стоит взять на заметку: если не требуется указывать конкретную версию JavaScript, то, как правило, можно опустить type="text/javascript" и использовать для начала интерпретации JavaScript тег <script>.

Ранее было упомянуто, что изначально JavaScript разрабатывался для того, чтобы получить возможность динамического управления различными элементами, находящимися внутри HTML-документа, и это его предназначение по-прежнему является основным. Но все чаще JavaScript используется для реализации технологии Ajax. Это понятие используется для обозначения процессов доступа к веб-серверу в фоновом режиме. (Сначала оно означало «асинхронный JavaScript и XML» — Asynchronous JavaScript and XML, но сейчас это определение несколько устарело.)

Ajax является основным процессом, лежащим в основе технологии, известной как Web 2.0 (этот термин популяризирован Тимом О'Рейли (Tim O'Reilly), основателем и исполнительным директором издательства, в котором эта книга вышла на английском языке), при использовании которой веб-страницы стали напоминать автономные программы, поскольку их уже не нужно загружать целиком. Вместо этого в быстром вызове Ajax может быть задействован отдельный элемент веб-страницы, например, может быть изменена ваша фотография на сайте социальной сети или заменена кнопка, на которой нужно щелкнуть, отвечая на вопрос. Полностью эта тема будет рассмотрена в главе 17.

## Использование CSS

После появления третьего стандарта (CSS3) CSS предлагает уровень динамической интерактивности, которая прежде поддерживалась только с помощью JavaScript. Например, вы можете не только придать стиль любому элементу HTML, чтобы изменить его размеры, цвета, границы, интервалы, но теперь, используя всего лишь несколько строк CSS, вы можете добавить своим веб-страницам анимированные переходы и преобразования.

Применение CSS может просто заключаться во вставке правил между тегами <style> и </style>, расположенными в заголовке веб-страницы:

```
<style>
  p
  {
     text-align:justify;
     font-family:Helvetica;
  }
</style>
```

Эти правила будут изменять выравнивание по умолчанию тега <p>, чтобы содержащиеся в нем абзацы были полностью выровнены и для них использовался шрифт Helvetica.

В главе 18 будет показано, что существует множество различных способов задания правил CSS и их также можно включать непосредственно в теги или сохранять во внешнем файле, предназначенном для отдельной загрузки. Такая гибкость позволяет не только проводить точную настройку стиля HTML. Вы также увидите, как с ее помощью можно, например, предоставить встроенную функцию hover для анимирования объектов при проходе над ними указателя мыши. Кроме того, вы научитесь получать доступ ко всем свойствам CSS-элемента из JavaScript и из HTML.

# Веб-сервер Apache

В дополнение к PHP, MySQL, JavaScript и CSS в динамической веб-технологии фигурирует и пятый герой — веб-сервер. В нашей книге предполагается, что это веб-сервер Apache. Мы уже немного касались того, что делает веб-сервер в процессе обмена информацией между клиентом и сервером по протоколу HTTP, но на самом деле негласно он выполняет куда более масштабную работу.

Например, Apache обслуживает не только HTML-файлы — он работает с широким спектром файлов, начиная с файлов изображений и Flash-роликов и заканчивая аудиофайлами формата MP3, файлами RSS-потоков (Really Simple Syndication — простое распространение по подписке) и т. д. Каждый элемент, найденный на HTML-странице веб-клиентом, также запрашивается у сервера, который затем и осуществляет обслуживание.

Но эти объекты не должны быть статическими файлами, такими как изображения GIF-формата. Все они могут быть сгенерированы программами, такими как сценарии PHP. И это действительно возможно: PHP способен даже создавать для вас изображения и другие файлы, либо на лету, либо заранее, в расчете на последующее обслуживание. Для этого обычно имеются модули, либо предварительно скомпилированные в Apache или PHP, либо вызываемые во время выполнения программы. Одним из таких модулей является библиотека GD (сокращение от Graphics Draw — «рисование графики»), которую PHP использует для создания и обработки графических элементов.

Apache поддерживает также обширный арсенал собственных модулей. В дополнение к модулям PHP наиболее важными для вас как для веб-программиста будут модули, занимающиеся обеспечением безопасности. В качестве других примеров могут послужить модуль Rewrite, позволяющий веб-серверу обрабатывать широкий

диапазон типов URL-адресов и перезаписывать их в соответствии с его внутренними требованиями, и модуль Proxy, который можно использовать для обслуживания часто запрашиваемых страниц из кэша, для того чтобы снизить нагрузку на сервер.

Далее в этой книге будет показано практическое применение этих модулей для улучшения свойств, предоставляемых рассматриваемыми нами основными технологиями.

## Несколько слов о программах с открытым исходным кодом

Часто спорят, обусловлена или нет популярность этих технологий тем, что они представлены программами с открытым исходным кодом, но PHP, MySQL и Apache *действительно* являются наиболее востребованными инструментами в своих категориях.

Тем не менее следует отметить, что их принадлежность к продуктам с открытым кодом означает, что они были разработаны в сообществе команд программистов, которые придавали им свойства в соответствии со своими желаниями и потребностями и хранили исходный код доступным для всеобщего просмотра и изменения. Ошибки и бреши в системе безопасности могли предотвращаться еще до их проявления.

Есть и еще одно преимущество: все эти программы могут использоваться бесплатно. Если вы наращиваете пропускную способность своего веб-сайта и привлекаете к его обслуживанию дополнительные серверы, не нужно задумываться о приобретении дополнительных лицензий. Также не нужно пересматривать свой бюджет перед тем, как принять решение об обновлении системы и установке самых последних версий этих продуктов.

## А теперь все это, вместе взятое

Истинная красота PHP, MySQL, JavaScript и CSS проявляется в том замечательном способе, благодаря которому они совместно работают над производством динамического веб-контента: PHP занят основной работой на веб-сервере, MySQL управляет данными, а комбинация CSS и JavaScript заботится о представлении веб-страницы. JavaScript может также взаимодействовать с вашим PHP-кодом на веб-сервере, когда ему нужно что-нибудь обновить (как на сервере, так и на веб-странице).

Неплохо бы теперь подвести краткий итог всему, что изложено в данной главе, и, не используя программный код, рассмотреть процесс, сочетающий в себе все три технологии в повседневно использующейся многими веб-сайтами функции Ajax: проверке в процессе регистрации новой учетной записи, не используется ли на сайте выбранное имя кем-нибудь другим. Хорошим примером подобного использования технологий может послужить почтовый сервер Gmail (рис. 1.3).

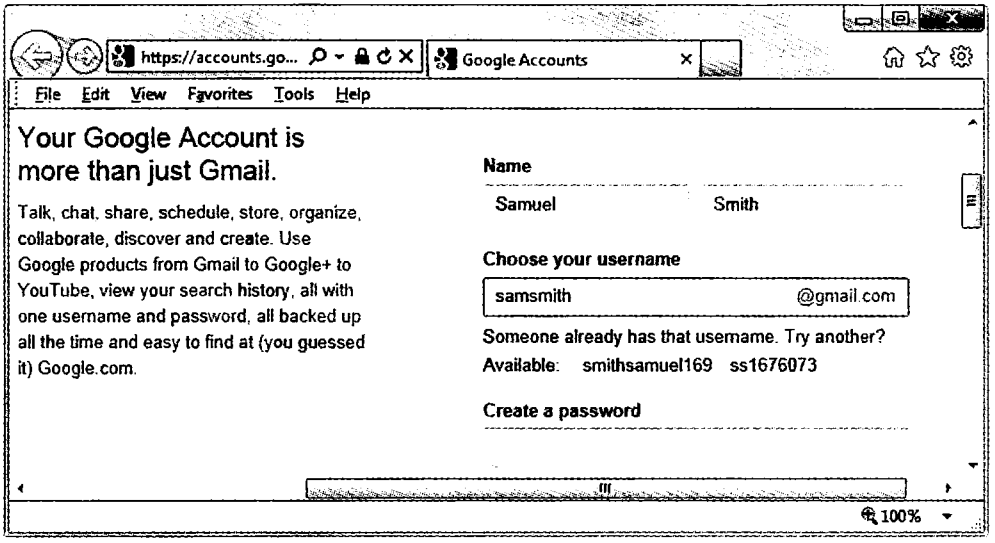



**Рис. 1.3.** Gmail использует технологию Ajax для проверки допустимости пользовательских имен

Этот Ajax-процесс состоит примерно из следующих шагов.

1. Сервер выдает код HTML для создания веб-формы, запрашивающей необходимые данные: имя пользователя, настоящее имя, настоящую фамилию и адрес электронной почты.
2. Одновременно с этим сервер вкладывает в HTML код JavaScript, позволяющий отслеживать содержимое поля ввода имени пользователя и проверять два обстоятельства: введен ли в это поле какой-нибудь текст и был ли фокус ввода перемещен из этого поля по щелчку пользователя на другом поле ввода.
3. Как только будет введен текст и фокус ввода перемещен на другой элемент формы, код JavaScript в фоновом режиме передает введенное имя пользователя PHP-сценарию на веб-сервере и ждет ответной реакции.
4. Веб-сервер осуществляет поиск имени пользователя и возвращает коду JavaScript ответ, в котором сообщает, было ли уже задействовано такое же имя.
5. Затем JavaScript размещает под полем ввода имени пользователя индикатор приемлемости имени пользователя, возможно, в виде зеленой галочки или красного крестика, сопровождая его текстом.
6. Если пользователь ввел неприемлемое имя, но все же пытается отправить форму, код JavaScript прерывает отправку и повторно обращает внимание пользователя (возможно, выводя более крупный графический индикатор и/или открывая окно предупреждения) на необходимость выбора другого имени.
7. Усовершенствованная версия этого процесса может даже изучить имя, запрошенное пользователем, и предложить альтернативное доступное на данный момент имя.

Все это для удобства пользователя и целостности восприятия им всего происходящего делается без привлечения его внимания в фоновом режиме. Без использования Ajax на сервер будет отправлена вся форма, затем он вернет код HTML с подсветкой тех полей, в которых были допущены ошибки. Можно, конечно, сделать и так, но обработка поля на лету будет выглядеть намного интереснее и приятнее.

Технология Ajax может использоваться для решения куда более широкого круга задач, чем простой контроль и обработка вводимой информации. Далее в этой книге будет рассмотрено много дополнительных приемов, реализуемых с применением Ajax.

В этой главе вашему вниманию было представлено довольно полное введение в основные технологии применения PHP, MySQL, JavaScript и CSS (а также Apache) и рассмотрен порядок их совместной работы. В главе 2 будут рассмотрены способы установки вашего собственного сервера, предназначенного для веб-разработок, на котором можно будет освоить на практике весь изучаемый материал. Но сначала ответьте на следующие вопросы.

# Проверьте ваши знания

1. Какие четыре компонента необходимы для создания полностью динамических веб-сайтов?
2. Что означает аббревиатура HTML?
3. Почему в названии MySQL присутствуют буквы SQL?
4. И PHP и JavaScript являются языками программирования, генерирующими динамическое содержимое веб-страниц. В чем состоит их главное различие и почему вы будете использовать оба этих языка?
5. Что означает аббревиатура CSS?
6. Если вам удастся обнаружить ошибку в одном из инструментальных средств с открытым кодом (что случается довольно редко), то как, по-вашему, можно получить исправленную версию?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 1».

# 2 Установка сервера, предназначенного для разработки

Если у вас есть желание разрабатывать интернет-приложения, но нет собственного сервера для их разработки, то прежде, чем протестировать каждую созданную модификацию приложения, вам придется загружать ее на сервер, находящийся где-нибудь в Интернете.

Даже при наличии высокоскоростного широкополосного подключения это обстоятельство может существенно замедлить разработку. А на локальном компьютере тестирование может быть не сложнее обновления программы (зачастую запускается простым щелчком на значке) с последующим нажатием кнопки Refresh (Обновить) в браузере.

Еще одно преимущество разработочного сервера заключается в том, что при написании и тестировании программ не нужно волноваться о смущающих разработчика ошибках или проблемах безопасности, однако при размещении приложения на публичном веб-сайте следует знать о том, что люди могут увидеть, или о том, что они могут сделать с вашим приложением. Лучше решить все проблемы, пока вы работаете дома или в небольшом офисе, который, вероятнее всего, защищен межсетевыми экранами (брандмауэрами) и другими средствами обеспечения безопасности.

Получив в свое распоряжение разработочный сервер, вы удивитесь, как раньше могли обходиться без него, а также обрадуетесь легкости его настройки. Нужно лишь пройти все шаги, изложенные в следующих разделах, и выполнить соответствующие указания для обычных персональных компьютеров, Mac- или Linux-систем.

В этой главе будет рассмотрена только серверная сторона сетевого взаимодействия, о которой шла речь в главе 1. Но для тестирования результатов вашей работы, особенно потом, когда мы приступим к использованию JavaScript и CSS, понадобится также копия каждого основного веб-браузера, работающего под управлением удобной для вас системы. Список браузеров должен состоять по крайней мере из Internet Explorer, Mozilla Firefox, Opera, Safari и Google Chrome. Если вы хотите убедиться, что ваши приложения также хорошо выглядят на мобильных устройствах, постарайтесь использовать для тестирования телефоны и планшеты под управлением Apple iOS и Google Android.

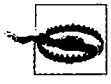
# Что такое WAMP, MAMP и LAMP

WAMP, MAMP и LAMP — это сокращения от «Windows, Apache, MySQL и PHP», «Mac, Apache, MySQL и PHP» и «Linux, Apache, MySQL и PHP» соответственно. Данными сокращениями описываются полноценные функциональные установки, используемые для разработки динамических веб-страниц.

Системы WAMP, MAMP и LAMP поставляются в форме пакетов, связывающих упакованные программы таким образом, чтобы их не нужно было устанавливать и настраивать по отдельности. Это означает, что нужно просто загрузить и установить одну программу и следовать простым подсказкам, чтобы подготовить разработочный сервер и запустить его в кратчайшие сроки и с минимальными усилиями.

В процессе установки будут созданы исходные настройки. Конфигурация безопасности при такой установке не будет столь же строгой, как на технологическом веб-сервере, поскольку она оптимизирована для использования на локальной машине. Поэтому не следует пользоваться такими настройками при установке технологического сервера.

Однако для разработки и тестирования веб-сайтов и приложений подобная установка подойдет как нельзя лучше.



Если для создания своей системы разработки вы решили не использовать W/L/MAMP, следует учесть, что загрузка и самостоятельная взаимоувязка составных частей займет очень много времени и может отнять большое количество сил на исследования для создания полноценной конфигурации всей системы. Но если все компоненты у вас уже установлены и согласованы друг с другом, они смогут работать с примерами, приводимыми в этой книге.

# Установка WAMP на систему Windows

Существует несколько доступных WAMP-серверов, каждый из которых предлагает свою немного отличающуюся от других конфигурацию. Наверное, самый лучший из них — Zend Server CE (где CE означает Community Edition — «издание сообщества»), поскольку он распространяется бесплатно самими разработчиками PHP. Его можно загрузить с веб-сайта http://tinyurl.com/zendce (рис. 2.1).

Я рекомендую вам всегда загружать последний стабильный выпуск (в данном примере это 5.6.0 SP1 для Windows), который будет показан на веб-странице в разделе загрузок первым. На странице должен быть изображен установщик, подходящий для вашего компьютера: Linux, Windows или OS X. Перед загрузкой вам будет предложено войти на сайт. Можно щелкнуть на ссылке, чтобы получить файл без входа на сайт или регистрации, однако в таком случае вам не будут приходить по электронной почте сообщения о выходе обновлений и другие новости.

Возможно, когда данное издание выйдет в свет, некоторые виды экранов и настройки, рассмотренные далее, могут измениться. Если это произойдет, руководствуйтесь здравым смыслом, чтобы выполнить приведенную последовательность действий аналогичным образом.

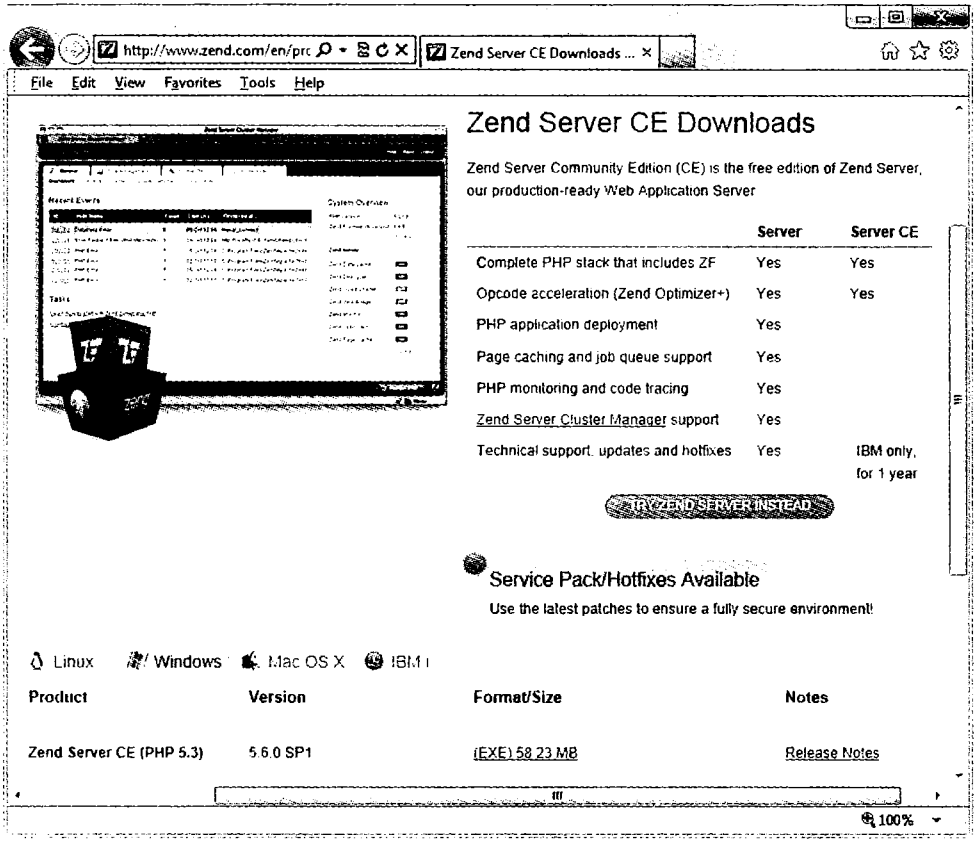



**Рис. 2.1.** Zend Server CE для Windows можно загрузить с веб-сайта Zend

После загрузки запустите установщик, чтобы появилось окно, показанное на рис. 2.2.

Щелкните на кнопке Next (Далее) и примите лицензионное соглашение, чтобы переместиться на экран типа установки (Setup Type) (рис. 2.3). Установите переключатель в положение Custom (По выбору пользователя), чтобы установить еще и сервер MySQL.

Когда появится окно установки по выбору пользователя (Custom Setup), прокрутите вниз до конца список вариантов установки и убедитесь, что установлены флажки phpMyAdmin и MySQL Server (рис. 2.4). Затем щелкните на кнопке Next (Далее).

На следующем экране (рис. 2.5), даже если у вас уже установлен веб-сервер IIS, я рекомендую выбрать установку веб-сервера Apache, поскольку примеры в этой книге рассчитаны на его применение. После выполнения необходимых настроек щелкните на кнопке Next (Далее).

Согласитесь с предлагаемым по умолчанию значением 80 для порта веб-сервера и со значением 10081 для порта интерфейса сервера Zend (рис. 2.6) и нажмите кнопку Next (Далее).

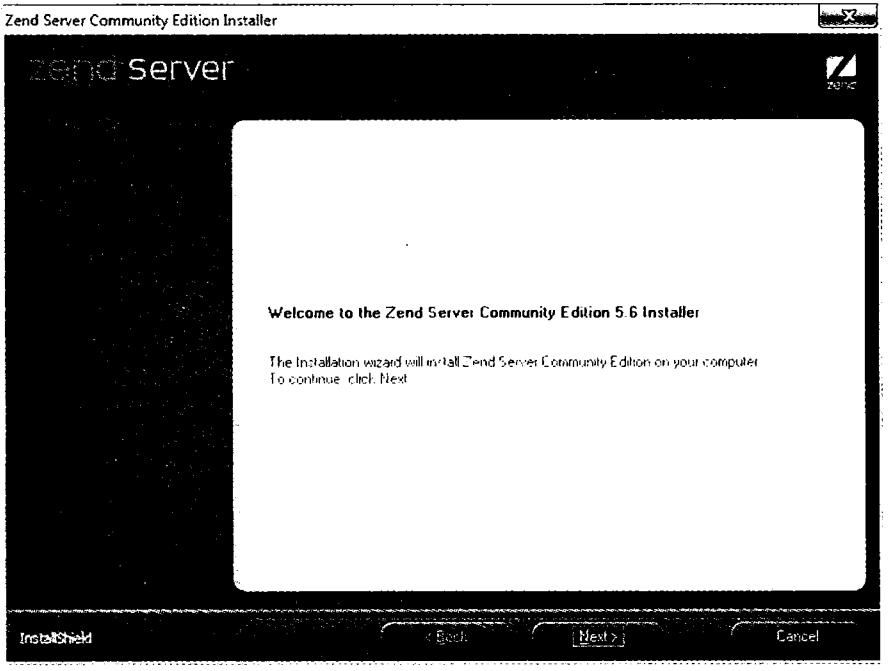



**Рис. 2.2.** Основное окно установщика Zend Server CE

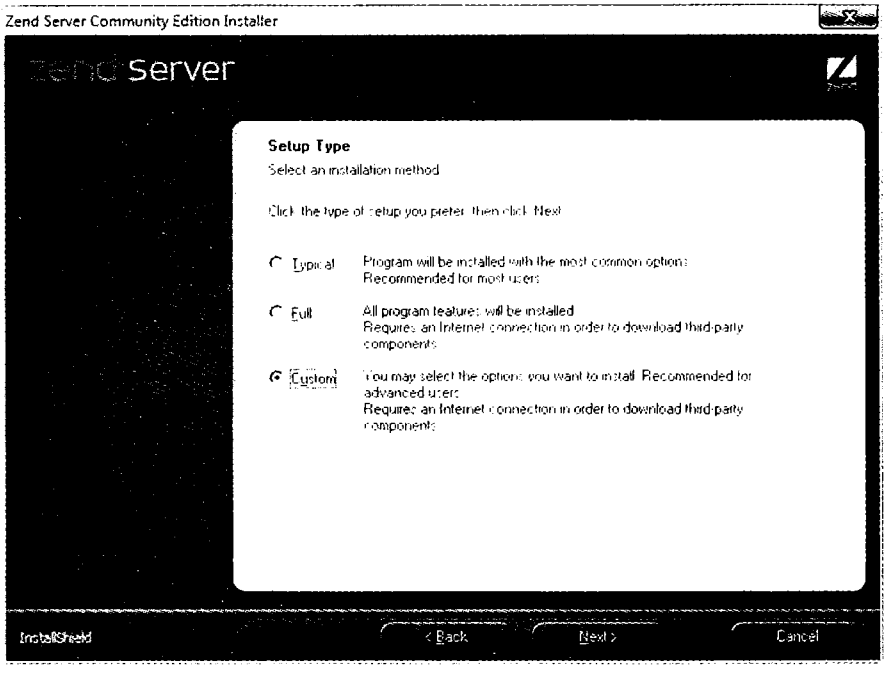



**Рис. 2.3.** Выберите режим установки Custom (По выбору пользователя)

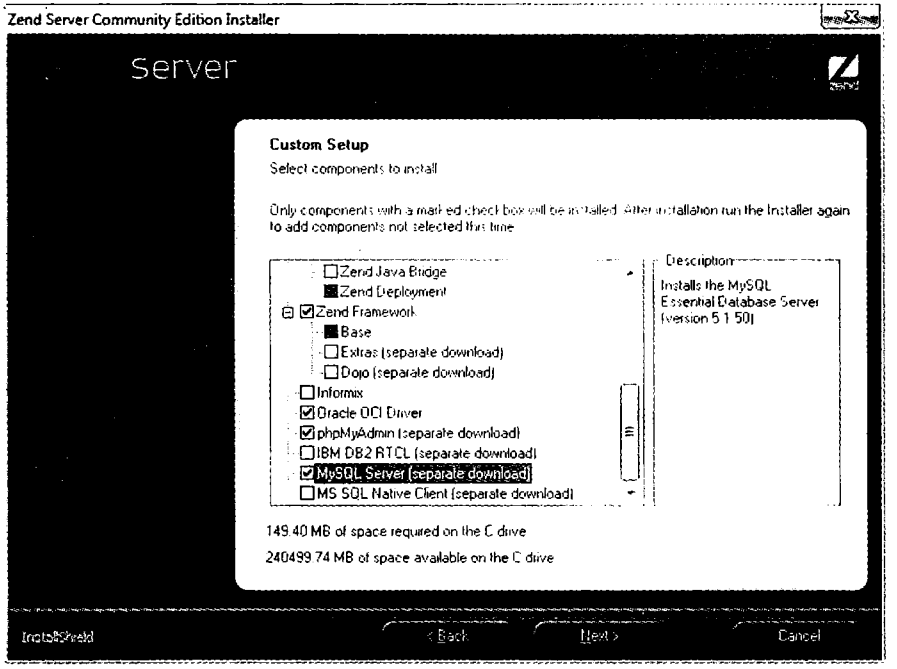



**Рис. 2.4.** Установите флажки для phpMyAdmin и MySQL Server

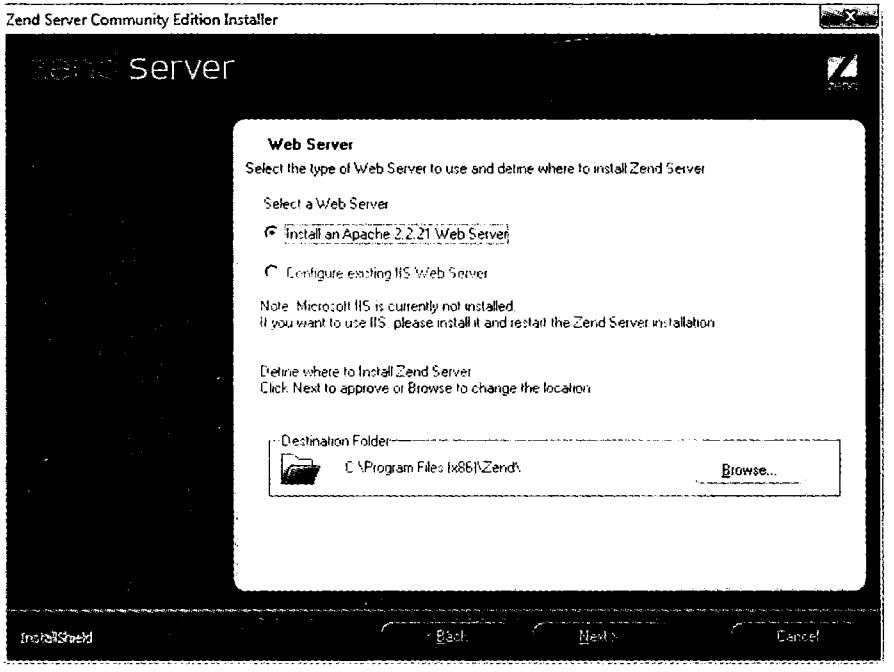



**Рис. 2.5.** Установка веб-сервера Apache

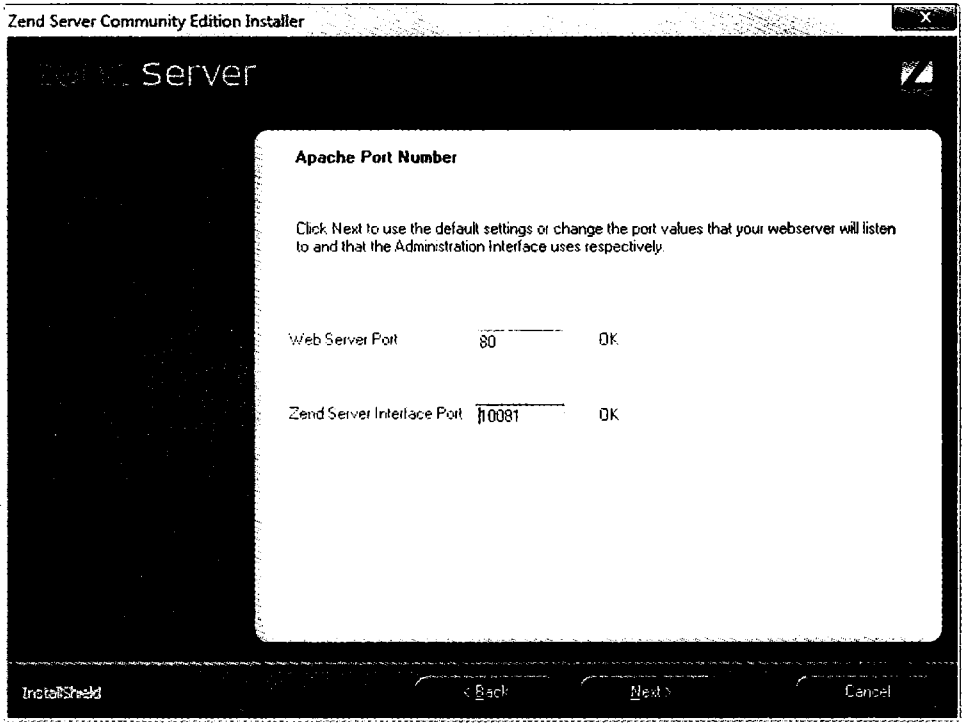



**Рис. 2.6.** Согласие с предлагаемыми по умолчанию портами

После назначения портов вы попадете на экран, показанный на рис. 2.7, где для начала установки нужно щелкнуть на кнопке Install (Установить).

В ходе установки могут быть загружены некоторые дополнительные файлы, поэтому на установку программ может уйти несколько минут. Когда все будет готово, появится уведомление о том, что после щелчка на кнопке Finish (Завершить) можно будет приступить к использованию программного обеспечения. Как только это будет сделано, откроется используемый вами по умолчанию браузер с загруженной в него страницей, которая показана на рис. 2.8. На ней необходимо установить флажок, чтобы согласиться с условиями.

Теперь вы готовы к установке пароля (рис. 2.9). Выберите такой пароль, который легко запомнить, и щелкните на кнопке Next (Далее) для перехода к экрану, показанному на рис. 2.10. Нажмите кнопку Finish (Завершить).

Если какой-либо из предлагаемых портов утверждает, что он уже занят (обычно это бывает при наличии другого запущенного веб-сервера), и вы не можете использовать значения по умолчанию, попробуйте установить значение 8080 (или 8000) для порта веб-сервера и значение 10082 для порта интерфейса сервера Zend. Не забудьте воспользоваться этими значениями позже, при доступе как к веб-страницам, так и к серверу Zend. Например, вместо веб-страницы http://localhost/index.htm, нужно воспользоваться адресом http://localhost:8080/index.htm.

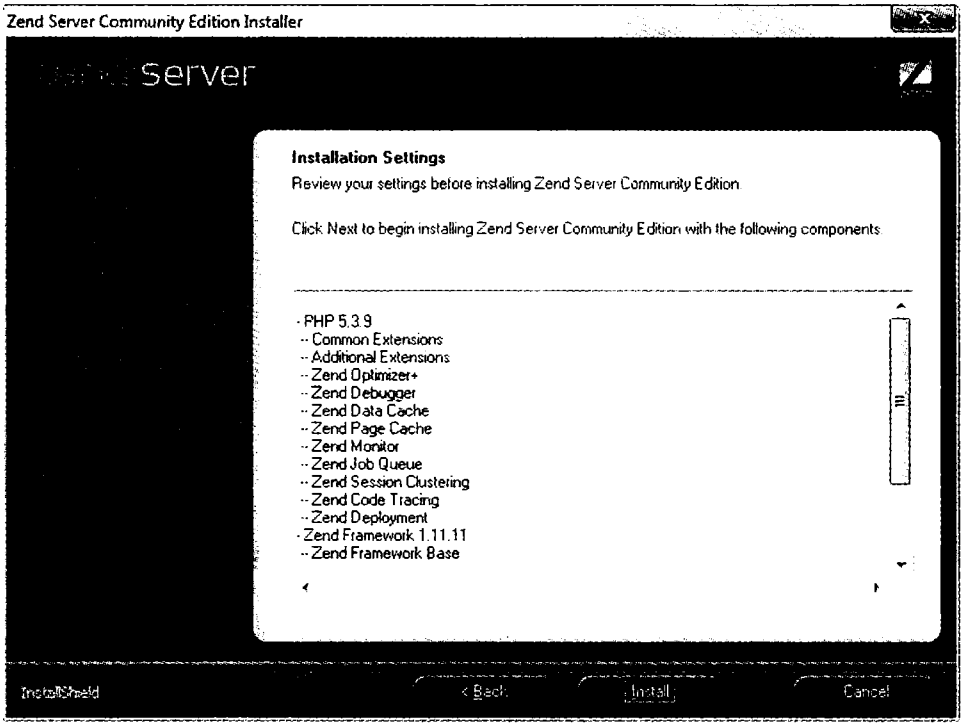



**Рис. 2.7.** Щелкните на кнопке Install (Установить) для начала установки

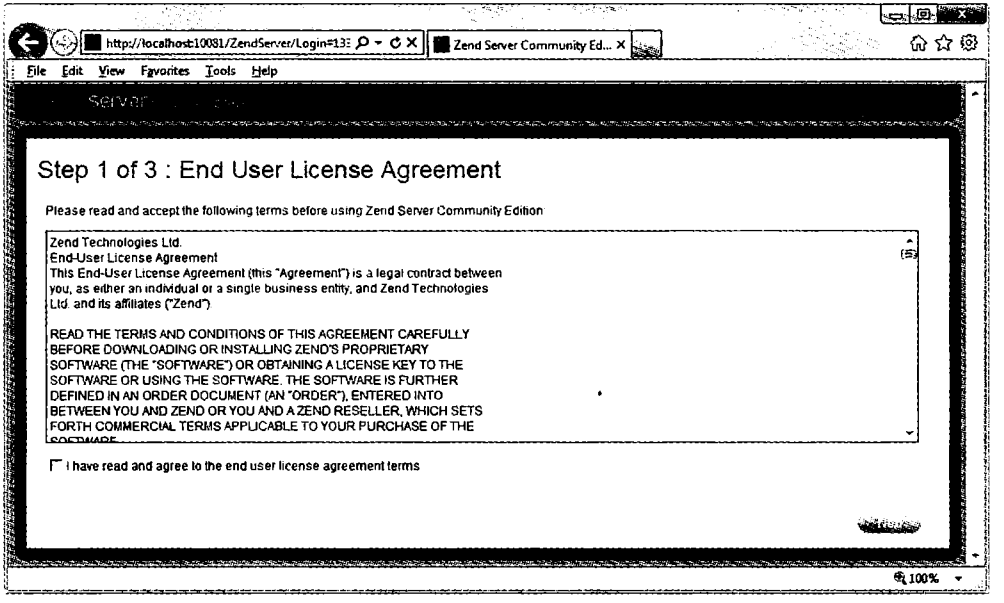



**Рис. 2.8.** Чтобы воспользоваться Zend Server CE, нужно согласиться с условиями лицензии

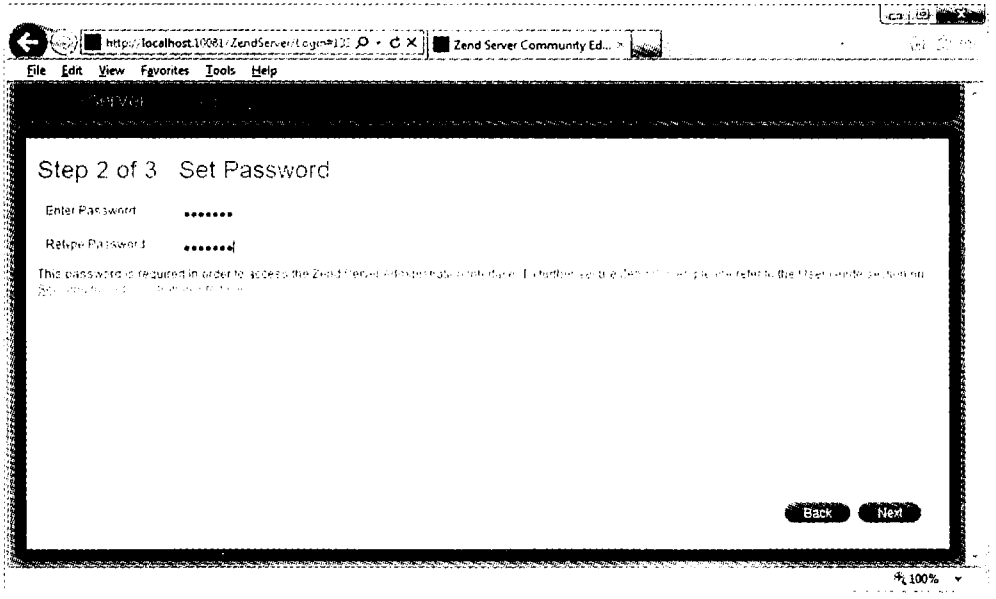



**Рис. 2.9.** Выберите пароль и введите его дважды

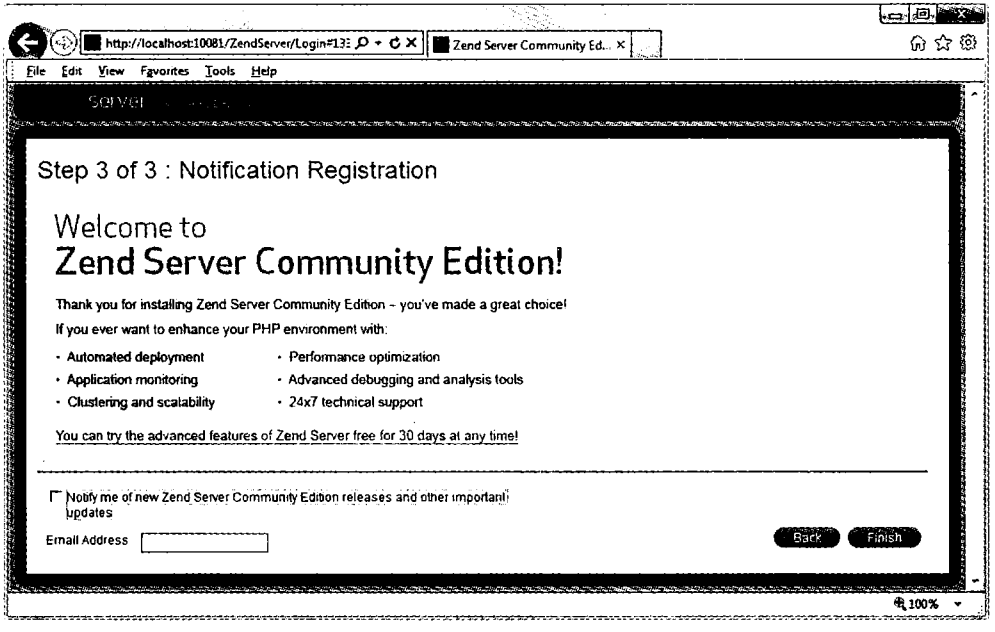



**Рис. 2.10.** Для завершения установки щелкните на кнопке Finish (Завершить)

В результате браузер отобразит экран Dashboard (Инструментальная панель), который позволит вам управлять сервером (рис. 2.11).

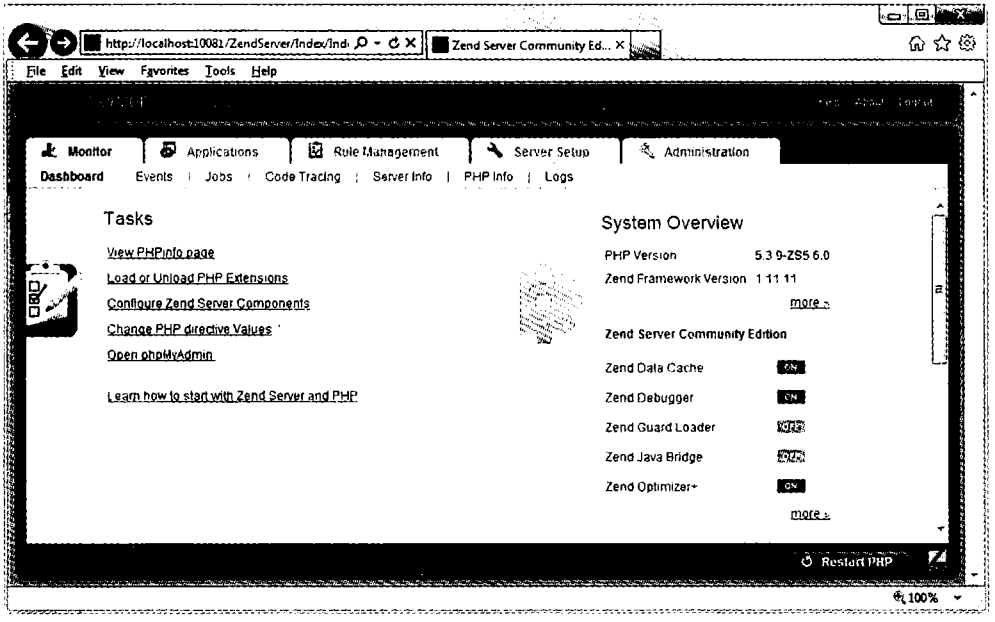



**Рис. 2.11.** Экран управления Zend Server CE

Вы можете вернуться на этот экран в любое время, для чего нужно ввести в адресную строку вашего браузера адрес http://localhost:10081 (или, если для порта интерфейса сервера Zend было указано значение, отличное от 10081, вы можете зайти на этот экран, используя заданное значение после двоеточия).

## Тестирование установки

На данном этапе нужно проверить, что все работает должным образом. Для этого следует попытаться отобразить исходную веб-страницу, которая была сохранена в корневой папке сервера (рис. 2.12). Введите любой из следующих URL-адресов в адресную строку браузера:

```
http://localhost
http://127.0.0.1
```

Слово `localhost` используется в URL-адресах для указания локального компьютера, который также будет отвечать на IP-адрес 127.0.0.1, поэтому исходный источник документов вашего веб-сервера можно вызывать любым из этих методов.

Если в процессе установки был выбран серверный порт, отличный от 80 (например 8080), то после любого из предыдущих URL-адресов нужно поставить двоеточие и значение порта. Например: http://localhost:8080. То же самое придется сделать для всех примеров в данной книге. Например, вместо URL http://localhost/example.php, следует ввести http://localhost:8080/example.php (или то значение, которое вы выбрали).

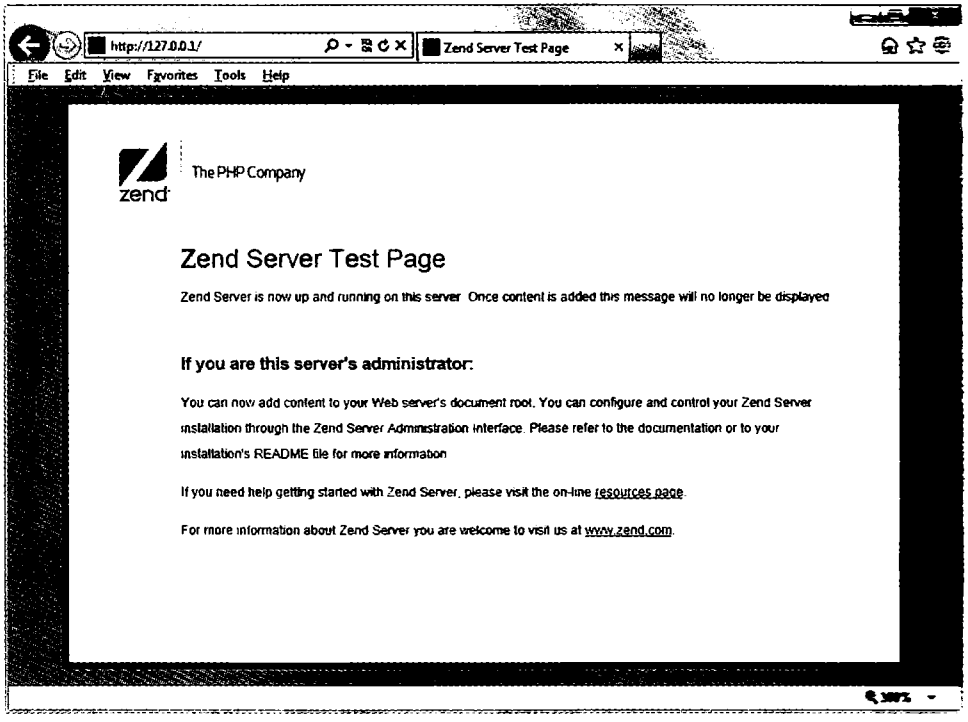



**Рис. 2.12.** Так по умолчанию должна выглядеть главная страница

Исходным источником документов является каталог, в котором содержатся главные веб-документы домена. Именно он вводится, когда в браузере набирается базовый URL без пути, например http://yahoo.com или на локальном сервере http://localhost.

По умолчанию Zend Server CE использует для этого каталога одно из следующих мест (первое из них относится к 32-разрядным компьютерам, а второе — к 64-разрядным):

```
C:/Program Files/Zend/Apache2/htdocs
C:/Program Files (x86)/Zend/Apache2/htdocs
```

Если вы не знаете, какой у вас компьютер, 32- или 64-разрядный, попробуйте перейти в первый каталог. Если он существует, значит, у вас 32-разрядная машина. Если нет, откройте второй каталог, поскольку у вас 64-разрядный компьютер.

Теперь, чтобы убедиться в том, что все сконфигурировано должным образом, нужно создать обязательную программу Hello World. Создайте небольшой HTML-файл, содержащий следующие строки, применив Windows-программу Блокнот или любое другое приложение или текстовый редактор (при использовании текстовых процессоров, таких как Microsoft Word, которые создают форматированный текст, файл нужно сохранять в виде простого текста):

```
<html>
<head>
   <title>A quick test</title>
</head>
<body>
   Hello World!
</body>
</html>
```

После набора сохраните файл в ранее упомянутом каталоге, являющемся исходным источником документов, использовав в качестве имени файла `test.htm`. Если вы создаете файл в Блокноте, убедитесь в том, что в окне Сохранить как в списке Тип файла выбран вариант Все файлы, а не Текстовые документы (*.txt).

Теперь вы сможете вызвать эту страницу в своем браузере, введя в адресной строке следующий URL-адрес (рис. 2.13):

```
http://localhost/test.htm
```

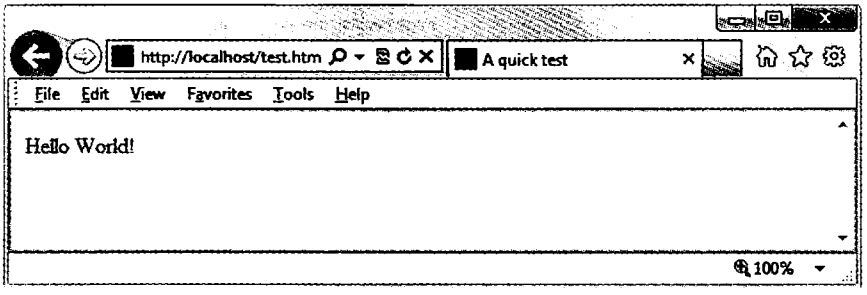



**Рис. 2.13.** Ваша первая веб-страница

Теперь вы получили успешно установленный программный продукт с полностью работоспособной системой WAMP. Если у вас возникнут трудности, обратитесь к подробной документации, размещенной по адресу http://tinyurl.com/zendcedocs.

# Другие системы WAMP

При обновлении программы иногда работают неожиданным для вас образом и в них даже могут проявиться какие-нибудь ошибки. Поэтому если с использованием Zend Server CE возникают неразрешимые трудности, то вместо этого сервера вы можете остановить свой выбор на одном из многих других, к которому есть доступ в Интернете.

При выборе другого сервера вы точно так же сможете воспользоваться всеми примерами, приводимыми в данной книге, но при этом придется пользоваться инструкциями, поставляемыми с каждым WAMP-сервером, что может быть сложнее, чем следовать ранее упомянутому руководству.

Вот наиболее подходящие, на мой взгляд, серверы:

- EasyPHP: http://www.easyphp.org;
- XAMPP: http://apachefriends.org/en/xampp.html;

- WAMPServer: http://wampserver.com/en/;
- Glossword WAMP: http://glossword.biz/glosswordwamp/.

# Установка MAMP на систему Mac OS X

Zend Server CE также доступен и для OS X. Его можно загрузить на странице **http://tinyurl.com/zendce** (рис. 2.14).

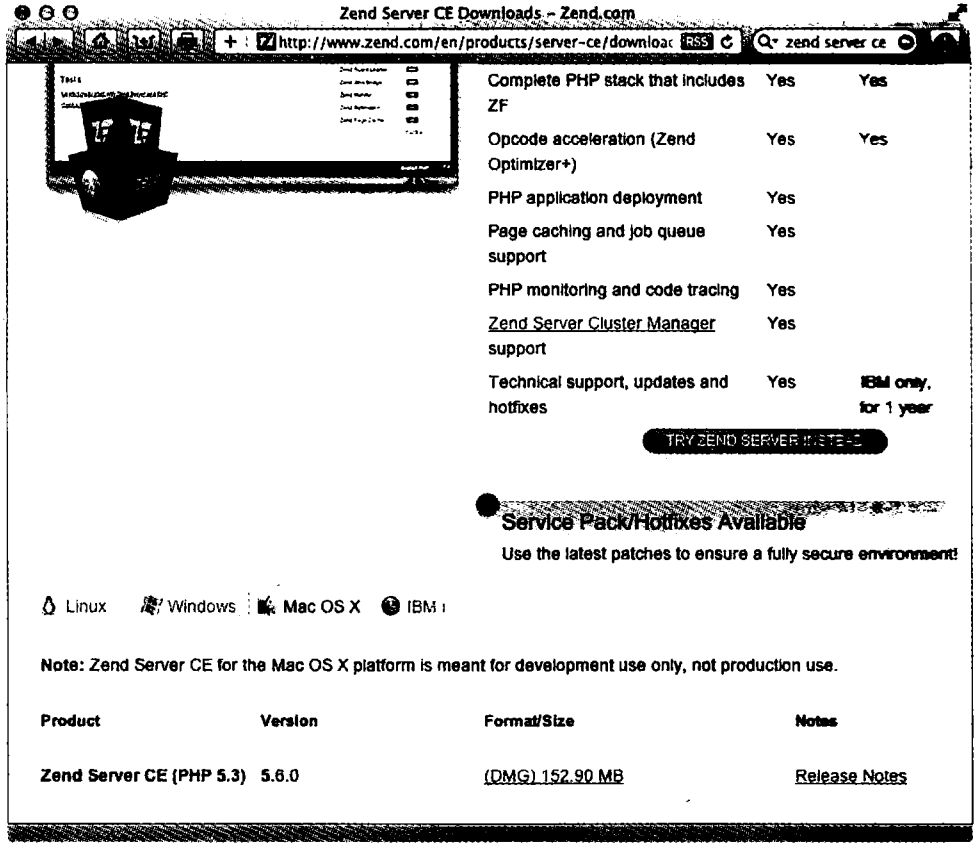



**Рис. 2.14.** Zend Server CE для OS X можно загрузить с веб-сайта Zend

Всегда загружайте самые последние стабильные версии (в данном случае это версия 5.6.0 SP1 для OS X), которые на веб-странице в разделе загрузки указываются первыми в списке. На странице должен быть изображен нужный установщик для вашего компьютера: Linux, Windows или OS X. Перед загрузкой вам будет предложено войти на сайт под своим именем. Можете щелкнуть на ссылке для получения файла без входа и без регистрации, но тогда вам не будут приходить по электронной почте уведомления об обновлении продукта и другие новости.

Когда загрузка завершится, дважды щелкните на файле с расширением DMG. Дождитесь проверки загрузки, после чего должно появиться окно, показанное на рис. 2.15. Если вы хотите ознакомиться с инструкциями, дважды щелкните на ссылке на файл READ ME. Перетащите Zend Server (Сервер Zend) и Zend Controller (Контроллер Zend) в свою папку Applications, которая доступна в меню Go (Перейти) файлового менеджера Finder, чтобы установить сервер и контроллер.

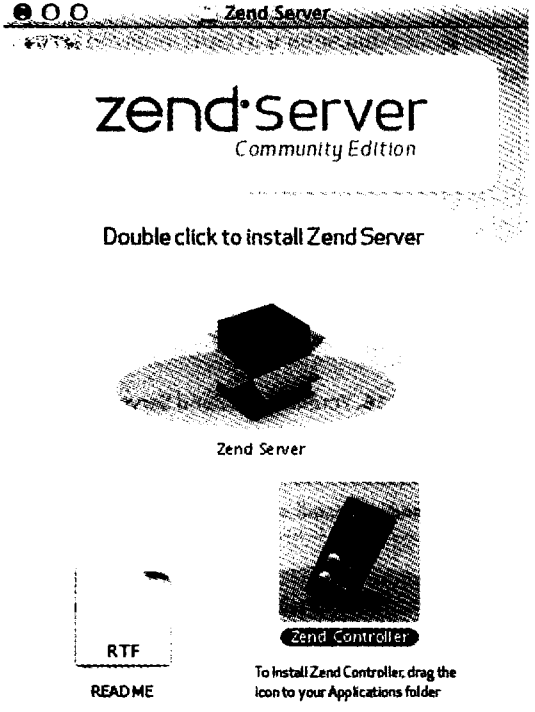



**Рис. 2.15.** Перетащите приложения в папку Applications (Приложения)

Чтобы приступить к работе, найдите файл Zend Server в папке Applications. Дважды щелкните на нем, чтобы открыть окно установки (рис. 2.16).

Щелкните на кнопке Continue (Далее), прочитайте показанные после этого инструкции, а затем еще раз нажмите кнопку Continue (Далее), чтобы перейти на экран, показанный на рис. 2.17. Здесь можно принять решение, куда поместить устанавливаемое программное обеспечение. По умолчанию предлагается Macintosh HD. Когда все будет готово, нажмите Install (Установить) и введите при необходимости свой пароль.

В ходе инсталляции может быть задан вопрос, не хотите ли вы установить дополнительное программное обеспечение. Рекомендую принять все предложения, щелкнув на кнопке Install (Установить). Когда установка завершится, для закрытия установщика можно щелкнуть на кнопке Close (Закрыть).

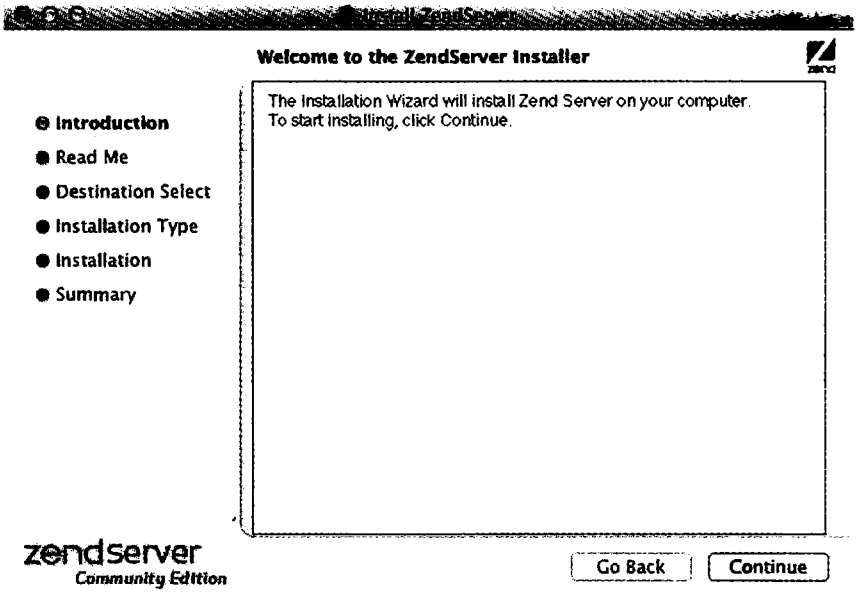



**Рис. 2.16.** Установщик Zend Server CE

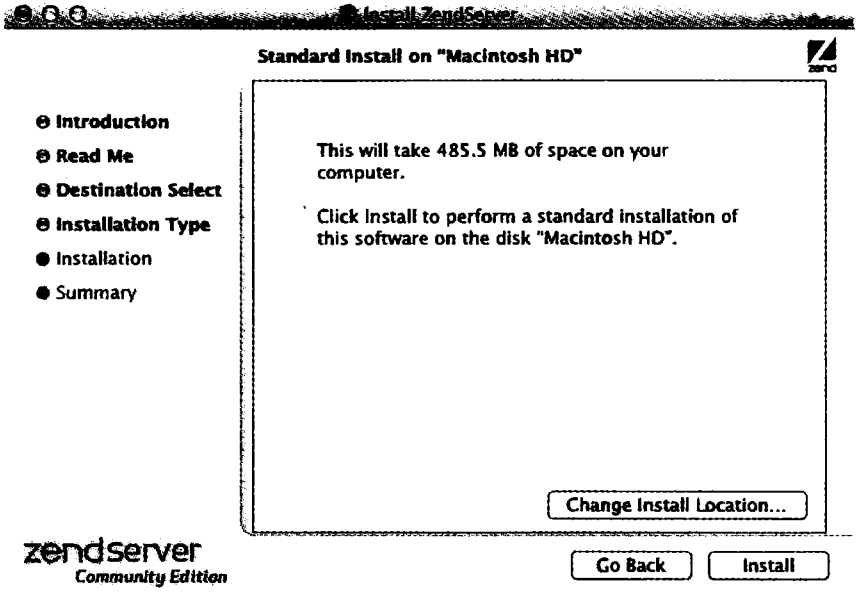



**Рис. 2.17.** Выбор места установки

Чтобы завершить настройку, необходимо после установки найти в папке Applications значок программы Zend Server и дважды щелкнуть на нем кнопкой мыши. В браузере, используемом по умолчанию, должна появиться страница, похожая на

показанную на рис. 2.8. Теперь следуйте приглашениям, показанным на рис. 2.8–2.10, согласно которым нужно принять условия лицензионного соглашения, выбрать пароль и завершить установку. После этого вы попадете на основную панель инструментов.

## Настройка MySQL

К сожалению, установщик Zend Server CE не устанавливает команды, необходимые для того, чтобы запускать, останавливать и перезапускать сервер MySQL, поэтому вам придется сделать это самостоятельно, для чего нужно открыть окно программы `Terminal` и ввести следующую команду:

```
sudo nano /usr/local/zend/bin/zendctl.sh
```

После ввода пароля вы окажетесь в текстовом редакторе `Nano`, где нужно будет с помощью клавиши стрелки вниз переместить курсор на несколько строк вниз и, когда покажется строка с текстом `MySQL_EN="false"`, изменить слово `false` на `true`.

Теперь опустите курсор еще ниже, пока не появятся следующие две строки:

```
case $1 in
        "start")
```

Ниже этих строк будет строка с отступом, имеющая следующий вид:

```
                $0 start-apache
```

Сразу же после этой строки вставьте новую строку, имеющую следующий вид:

```
                $0 start-MySQL
```

Это позволит MySQL запуститься. Теперь опустите курсор еще ниже, пока не появится раздел, начинающийся со следующей строки:

```
        "restart")
```

Ниже ее вы увидите строку с отступом, имеющую следующий вид:

```
                $0 restart-apache
```

Сразу же после этой строки вставьте новую строку:

```
                $0 restart-MySQL
```

Теперь создана возможность для перезапуска MySQL, но это еще не все. Чтобы можно было остановить MySQL, опустить курсор еще ниже, пока не появится следующая строка:

```
        "stop")
```

Ниже этой строки есть следующая строка с отступом:

```
                $0 stop-apache %
```

Сразу же после этой строки вставьте новую строку:

```
                $0 stop-MySQL
```

Теперь можно нажать сочетание клавиш Ctrl+X для выхода из режима редактирования. Когда поступит запрос на сохранение изменений, нажмите клавишу Y, а затем нажмите клавишу Return для сохранения отредактированного файла.

## Обеспечение запуска MySQL при загрузке системы

Чтобы программа MySQL запускалась при старте Mac-компьютера, нужно в программе Terminal ввести следующие команды:

```
cd /Library/StartupItems/ZendServer_init/
sudo rm zendctl.sh
sudo ln -s /usr/local/zend/bin/zendctl.sh ./
```

Теперь Mac настроен, но программа MySQL еще не запущена. Чтобы завершить подготовку, введите следующую команду:

```
sudo /Library/StartupItems/ZendServer_init/zendctl.sh restart
```

## Проверка установки

Теперь можно проверить установку путем ввода следующих URL в адресную строку браузера для вызова экрана, показанного на рис. 2.18:

```
http://localhost:10088
http://127.0.0.1:10088
```

Слово localhost указывает на локальный компьютер, который также будет откликаться на IP-адрес 127.0.0.1. Причина, по которой нужно вводить :10088 заключается в том, что на многих Mac-компьютерах уже есть запущенный веб-сервер: ввод этого адреса порта избавит от конфликтов. По этой же причине не следует забывать набирать :10088 после каждого http://localhost для всех примеров, приводимых в данной книге. Например, если используется файл с именем test.php, из браузера его нужно вызвать с помощью URL http://localhost:10088/test.php.

Если вы уверены в том, что на вашем Mac-компьютере нет никакого другого запущенного веб-сервера, можно отредактировать файл конфигурации (если у вас есть разрешение на это действие), который находится по адресу /usr/local/zend/apache2/conf/httpd.conf. Измените команду Listen 10088 (она находится приблизительно в строке под номером 40) на Listen 80.

Затем перезапустите сервер, открыв утилиту Terminal и запустив следующую команду: sudo /usr/local/zend/bin/zendctl.sh restart.

В результате вам не придется добавлять :10088 к локальным URL-адресам.

Страница, отображаемая в браузере при переходе по адресу http://localhost или http://localhost:10088, представляет собой файл index.html, который находится в каталоге, являющемся исходным источником документов и содержащем главные веб-документы домена. Именно он вводится, когда в адресной строке браузера набирается базовый URL-адрес без указания пути, например http://yahoo.com, или в случает использования вашего локального веб-сервера — http://localhost.

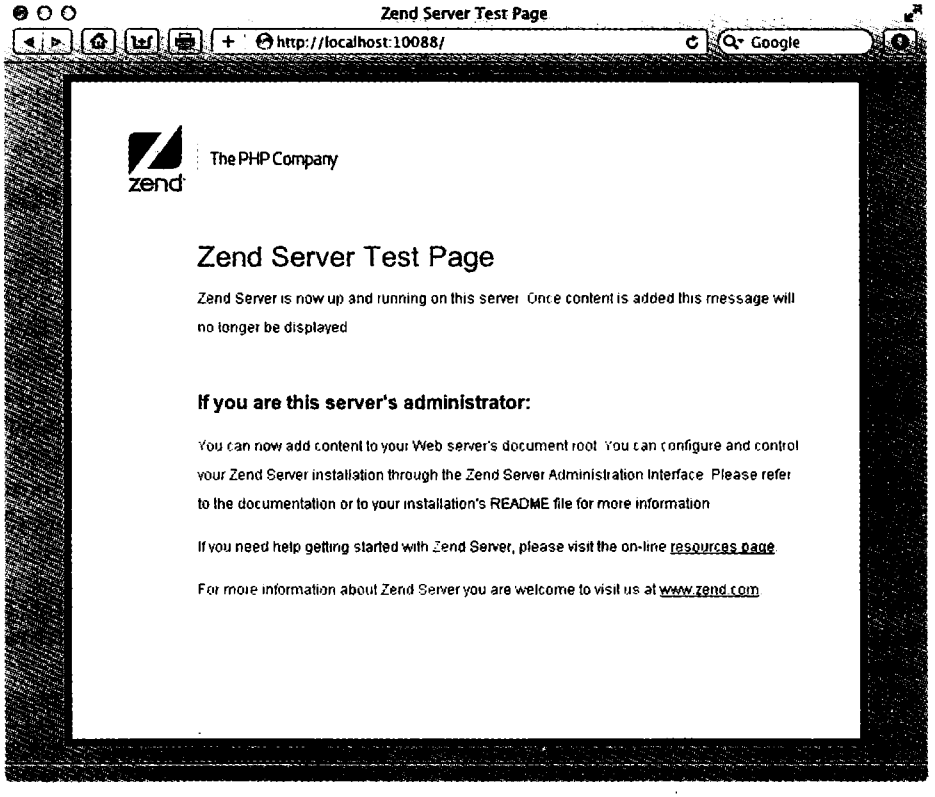



**Рис. 2.18.** Веб-сервер запущен и работает

По умолчанию Zend Server CE использует в качестве своей папки исходного источника документов следующий каталог:

```
/usr/local/zend/apache2/htdocs
```

Теперь, чтобы убедиться в правильности всех настроек, нужно загрузить тестовый файл. С помощью, например, программы TextEdit (при использовании текстовых процессоров, таких как Microsoft Word, которые создают форматированный текст, файл нужно сохранять в виде простого текста) создайте небольшой HTML-файл, состоящий всего из нескольких строк:

```
<html>
  <head>
    <title>A quick test</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

После набора этого текста сохраните файл в каталоге исходного источника документов, используя имя файла `test.htm`. Теперь вы можете вызвать эту

страницу в своем браузере путем ввода в адресную строку следующего URL-адреса (рис. 2.19):

```
http://localhost:10088/test.htm
```

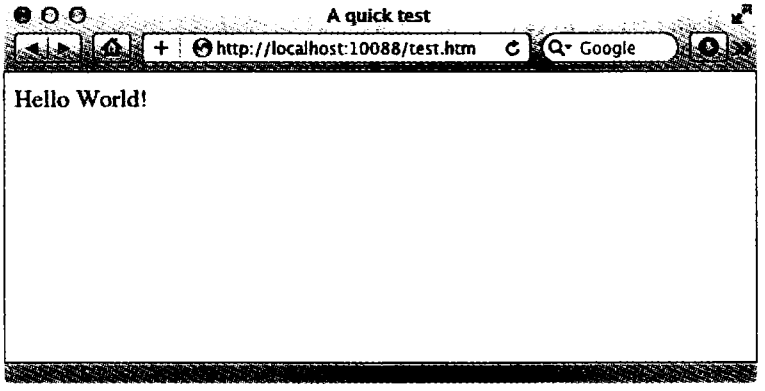



**Рис. 2.19.** Если вы видите эту страницу, значит, все работает правильно

Вы получили успешно установленный программный продукт с полностью работоспособной системой MAMP. При появлении проблем вы можете обратиться к документации, размещенной по адресу http://tinyurl.com/zendcedocs.

Для удобства я использую в книге для длинных URL-адресов, которые сложно набирать, службу сокращения веб-адресов tinyurl.com. Например, адрес http://tinyurl.com/zendcedocs намного короче, чем тот URL-адрес, к которому он приводит: http://files.zend.com/help/Zend-Server-Community-Edition/zend-server-community-edition.htm.

# Установка LAMP на Linux

Инструкции и примеры, приведенные в этой книге, также будут хорошо работать и на Linux-компьютере. Существуют десятки популярных разновидностей Linux, на каждую из которых LAMP может устанавливаться со своими особенностями. Многие версии Linux поступают с предустановленным веб-сервером и MySQL, поэтому есть вероятность, что у вас уже все готово к работе. Чтобы понять, так ли это, попробуйте ввести в браузер следующий адрес и посмотрите, получите ли вы веб-страницу, используемую по умолчанию в исходном источнике документов:

```
http://localhost
```

Если все заработает, то у вас, наверное, установлен сервер Apache, а также может быть установлена и запущена база данных MySQL, но чтобы окончательно удостовериться в этом, проверьте факт их установки вместе со своим системным администратором.

Если веб-сервер не установлен, можете воспользоваться версией Zend Server CE, которую можно загрузить со страницы http://tinyurl.com/zendce.

Все необходимые инструкции можно найти на странице загрузки. Вы можете точно следовать этим инструкциям или воспользоваться предоставляемыми сценариями, и тогда вы сможете работать со всеми примерами, приведенными в книге.

# Работа в удаленном режиме

Если есть доступ к веб-серверу, на котором уже имеются сконфигурированные PHP и MySQL, то вы всегда можете применить его для веб-разработок. Но это не самый лучший вариант, если только вы не пользуетесь высокоскоростным подключением. Разработка, выполняемая на локальной машине, позволяет протестировать новые модификации практически без задержки или с небольшой задержкой на загрузку.

Может вызвать трудности и удаленный доступ к MySQL. Для ручного создания баз данных и установки прав доступа из командной строки может понадобиться сервер Telnet или SSH. Компания, предоставляющая веб-хостинг, посоветует, как это можно сделать наилучшим образом, и предоставит пароль для доступа к MySQL (а в первую очередь, разумеется, для доступа к самому серверу).

## Вход в систему

Я рекомендую пользователям Windows для доступа к Telnet и SSH (следует помнить, что уровень безопасности у SSH значительно выше, чем у Telnet) установить как минимум программу PuTTY, доступную по адресу http://putty.org.

На компьютере Mac система SSH будет доступна изначально. Нужно только выбрать папку Applications, перейти к папке Utilities, а затем запустить программу `Terminal`. В окне терминала нужно войти на сервер, используя SSH в следующей команде:

```
ssh mylogin @ server.com
```

где `server.com` — это имя сервера, на который необходимо войти, а `mylogin` — имя пользователя, под которым нужно войти в систему. Затем у вас будет запрошен пароль для данного имени пользователя, и если вы введете правильный пароль, вход состоится.

## Использование FTP

Для переноса файлов на веб-сервер и обратно понадобится FTP-программа. Если искать подходящую программу в Интернете, можно найти так много программ, что на выбор нужной именно вам уйдет уйма времени.

На данный момент я рекомендую программу FireFTP, потому что она обладает следующими преимуществами:

- она является дополнением к веб-браузеру Firefox и поэтому будет работать на любой платформе, на которой работает этот браузер;
- вызвать ее так же просто, как выбрать закладку;
- это одна из самых быстрых и простых в использовании FTP-программ из тех, которые мне когда-либо попадались.

Вы можете возразить: «Но я пользуюсь только Microsoft Internet Explorer, а FireFTP для него недоступна». Я бы на это ответил, что, раз уж вы решили заняться разработкой веб-страниц, вам все равно понадобится установить на свой компьютер все основные браузеры, что уже предлагалось сделать в начале этой главы.

Для установки FireFTP нужно пройти на веб-сайт http://fireftp.mozdev.org, используя Firefox, и щелкнуть на ссылке Download FireFTP (Скачать FireFTP). Программа занимает примерно 500 Кбайт и устанавливается очень быстро. После инсталляции нужно перезапустить Firefox, и тогда можно будет получить доступ к FireFTP из пункта меню Tools (Инструменты) (рис. 2.20).

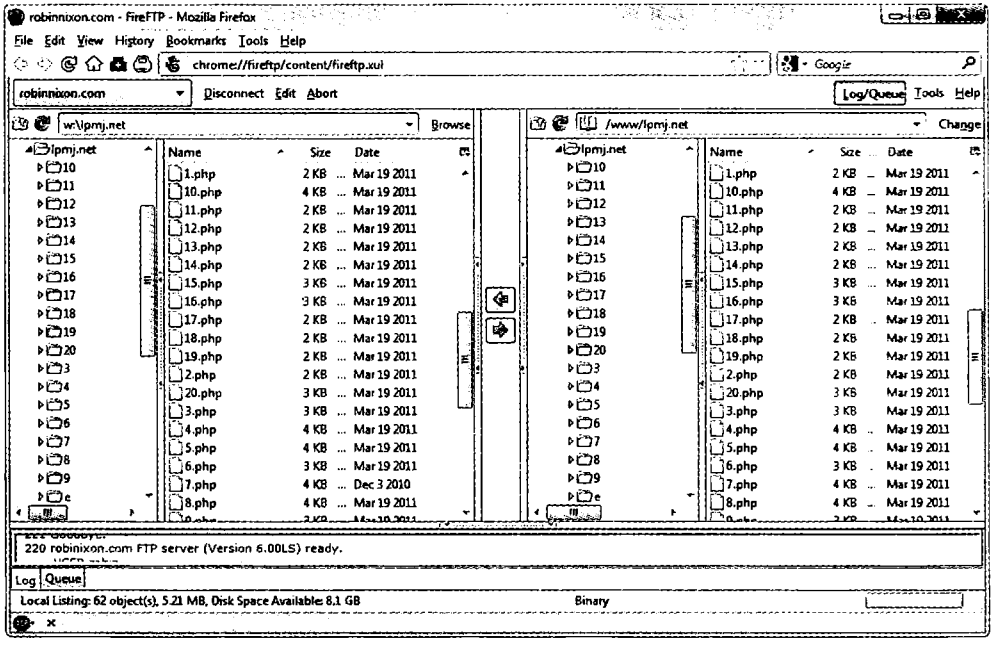



**Рис. 2.20.** FireFTP предоставляет полный доступ к FTP прямо из Firefox

Еще одной замечательной FTP-программой является FileZilla, доступная по адресу http://filezilla-project.org в версиях для Windows, Linux и Mac OS X 10.5 и выше.

Разумеется, если у вас уже есть FTP-программа, вы можете использовать ее.

# Использование редактора программ

Хотя для редактирования HTML, PHP, JavaScript и CSS подходит любой текстовый редактор, существуют очень удобные приложения, специально предназначенные для редактирования текста программ, в которых имеются весьма полезные возможности, например цветная подсветка синтаксиса. Современные редакторы программ хорошо продуманы и могут еще до запуска программы на выполнение показывать места, в которых допущены синтаксические ошибки. Перейдя к использованию современного редактора, вы будете удивлены тому, что раньше обходились без него.

Есть множество хороших и доступных программ, но я остановил свой выбор на программе Editra, поскольку она распространяется бесплатно и доступна для Mac OS X, Windows и Linux/Unix (рис. 2.21). Программы можно загрузить, перейдя на веб-сайт http://editra.org и выбрав ссылку Download (Скачать) в левом верхнем углу страницы, где также можно найти и документацию на редактор.

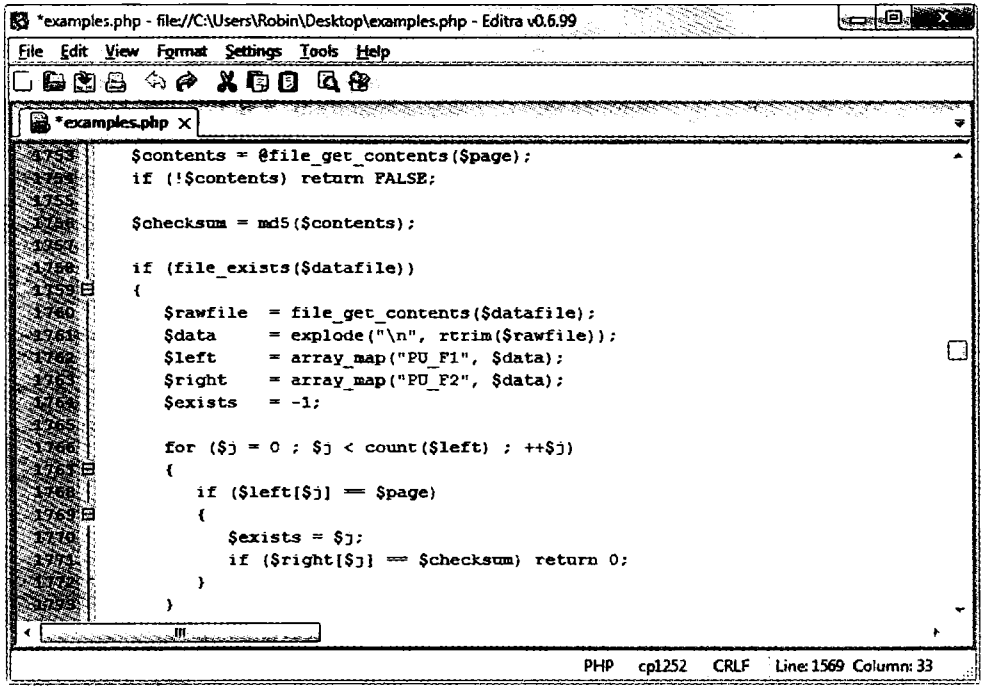



**Рис. 2.21.** Окно программы Editra

Editra выделяет синтаксис, используя соответствующие цвета, что очень удобно. Можно поместить курсор за квадратными или фигурными скобками, и Editra подсветит соответствующую парную скобку, давая возможность определить лишние или недостающие скобки. В Editra предлагается также множество других свойств, облегчающих работу с текстом программы.

# Использование IDE

При всех достоинствах специализированных редакторов программ, позволяющих повысить производительность труда программиста, они не могут сравниться с интегрированными средами разработки (Integrated Development Environment, IDE), предлагающими множество дополнительных возможностей, например проведение отладки и тестирования программ прямо в редакторе, а также отображение описаний функций и многое другое.

На рис. 2.22 показана популярная интегрированная среда разработки phpDesigner с программой PHP, которая загружена в главное окно, и с расположенным в правой части анализатором кода Code Explorer, в котором перечислены различные классы, функции и переменные, используемые в этой программе.

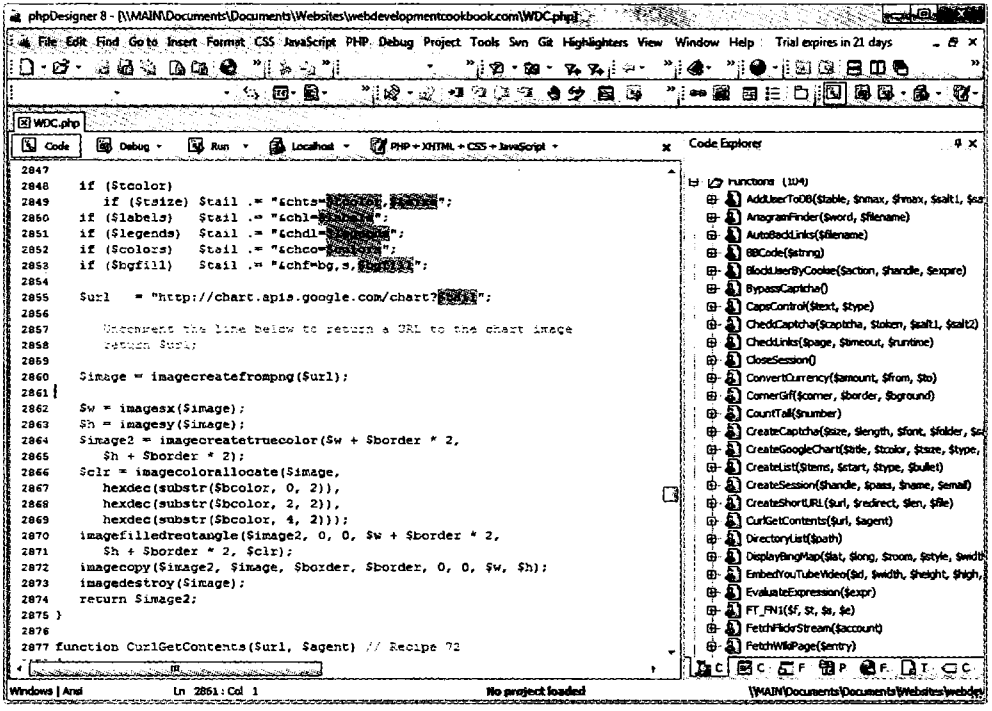



**Рис. 2.22.** При использовании такой IDE, как phpDesigner, разработка PHP-программы идет намного быстрее и проще

При разработке продукта в IDE можно установить контрольные точки, а затем запустить весь код целиком (или по частям), тогда его выполнение будет останавливаться в контрольных точках и вам будет предоставляться информация о текущем состоянии программы.

Помочь изучению процесса создания программ может то, что приводимые в данной книге примеры могут быть введены в IDE и запущены в этой среде, тогда вызывать браузер уже не понадобится.

Существует несколько интегрированных сред разработки, доступных для различных платформ. Большинство из них являются коммерческими продуктами, но встречаются и бесплатные версии. В табл. 2.1 приведены некоторые наиболее популярные интегрированные среды разработки PHP-программ и URL-адреса, с которых их можно загрузить.

**Таблица 2.1.** Интегрированные среды разработки для PHP

| IDE | URL-адрес загрузки | Цена | Win | Mac | Linux |
|---|---|---|---|---|---|
| Eclipse PDT | http://eclipse.org/pdt/downloads/ | Бесплатно | ✓ | ✓ | ✓ |
| Komodo IDE | http://activestate.com/Products/komodo_ide | $245 | ✓ | ✓ | ✓ |
| NetBeans | http://www.netbeans.org | Бесплатно | ✓ | ✓ | ✓ |
| phpDesigner | http://mpsoftware.dk | $39 | ✓ | – | – |
| PHPEclipse | http://phpeclipse.de | Бесплатно | ✓ | ✓ | ✓ |
| PhpED | http://nusphere.com | $119 | ✓ | – | ✓ |
| PHPEdit | http://phpedit.com | $119 | ✓ | – | – |
| Zend Studio | http://zend.com/en/downloads | $395 | ✓ | ✓ | ✓ |

Выбор IDE может иметь сугубо индивидуальный характер, поэтому, если вы склоняетесь к использованию какой-нибудь одной среды, я советую загрузить не менее двух разновидностей этих программ, чтобы сначала опробовать их в работе. Все они либо бесплатны, либо имеют пробную версию, поэтому вы не понесете никаких затрат.

Вооружившись редактором программ или IDE, вы подготовитесь к переходу к главе 3, в которой начнется углубленное изучение PHP и совместной работы HTML и PHP, а также структуры самого языка PHP. Но перед тем, как перейти к этой главе, я предлагаю проверить вновь приобретенные знания, ответив на следующие вопросы.

# Проверьте ваши знания

1. В чем разница между WAMP, MAMP и LAMP?
2. Что общего у IP-адреса 127.0.0.1 и URL-адреса http://localhost?
3. Для чего предназначена FTP-программа?
4. Назовите основной недостаток работы на удаленном веб-сервере.
5. Почему лучше воспользоваться редактором программ, а не обычным текстовым редактором?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 2».

# 3 Введение в PHP

В главе 1 о PHP говорилось как о языке, заставляющем сервер генерировать динамическую, потенциально разную выходную информацию при каждом запросе браузером веб-страницы. В данной главе начнется изучение этого простого, но мощного языка, которое продолжится в следующих главах и завершится в главе 6.

Я призываю вас выполнять разработку кода PHP в одной из интегрированных средств разработки (IDE), упомянутых в главе 2. Она поможет выявить опечатки, что существенно ускорит обучение, по сравнению с работой в менее функциональных редакторах.

Многие из IDE позволяют запускать код, рассматриваемый в этой главе, и изучать производимую им выходную информацию. Мы также изучим методы вставки кода PHP в файл HTML, чтобы иметь представление о внешнем виде выходной информации на веб-странице (то есть о том виде, в котором она в итоге предстанет перед пользователями).

В процессе создания веб-страницы будут представлять собой комбинацию PHP, HTML, JavaScript, инструкций MySQL и форматирования с помощью CSS. Кроме того, каждая страница может привести на другие страницы, предоставляя пользователям возможности щелкать на ссылках и заполнять формы. Хотя при изучении этих языков можно обойтись и без этих сложностей. На данном этапе нужно сконцентрироваться исключительно на написании PHP-кода и на достижении предсказуемости содержимого выходной информации или по крайней мере на умении разбираться в характере этой информации.

## Включение PHP в HTML

По умолчанию в конце имен PHP-документов ставится расширение PHP. Когда веб-сервер встречает в запрашиваемом файле это расширение, он автоматически передает файл PHP-процессору. Веб-серверы имеют довольно широкий диапазон настроек, и некоторые веб-разработчики выбирают такой режим работы, при котором для разбора PHP-процессору принудительно передаются также файлы с расширениями HTM или HTML. Обычно это связано с тем, что разработчики хотят скрыть факт использования PHP.

Программа на PHP отвечает за возвращение файла в чистом виде, пригодном для отображения в веб-браузере. В простейшем случае на выходе документа PHP

будет получаться только код HTML. Чтобы убедиться в этом, можно взять любой обычный HTML-документ, например файл `index.html`, сохранить его под именем `index.php`, и он будет отображаться точно так же, как и исходный файл.

## Вызов парсера PHP

Для запуска команд PHP нужно изучить новый тег. Его открывающая часть имеет следующий вид:

```
<?php
```

Первое, что может броситься в глаза, — незавершенность тега. Это обусловлено тем, что внутри тега могут помещаться целые фрагменты кода PHP. Они заканчиваются, только когда встречается закрывающая часть тега, имеющая следующий вид:

```
?>
```

Небольшая PHP-программа Hello World может иметь вид, показанный в примере 3.1.

**Пример 3.1.** Вызов PHP

```
<?php
echo "Hello world";
?>
```

Этот тег очень гибок в использовании. Некоторые программисты открывают тег в начале документа, а закрывают его в самом конце и выводят любой код HTML путем непосредственного использования команды PHP.

Другие программисты предпочитают помещать в эти теги как можно меньшие фрагменты кода PHP, и именно в тех местах, где нужно воспользоваться динамическими сценариями, а весь остальной документ составлять из стандартного кода HTML.

Сторонники последнего метода программирования зачастую аргументируют свой выбор тем, что такой код выполняется быстрее, а сторонники первого метода утверждают, что увеличение скорости настолько мизерное, что оно не может оправдать дополнительные сложности многочисленных вставок PHP в отдельно взятый документ.

По мере изучения языка вы, несомненно, определитесь в своих стилевых предпочтениях при создании разработок на PHP, но в целях упрощения примеров, приводимых в этой книге, я свел количество переходов между PHP и HTML к минимуму, в среднем к одному-двум переходам на один документ.

Кстати, существует и несколько иной вариант синтаксиса PHP. Если поискать примеры PHP-кода в Интернете, то можно встретить код, где используется следующий синтаксис открывающего и закрывающего тегов:

```
<?
echo "Hello world";
?>
```

Несмотря на то что здесь неочевиден вызов PHP-парсера, это вполне приемлемый альтернативный синтаксис, который, как правило, также работает. Но я не

советую его использовать, поскольку он несовместим с XML и в настоящее время его применение не приветствуется (это значит, что он больше не рекомендуется и может быть удален в будущих версиях).

Если в файле содержится только код PHP, то закрывающий тег ?> можно опустить. Именно так и нужно делать, чтобы гарантировать отсутствие в файлах PHP лишнего пустого пространства (что имеет особую важность при написании объектно-ориентированного кода).

# Примеры, приводимые в этой книге

Чтобы вы не тратили время на набор примеров, приводимых в этой книге, все они заархивированы на веб-сайте http://lpmj.net, откуда можно загрузить файл `2nd_edition_examples.zip`, щелкнув на ссылке Download Examples (Загрузить примеры), которая находится в верхней части страницы (рис. 3.1).

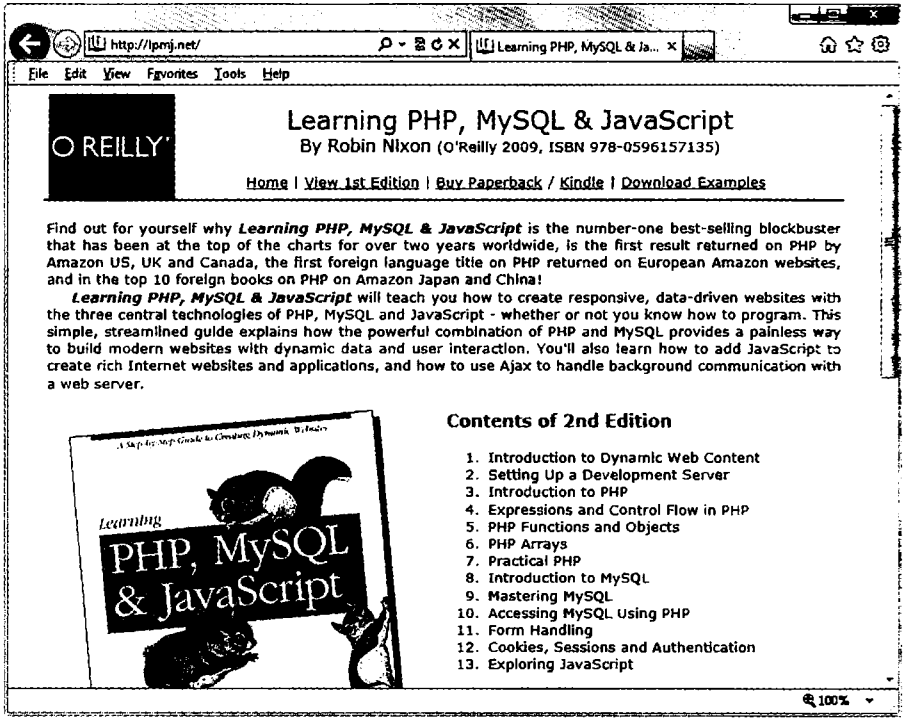



**Рис. 3.1.** Примеры, приводимые в этой книге, можно просмотреть по адресу http://lpmj.net

Все примеры хранятся под номерами и сгруппированы по главам (например, `example3-1.php`). Кроме того, на сайте имеется архив `examples.zip`, в котором дополнительно есть папка `named_examples`, где можно найти все примеры, которые пред-

лагалось сохранять в файлах с конкретными именами (как в показанном далее примере 3.4, который нужно будет сохранить в файле с именем test1.php).

# Структура PHP

В этом разделе будет рассмотрено довольно много основных положений. Разобраться во всем этом будет несложно, но я рекомендую проработать материал как можно тщательнее, поскольку он служит основой для понимания всей остальной книги. Как всегда, в конце главы будут заданы вопросы, с помощью которых можно будет проверить, насколько глубоко усвоен материал.

## Использование комментариев

Существует два способа добавления комментариев к коду PHP. Первый, предусматривающий размещение в начале строки двух прямых слешей, превращает в комментарий отдельную строку:

```
// Это комментарий
```

Он хорошо подходит для временного исключения из программы строки кода, являющейся источником ошибок. Например, такой способ комментирования можно применить для того, чтобы скрыть строку кода до тех пор, пока в ней не возникнет необходимость:

```
// echo "X equals $x";
```

Такой комментарий можно также вставить сразу же после строки кода, чтобы описать ее действие:

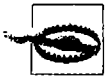
```
$x += 10; // Увеличение значения $x на 10
```

Когда понадобится комментарий, состоящий из нескольких строк, нужно воспользоваться вторым способом комментирования, который показан в примере 3.2.

**Пример 3.2.** Многострочный комментарий

```
<?php
/* Это область
   многострочного комментария,
   которая не будет
   подвергаться интерпретации */
?>
```

Для открытия и закрытия комментария можно воспользоваться парами символов /* и */ практически в любом произвольно выбранном месте кода. Если не все, то большинство программистов используют эту конструкцию для временного превращения в комментарий целого неработоспособного раздела кода или такого раздела, который по тем или иным причинам нежелательно интерпретировать.



Типичной ошибкой является применение пар символов /* и */ для того, чтобы закомментировать большой фрагмент кода, который уже содержит закомментированную область, где используются эти же пары символов. Комментарии не могут быть вложенными друг в друга, поскольку PHP-интерпретатор не поймет, где заканчивается комментарий, и выведет на экран сообщение об ошибке. Но если вы используете редактор программ или интегрированную среду разработки с подсветкой синтаксиса, то ошибку такого рода нетрудно будет заметить.

# Основной синтаксис

PHP — очень простой язык, уходящий своими корнями в языки C и Perl, но все же больше похожий на Java. Он очень гибок, но существует несколько правил, относящихся к его синтаксису и структуре, которые следует изучить.

## Точки с запятыми

В предыдущих примерах можно было заметить, что команды PHP завершаются точкой с запятой:

```
$x += 10;
```

Возможно, чаще всего причиной ошибок, с которыми приходится сталкиваться при работе с PHP, становится забывчивость. Если не поставить эту точку с запятой, PHP вынужден будет рассматривать в качестве одной сразу несколько инструкций, при этом он не сможет разобраться в ситуации и выдаст ошибку синтаксического разбора — Parse error.

## Символ $

Символ $ используется в разных языках программирования в различных целях. Например, в языке BASIC символ $ применялся в качестве завершения имен переменных, чтобы показать, что они относятся к строкам.

А в PHP символ $ должен ставиться перед именами всех переменных. Это нужно для того, чтобы PHP-парсер работал быстрее, сразу же понимая, что имеет дело с переменной. К какому бы типу ни относились переменные, к числам, строкам или массивам, все они должны выглядеть так, как показано в примере 3.3.

**Пример 3.3.** Три разновидности присваивания значений переменным

```
<?php
$mycounter = 1;
$mystring  = "Hello";
$myarray   = array("One", "Two", "Three");
?>
```

Вот, собственно, и весь синтаксис, который нужно усвоить. В отличие от таких языков, как Python, в котором отношение к способам отступа текста программы и размещения кода очень строгое, PHP даст полную свободу использования (или игнорирования) любых отступов и любого количества пробелов по вашему усмотрению. В действительности же разумное использование того, что называется свободным

пространством, обычно поощряется (наряду с всесторонним комментированием), поскольку помогает разобраться в собственном коде, когда к нему приходится возвращаться по прошествии некоторого времени. Это помогает и другим программистам, вынужденным поддерживать ваш код.

# Осмысление переменных

Понять, что такое переменные PHP, поможет простая метафора. Думайте о них, как о небольших (или больших) спичечных коробках! Именно как о спичечных коробках, которые вы раскрасили и на которых написали некие имена.

## Строковые переменные

Представьте, что у вас есть коробок, на котором написано слово username (имя пользователя). Затем вы пишете на клочке бумаги Fred Smith и кладете эту бумажку в коробок (рис. 3.2). Этот процесс похож на присваивание переменной строкового значения:

```
$username = "Fred Smith";
```

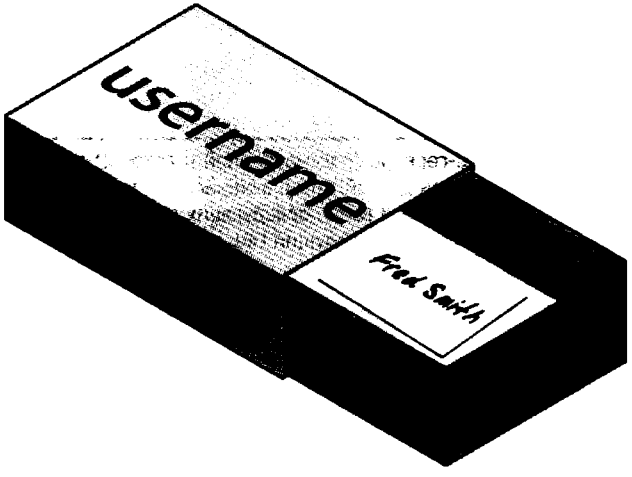



**Рис. 3.2.** Переменные можно представить в виде спичечного коробка, содержащего какие-то предметы

Кавычки служат признаком того, что Fred Smith является *строкой* символов. Каждую строку нужно заключать либо в двойные, либо в одинарные кавычки (апострофы). Между этими двумя видами кавычек есть весьма существенное различие, которое будет рассмотрено далее.

Когда хочется посмотреть, что находится внутри коробка, вы его открываете, вынимаете бумажку и читаете, что на ней написано. В PHP подобное действие выглядит следующим образом:

```
echo $username;
```

Можно также присвоить содержимое другой переменной (то есть сделать ксерокопию бумажки и поместить ее в другой коробок):

```
$current_user = $username;
```

Если вы стремитесь самостоятельно освоить работу с PHP, то можете попробовать вводить примеры, приводимые в этой главе, в интегрированную среду разработки (согласно рекомендациям, которые были даны в конце главы 2), чтобы тут же посмотреть на результаты, или же можете ввести код примера 3.4 в редактор программ (который также рассматривался в главе 2) и сохранить этот код в каталоге исходного источника документов вашего сервера под именем `test1.php`.

**Пример 3.4.** Ваша первая PHP-программа

```
<?php // test1.php
$username = "Fred Smith";
echo $username;
echo "<br />";
$current_user = $username;
echo $current_user;
?>
```

Теперь эту программу можно запустить путем ввода в адресную строку браузера следующего адреса:

```
http://localhost/test1.php
```

Если в ходе установки веб-сервера (рассмотренной в главе 2) вы изменили назначенный серверу порт на какой-нибудь другой, отличающийся от порта 80, вы должны поместить номер этого порта в URL в этом и всех последующих примерах из этой книги. Например, если вы изменили порт на 8080, предыдущий URL приобретет следующий вид:

http://localhost:8080/test1.php

Не забудьте об этом при тестировании других примеров из книги или при написании собственного кода.

Результатом запуска этого кода будет двойное появление имени `Fred Smith`, первое — в результате выполнения команды `echo $username`, а второе — в результате выполнения команды `echo $current_user`.

## Числовые переменные

Переменные могут содержать не только строки, но и числа. Пользуясь аналогией со спичечным коробком, сохранение в переменной `$count` числа 17 будет эквивалентно помещению, скажем, 17 бусин в коробок, на котором написано слово count:

```
$count = 17;
```

Можно также использовать числа с плавающей точкой (содержащие десятичную точку); синтаксис остается прежним:

```
$count = 17.5;
```

Чтобы узнать о содержимом коробка, его нужно просто открыть и посчитать бусины. В PHP можно присвоить значение переменной $count другой переменной или вывести его с помощью веб-браузера на экран, воспользовавшись командой echo.

## Массивы

*Массивы* можно представить в виде нескольких склеенных вместе спичечных коробков. Например, нам нужно сохранить имена пяти футболистов одной команды в массиве $team. Для этого мы склеим вместе боковыми сторонами пять коробков, запишем имена всех игроков на отдельных клочках бумаги и положим каждый клочок в свой коробок.

Вдоль всей верхней стороны склеенных вместе коробков напишем слово team (рис. 3.3). В PHP эквивалентом этому действию будет следующий код:

```
$team = array('Bill', 'Joe', 'Mike', 'Chris', 'Jim');
```

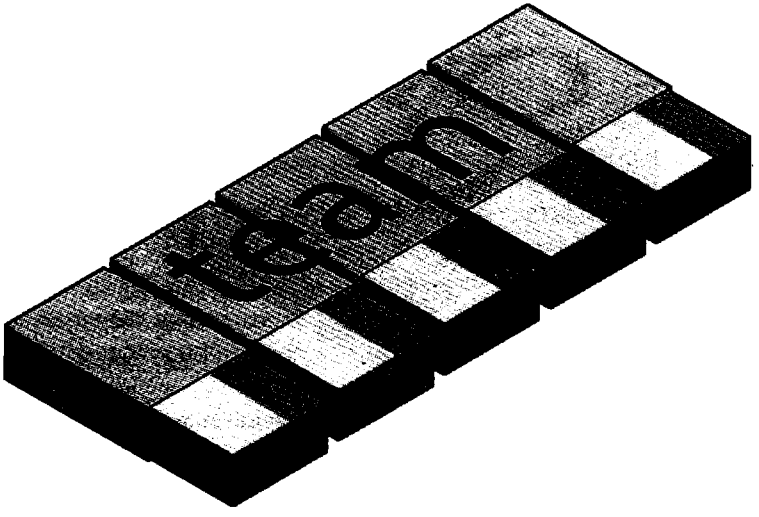



**Рис. 3.3.** Массив похож на несколько склеенных вместе спичечных коробков

Этот синтаксис несколько сложнее рассмотренных ранее инструкций. Код создания массива представляет собой следующую конструкцию:

```
array();
```

с пятью строками внутри круглых скобок. Каждая строка заключена в одинарные кавычки.

Когда потребуется узнать, кто является игроком номер 4, можно воспользоваться следующей командой:

```
echo $team[3]; // Эта команда отображает имя Chris
```

Использование в предыдущем примере числа 3, а не 4 обусловлено тем, что первый элемент PHP-массива является, как правило, нулевым, поэтому номера игроков распределяются в интервале от 0 до 4.

## Двумерные массивы

Диапазон использования массивов очень широк. Например, вместо выстраивания одномерных рядов коробков из них можно построить двумерную матрицу, а массивы могут иметь три и более измерения.

Чтобы привести пример двумерного массива, представим, что нужно отслеживать ход игры в крестики-нолики, для чего требуется структура данных, состоящая из девяти клеток, сгруппированных в квадрат 3 × 3. Чтобы представить это в виде спичечных коробков, вообразите себе девять коробков, склеенных в матрицу, состоящую из трех строк и трех столбцов (рис. 3.4).


**Рис. 3.4.** Многомерный массив, смоделированный с помощью коробков

Теперь для каждого хода можно класть в нужные коробки клочки бумаги с крестиком или ноликом. Чтобы сделать это в коде PHP, необходимо создать массив, содержащий три других массива, как в примере 3.5, в котором массив создается для отображения уже ведущейся игры.

**Пример 3.5.** Определение двумерного массива

```
<?php
$oxo = array(array('x', '',  'o'),
             array('o', 'o', 'x'),
             array('x', 'o', '' ));
?>
```

Мы сделали еще один шаг к усложнению, но смысл его нетрудно понять, если усвоен основной синтаксис массива. Здесь три конструкции array() вложены во внешнюю по отношению к ним конструкцию array().

Для возвращения в дальнейшем третьего элемента во второй строке этого массива можно воспользоваться следующей PHP-командой, которая отобразит символ x

```
echo $oxo[1][2];
```

Не забывайте о том, что отсчет индексов массива (указателей на элементы внутри массива) начинаются с нуля, а не с единицы, поэтому в предыдущей команде индекс [1] ссылается на второй из трех массивов, а индекс [2] — на третью позицию внутри этого массива. Указанная выше команда вернет содержимое третьего слева и второго сверху коробка.

Как уже упоминалось, поддерживаются даже массивы с большей размерностью, получаемые путем простого создания большего количества вложенных друг в друга массивов. Но в данной книге массивы с размерностью больше двух рассматриваться не будут.

Подробнее массивы будут рассмотрены в главе 6.

### Правила присваивания имен переменным

При создании PHP-переменных следует придерживаться следующих четырех правил.

- Имена переменных должны начинаться с буквы или с символа _ (подчеркивания).
- Имена переменных могут содержать только символы: a–z, A–Z, 0–9, и _ (подчеркивание).
- Имена переменных не должны включать в себя пробелы.

  Если имя переменной нужно составить более чем из одного слова, то в качестве разделителя следует использовать символ подчеркивания (например, `$user_name`).
- Имена переменных чувствительны к регистру символов. Переменная `$High_Score` отличается от переменной `$high_score`.

## Операторы

Операторы — это математические, строковые, логические команды и команды сравнения, такие как плюс, минус, умножить и разделить. Код PHP во многом похож на обычные арифметические записи. Например, в результате работы следующего оператора выводится число 8:

```
echo 6 + 2;
```

Перед тем как приступить к изучению возможностей PHP, следует уделить немного внимания изучению предоставляющих эти возможности различных операторов.

### Арифметические операторы

Арифметические операторы проделывают вполне ожидаемую работу. Они применяются для выполнения математических операций. Их можно использовать для проведения четырех основных операций (сложения, вычитания, умножения и деления), а также для нахождения модуля (остатка от деления) и увеличения или уменьшения значения на единицу (табл. 3.1).

**Таблица 3.1.** Арифметические операторы

| Оператор | Описание | Пример |
|---|---|---|
| + | Сложение | $j + 1 |
| - | Вычитание | $j – 6 |
| * | Умножение | $j * 11 |
| / | Деление | $j / 4 |
| % | Модуль (остаток от деления) | $j % 9 |
| ++ | Инкремент (приращение) | ++$j |
| -- | Декремент (отрицательное приращение) | --$j |

## Операторы присваивания

Эти операторы используются для присваивания значений переменным. К ним относится самый простой оператор =, а также операторы +=, -= и т. д. (табл. 3.2). Оператор += вместо полного замещения находящегося слева значения добавляет к нему значение, которое находится справа от него. Итак, если переменная $count имела начальное значение 5, то оператор

```
$count += 1;
```

устанавливает значение $count равным 6 точно так же, как более привычный оператор присваивания:

```
$count = $count + 1;
```

**Таблица 3.2.** Операторы присваивания

| Оператор | Пример | Эквивалент |
|---|---|---|
| = | $j = 15 | $j = 15 |
| += | $j += 5 | $j = $j + 5 |
| -= | $j -= 3 | $j = $j – 3 |
| *= | $j *= 8 | $j = $j * 8 |
| /= | $j /= 16 | $j = $j / 16 |
| .= | $j .= $k | $j = $j . $k |
| %= | $j %= 4 | $j = $j % 4 |

У строк есть собственный оператор, точка (.), который будет более подробно рассмотрен в пункте «Объединение строк» далее.

## Операторы сравнения

Как правило, операторы сравнения используются внутри таких конструкций, как инструкция `if`, в которых требуется сравнивать значения двух элементов. Например, если необходимо узнать, не достигло ли значение переменной, подвергающееся приращению, какого-то конкретного значения или не превышает ли значение другой переменной установленного значения и т. д. (табл. 3.3).

**Таблица 3.3.** Операторы сравнения

| Оператор | Описание | Пример |
|---|---|---|
| == | Равно | $j == 4 |
| != | Не равно | $j != 21 |
| > | Больше чем | $j > 3 |
| < | Меньше чем | $j < 100 |
| >= | Больше чем или равно | $j >= 15 |
| <= | Меньше чем или равно | $j <= 8 |

Учтите, что операторы = и == предназначены для разных действий. Если первый из них является оператором присваивания, то второй — оператором сравнения. Иногда в спешке даже более опытные программисты могут вместо одного из них поставить другой, поэтому будьте внимательны, используя эти операторы.

## Логические операторы

Если логические операторы вам раньше не встречались, то поначалу они могут показаться чем-то необычным. Нужно представить, что вы делаете логические заключения на простом разговорном языке. Например, можно сказать самому себе: «Если время уже больше 12, но меньше 14 часов, значит, нужно пообедать». В PHP код для такого высказывания может выглядеть следующим образом:

```
if ($hour > 12 && $hour < 14) dolunch();
```

Здесь набор инструкций для самого обеда помещен в функцию по имени `dolunch`, которую позже нужно будет создать. В этой инструкции отсутствует элемент `then` (тогда), поскольку его присутствие само собой разумеется.

Как видно из предыдущего примера, логический оператор обычно используется для объединения результатов работы двух операторов сравнения, показанных в предыдущем разделе. Результат работы одного логического оператора может служить входным значением для другого логического оператора («Если время уже больше 12, но меньше 14 часов, или же если в комнате пахнет жареным и тарелки уже стоят на столе...»). Как правило, если какое-то действие имеет истинное или ложное значение — `TRUE` или `FALSE`, оно может служить входным значением для логического оператора, который берет два истинных или ложных входных значения и выдает в качестве результата истинное или ложное значение. Логические операторы показаны в табл. 3.4.

**Таблица 3.4.** Логические операторы

| Оператор | Описание | Пример |
|---|---|---|
| && | И | $j == 3 && $k == 2 |
| and | Низкоприоритетное И | $j == 3 and $k == 2 |
| \|\| | ИЛИ | $j < 5 \|\| $j > 10 |
| or | Низкоприоритетное ИЛИ | $j < 5 or $j > 10 |
| ! | НЕ | ! ($j == $k) |
| xor | Исключающее ИЛИ | $j xor $k |

Заметьте, что оператор && обычно взаимозаменяем с оператором and; то же самое справедливо и для операторов || и or. Но у операторов and и or более низкий приоритет, поэтому в некоторых случаях, для того чтобы принудительно расставить приоритеты, могут понадобиться дополнительные круглые скобки. В то же время бывают случаи, когда применимы только операторы and или or, как в следующем предложении, использующем оператор or (объяснения даны в главе 10):

```
mysql_select_db($database) or die("Невозможно выбрать базу данных");
```

Наиболее непривычным из этих операторов является xor, предназначенный для операции *исключающего ИЛИ*, который возвращает истинное значение TRUE, если любое из входных значений истинно, и возвращает ложное значение FALSE, если оба они имеют значение TRUE или FALSE. Чтобы понять его работу, представьте, что хотите изобрести чистящее средство для дома. Как аммиак (ammonia), так и хлорка (bleach) обладают хорошими чистящими свойствами, поэтому нужно, чтобы ваше средство содержало одно из этих веществ. Но оба они не могут в нем присутствовать, поскольку их сочетание опасно. В PHP это можно представить в следующем виде:

```
$ingredient = $ammonia xor $bleach;
```

В представленном фрагменте если любая из двух переменных, $ammonia или $bleach, имеет значение TRUE, то значение переменной $ingredient также будет установлено в TRUE. Но если обе они имеют значение TRUE или значение FALSE, значение переменной $ingredient будет установлено в FALSE.

# Присваивание значений переменным

Синтаксис присваивания значения переменной всегда имеет вид *переменная = значение*. Для передачи значения другой переменной он имеет немного иной вид *другая_переменная = переменная*.

Есть еще несколько дополнительных операторов присваивания, которые могут оказаться полезными. Например, нам уже встречался оператор:

```
$x += 10;
```

который предписывает PHP-парсеру добавить значение, расположенное справа от него (в данном случае это значение равно 10), к значению переменной $x. Подобным образом можно вычесть значение:

```
$y -= 10;
```

## Увеличение и уменьшение значения переменной на единицу

Добавление или вычитание единицы является настолько часто встречающейся операцией, что PHP предоставляет для этого специальные операторы. Вместо операторов += и -= можно воспользоваться одним из следующих операторов:

```
++$x;
--$y;
```

В сочетании с проверкой (инструкцией if) можно воспользоваться следующим кодом:

```
if (++$x == 10) echo $x;
```

Этот код предписывает PHP *сначала* увеличить значение переменной $x на единицу, а затем проверить, не имеет ли она значение 10; если переменная имеет такое значение, его следует вывести на экран. Можно также потребовать от PHP увеличить значение переменной на единицу (или, как в следующем примере, уменьшить на единицу) *после* того, как ее значение будет проверено:

```
if ($y-- == 0) echo $y;
```

что дает несколько иной результат. Предположим, что первоначальное значение переменной $y до выполнения оператора было равно нулю. Операция сравнения вернет результат TRUE, но после того, как она будет проведена, значение переменной $y будет установлено в –1. Тогда что же отобразит инструкция echo: 0 или –1? Попробуйте догадаться, а потом, чтобы подтвердить свою догадку, испытайте работу инструкции в PHP-процессоре. Поскольку такая комбинация операторов может вас запутать, ее можно применять только в качестве обучающего примера, но ни в коем случае не рассматривать в качестве приемлемого стиля программирования.

Короче говоря, когда именно увеличено или уменьшено на единицу значение переменной, до или после проверки, зависит от того, где помещен оператор инкремента или декремента, перед именем переменной или после него.

Кстати, правильный ответ на предыдущий вопрос таков: инструкция echo отобразит результат –1, потому что значение переменной $y было уменьшено на единицу сразу же после того, как к ней получила доступ инструкция if, и до того, как к ней получила доступ инструкция echo.

## Объединение строк

При объединении строк, когда к одной строке символов добавляется другая строка, используется символ точки (.). Самый простой способ объединения строк выглядит следующим образом:

```
echo "У вас " . $msgs . " сообщений.";
```

Если предположить, что переменной $msgs присвоено значение 5, то эта строка кода выведет следующую информацию:

```
У вас 5 сообщений.
```

Так же, как с помощью оператора += можно добавить значение к числовой переменной, с помощью оператора .= можно добавить одну строку к другой:

```
$bulletin .= $newsflash;
```

В данном случае, если в переменной $bulletin содержится сводка новостей, а в переменной $newsflash — экстренное сообщение, команда добавляет это сообщение к сводке новостей и теперь переменная $bulletin включает в себя обе строки текста.

## Типы строк

В PHP поддерживаются два типа строк, которые обозначаются типом используемых кавычек. Если требуется присвоить переменной значение текстовой строки, сохраняя ее точное содержимое, нужно воспользоваться одинарными кавычками (апострофами):

```
$info = 'Предваряйте имена переменных символом $, как в данном примере: $variable';
```

В данном случае переменной `$info` присваивается каждый символ, находящийся внутри строки в одинарных кавычках. Если воспользоваться двойными кавычками, то PHP попытается вычислить `$variable` и получить значение переменной.

В то же время, если *требуется* включить в состав строки значение переменной, используется строка, заключенная в двойные кавычки:

```
echo "За всю историю было $count президентов США ";
```

Из этого следует, что данный синтаксис предлагает более простую форму объединения, в которой для добавления одной строки к другой не нужно использовать символ точки или закрывать и снова открывать кавычки. Этот прием называется *подстановкой переменной*. Можно заметить, что в некоторых приложениях он используется довольно часто, а в других не применяется вообще.

## Изменение предназначения символов

Иногда в строке должны содержаться символы, имеющие специальное предназначение, которые могут быть неправильно интерпретированы. Например, следующая строка кода не будет работать, потому что вторая кавычка (апостроф), встреченная в слове `sister's`, укажет PHP-парсеру на то, что достигнут конец строки. Следовательно, вся остальная часть строки будет отвергнута как ошибочная:

```
$text = 'My sister's car is a Ford'; // Ошибочный синтаксис
```

Чтобы исправить ошибку, нужно непосредственно перед вызывающим неоднозначное толкование символом кавычки добавить обратный слеш, чтобы заставить PHP рассматривать этот символ буквально и не подвергать его интерпретации:

```
$text = 'My sister\'s car is a Ford';
```

Этот прием можно применить практически во всех ситуациях, где в противном случае PHP вернул бы ошибку, пытаясь интерпретировать символ. Например, следующая строка, заключенная в двойные кавычки, будет присвоена переменной без ошибок:

```
$text = "My Mother always said \"Eat your greens\".";
```

Кроме того, для вставки в строку различных специальных символов, например табуляции, новой строки и возврата каретки, могут применяться управляющие символы соответственно `\t`, `\n` и `\r`. Вот пример, в котором символы табуляции используются для разметки заголовка (они включены в строку исключительно для иллюстрации применения символа обратного слеша, поскольку существуют более подходящие способы разметки веб-страниц):

```
$heading = "Дата\tИмя\tПлатеж";
```

Эти специальные символы, предваряемые символами обратного слеша, работают только в строках, заключенных в двойные кавычки. Если заключить предыдущую строку в одинарные кавычки, то вместо символов табуляции в ней будут отображены нелепые последовательности символов \t. Внутри строк, заключенных в одинарные кавычки, в качестве символов с измененным предназначением распознаются только измененный апостроф (\') и сам измененный обратный слеш (\\).

## Многострочные команды

Иногда нужно вывести из PHP большой объем текста, а использование нескольких инструкций echo (или print), заняло бы много времени и было неразумным. PHP предлагает два удобных средства, предназначенных для того, чтобы справиться с подобной ситуацией. Первое из них состоит в заключении в кавычки нескольких строк, как в примере 3.6. Переменным также можно присвоить значения способом, показанным в примере 3.7.

**Пример 3.6.** Инструкция echo, использующая несколько строк

```
<?php
$author = "Альфред Ньюмен";

echo "Это заголовок

Это первая строка.
Это вторая строка.
Автор $author.";
?>
```

**Пример 3.7.** Многострочное присваивание

```
<?php
$author = "Альфред Ньюмен";

$text = "Это заголовок

Это первая строка.
Это вторая строка.
Автор $author.";
?>
```

В PHP можно также воспользоваться многострочной последовательностью, используя оператор <<<, который обычно называют *here-document* («здесь документ») или, для краткости, *heredoc*. Он представляет собой способ указания строкового литерала, сохраняющего в тексте обрывы строк и другие пустые пространства (включая отступы). Его использование показано в примере 3.8.

**Пример 3.8.** Еще один вариант инструкции echo, использующей сразу несколько строк

```
<?php
$author = "Альфред Ньюмен";

echo <<<_END
```

```
Это заголовок

Это первая строка.
Это вторая строка.
- Автор $author.
_END;
?>
```

Этот код предписывает PHP вывести все, что находится между двумя тегами _END, как будто все это является строкой, заключенной в двойные кавычки. Это означает, что разработчику можно, например, написать целый раздел HTML-кода прямо в коде PHP, а затем заменить конкретные динамические части переменными PHP.

Важно запомнить, что закрывающий тег _END; *должен* появляться строго в начале новой строки и он должен быть *единственным* содержимым этой строки — к ней не разрешается добавлять даже комментарии (нельзя ставить даже одиночный пробел). Как только многострочный блок закрыт, можно снова воспользоваться тем же самым именем тега.

Запомните: используя heredoc-конструкцию <<<_END..._END;, вы не должны добавлять символы \n, чтобы отправить команду на перевод строки, достаточно просто нажать клавишу Enter и приступить к набору новой строки. В отличие от других строк, заключенных в одинарные или двойные кавычки, внутри конструкции heredoc можно по своему усмотрению совершенно свободно пользоваться всеми одинарными или двойными кавычками, не изменяя их первоначального предназначения с помощью обратного слеша (\).

В примере 3.9 показано, как использовать этот же синтаксис для присваивания переменной многострочного значения.

**Пример 3.9.** Присваивание переменной многострочного значения

```
<?php
$author = "Альфред Ньюмен";

$out = <<<_END
Это заголовок

Это первая строка.
Это вторая строка.
- Автор $author.
_END;
?>
```

После этого переменная $out будет наполнена содержимым, размещенным между двумя тегами. Если не присваивать, а добавлять значение, то для добавления строки к значению переменной $out вместо оператора = можно воспользоваться оператором .=.

Будьте внимательны, не ставьте точку с запятой сразу же за первым тегом _END, поскольку она прервет многострочный блок еще до его начала и вызовет сообщение об ошибке синтаксического разбора — Parse error. Точку с запятой нужно ставить

только после закрывающего тега _END, хотя внутри блока можно свободно пользоваться точкой с запятой как обычным текстовым символом.

Кстати, тег _END — лишь один из многих, я выбрал его для этих примеров, поскольку его использование где-нибудь еще в коде PHP маловероятно. Вы можете использовать по собственному усмотрению любой тег, например _SECTION1 или _OUTPUT и т. д. И еще, чтобы отличать подобные теги от переменных или функций, обычно в начале их имени ставят знак подчеркивания; но если не хотите, можете им не пользоваться.

Многострочную разметку текста можно рассматривать как удобное средство, упрощающее чтение вашего кода PHP, поскольку как только текст отображается на веб-странице, вступают в силу правила форматирования HTML и пустые пространства скрываются (но имя переменной $author по-прежнему заменяется ее значением в соответствии с правилами вставки значений переменных).

Так, например, если загрузить эти примеры многострочного вывода в браузер, они не будут отображены в виде нескольких строк, потому что все браузеры рассматривают символы новой строки просто как пробелы. Но если воспользоваться свойством браузера, позволяющим просматривать исходный код, обнаружится, что все символы новой строки правильно расставлены и вывод появляется на нескольких строках.

## Типы переменных

PHP относится к очень слабо типизированным языкам. Это значит, что переменные не требуют объявления перед своим использованием и что PHP всегда преобразует переменные в тот тип, который требуется для их окружения на момент доступа к ним.

Например, можно создать число, состоящее из нескольких цифр, и извлечь из него *n*-ю цифру, просто предположив, что это число является строкой. В следующем фрагменте кода (пример 3.10) перемножаются числа 12 345 и 67 890 и возвращается результат 838 102 050, который затем помещается в переменную $number.

**Пример 3.10.** Автоматическое преобразование числа в строку

```
<?php
$number = 12345 * 67890;
echo substr($number, 3, 1);
?>
```

Когда присваивается значение, $number является числовой переменной. Но во второй строке кода вызов значения этой переменной помещен в PHP-функцию substr, которая должна вернуть из переменной $number один символ, стоящий на четвертой позиции (не забывайте, что в PHP отсчет позиции начинается с нуля). Для выполнения этой задачи PHP превращает $number в строку, состоящую из девяти символов, чтобы функция substr могла получить к ней доступ и вернуть символ, в данном случае 1.

То же самое происходит при необходимости превратить строку в число и т. д. А в примере 3.11 переменной $pi присвоено строковое значение, которое затем

в третьей строке кода автоматически превращается в число с плавающей точкой, чтобы стать частью уравнения по вычислению площади круга, которое выводит значение 78,5398175.

**Пример 3.11.** Автоматическое преобразование строки в число

```
<?php
$pi = "3.1415927";
$radius = 5;
echo $pi * ($radius * $radius);
?>
```

На практике все это означает, что вам не стоит слишком волноваться за типы переменных. Им следует просто присвоить значения, имеющие для вас смысл, и PHP при необходимости их преобразует. Затем, если понадобится извлечь значение, их нужно просто запросить, например, с помощью инструкции echo.

# Константы

*Константы*, как и переменные, хранят информацию для последующего доступа, за исключением того, что они оправдывают свое название констант (постоянных). Иными словами, после определения констант их значения устанавливаются для всей остальной программы и не могут быть изменены.

К примеру, константа может использоваться для хранения местоположения корневого каталога вашего сервера (папки, содержащей основные файлы вашего веб-сайта). Определить такую константу можно следующим образом:

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Затем для чтения содержимого константы нужно просто сослаться на нее, как на обычную переменную (но не предваряя ее имя знаком доллара):

```
$directory = ROOT_LOCATION;
```

Теперь, как только понадобится запустить ваш PHP-код на другом сервере с другой конфигурацией папок, нужно будет изменить только одну строку кода.

Нужно помнить о двух основных особенностях констант: перед их именами *не нужно* ставить символ $ (как перед именами обычных переменных) и их можно определить только с помощью функции define.

По общепринятому соглашению считается правилом хорошего тона использовать в именах констант буквы только верхнего регистра, особенно если ваш код будет также читать кто-нибудь другой.

## Предопределенные константы

PHP поставляется в виде готового продукта, с десятками предопределенных констант, которые редко используют такие новички, как вы. Тем не менее существуют константы, известные как *волшебные*, которые могут оказаться для вас полезными с самого начала. У имен волшебных констант в начале и в конце всегда стоят два

символа подчеркивания, чтобы нельзя было случайно назвать одну из собственных констант уже занятым под эти константы именем. Подробности о волшебных константах приведены в табл. 3.5. Понятия, упомянутые в таблице, будут раскрыты в следующих главах.

**Таблица 3.5.** Волшебные константы PHP

| Волшебная константа | Описание |
|---|---|
| __LINE__ | Номер текущей строки в файле |
| __FILE__ | Полное путевое имя файла. Если используется внутри инструкции include, то возвращается имя включенного файла. В версии PHP 4.0.2, __FILE__ всегда содержит абсолютный путь с раскрытыми символическими ссылками, а в предыдущих версиях при определенных обстоятельствах она может содержать относительный путь |
| __DIR__ | Каталог файла. Если используется внутри инструкции include, возвращается каталог включенного файла. Такой же результат дает применение функции dirname(__FILE__). В этом имени каталога отсутствует замыкающий слеш, если только этот каталог не является корневым. (Добавлена в PHP 5.3.0) |
| __FUNCTION__ | Имя функции. Начиная с PHP 5, возвращает имя функции, под которым она была объявлена (с учетом регистра символов). В PHP 4 возвращаемое значение всегда составлено из символов нижнего регистра. (Добавлена в PHP 4.3.0) |
| __CLASS__ | Имя класса. Начиная с PHP 5, возвращает имя класса, под которым он был объявлен (с учетом регистра символов). В PHP 4 возвращаемое значение всегда составлено из символов нижнего регистра. (Добавлена в PHP 4.3.0) |
| __METHOD__ | Имя метода класса. Возвращает имя метода, под которым он был объявлен (с учетом регистра символов). (Добавлена в PHP 5.0.0) |
| __NAMESPACE__ | Имя текущего пространства имен (с учетом регистра символов). Эта константа определена во время компиляции. (Добавлена в PHP 5.3.0) |

Эти константы полезны при отладке, когда нужно вставить строку кода, чтобы понять, до какого места дошло выполнение программы:

```
echo "Это строка " . __LINE__ . " в файле " . __FILE__;
```

Эта команда выведет в веб-браузер текущую строку программы с указанием текущего файла, исполняемого в данный момент (включая путь к нему).

## Различие между командами echo и print

Нам уже встречались разнообразные способы использования команды `echo` для вывода текста с сервера в браузер. В одних случаях выводится строковый литерал, в других сначала происходило объединение строк или вычисление значений переменных. Был также показан вывод, распространяющийся на несколько строк.

Но команде `echo` есть альтернатива, которой также можно воспользоваться: команда `print`. Эти две команды очень похожи друг на друга, но `print` — это функция, воспринимающая единственный параметр, а `echo` — конструкция языка PHP.

В общем, команда echo работает при выводе обычного текста быстрее print, поскольку не является функцией и не устанавливает возвращаемое значение.

С другой стороны, поскольку она не является функцией, ее, в отличие от print, нельзя использовать как часть более сложного выражения. В следующем примере для вывода информации о том, является ли значение переменной истинным (TRUE) или ложным (FALSE), используется функция print, но сделать то же самое с помощью команды echo не представляется возможным, поскольку она выведет на экран сообщение об ошибке синтаксического разбора — Parse error:

```
$b ? print "TRUE" : print "FALSE";
```

Вопросительный знак является простым способом задать вопрос о том, какое значение имеет переменная $b, истинное или ложное. Команда, которая располагается слева от двоеточия, выполняется в том случае, если $b имеет истинное значение, а команда, которая располагается справа, выполняется, если $b имеет ложное значение.

Тем не менее в приводимых здесь примерах чаще всего используется команда echo, и я рекомендую применять именно ее до тех пор, пока вам при PHP-разработке реально не потребуется задействовать функцию print.

## Функции

Функции используются для выделения блоков кода, выполняющих конкретную задачу. Например, если вам часто приходится искать какие-то данные и выводить их в определенном формате, то вполне разумно будет обратиться к функции. Код, выполняющий эту задачу, может занимать всего три строки, но пока вы не воспользуетесь функцией, необходимость вставлять этот код в программу десятки раз делает ее неоправданно большой и сложной. А если вы чуть позже захотите изменить формат вывода данных, помещение кода в функцию будет означать, что вам придется внести изменения только в одном месте программы.

Помещение кода в функцию не только сокращает размер исходного кода и делает его более удобным для чтения, но и дает дополнительные функциональные возможности (эта игра слов носит преднамеренный характер), поскольку функциям могут передаваться параметры, которые вносят изменения в характер их работы. Функции также могут возвращать значения вызывающему их коду.

Чтобы создать функцию, нужно ее объявить, как показано в примере 3.12.

**Пример 3.12.** Простое объявление функции

```
<?php
function longdate($timestamp)
{
    return date("l F jS Y", $timestamp);
}
?>
```

Эта функция использует в качестве входных данных отметку времени системы Unix (целое число, отображающее дату и время на основе количества секунд, прошедших с нуля часов 1 января 1970 года), а затем вызывает PHP-функцию date

с нужным форматом строки, чтобы вернуть дату в формате «Понедельник август 1 2016». Между стоящими после имени функции круглыми скобками может размещаться любое количество параметров, но для этой функции выбран прием только одного параметра. Весь код, который выполняется при последующем вызове функции, заключается в фигурные скобки.

Чтобы с помощью этой функции вывести сегодняшнюю дату, нужно поместить в свой код следующий вызов:

```
echo longdate(time());
```

В этом вызове для извлечения текущей отметки времени Unix и передачи ее только что созданной функции `longdate`, которая затем возвращает для отображения соответствующую строку команде `echo`, используется встроенная PHP-функция `time`. Если требуется вывести дату семнадцатидневной давности, нужно сделать следующий вызов:

```
echo longdate(time() - 17 * 24 * 60 * 60);
```

в котором функции `longdate` передается текущая отметка времени Unix, уменьшенная на количество секунд, которое прошло за 17 дней (17 дней × 24 ч × 60 мин × 60 с).

Функции могут также воспринимать несколько параметров и возвращать несколько результатов, используя технологию, которая будет рассмотрена в следующих главах.

# Область видимости переменной

Если программа очень длинная, то с подбором подходящих имен переменных могут возникнуть трудности, но, программируя на PHP, можно определить *область видимости* переменной. Иными словами, можно, к примеру, указать, что переменная `$temp` будет использоваться только внутри конкретной функции, чтобы забыть о том, что она после возврата из кода функции применяется где-нибудь еще. Фактически именно такой в PHP является по умолчанию область видимости переменных.

В качестве альтернативы можно проинформировать PHP о том, что переменная имеет глобальную область видимости и доступ к ней может быть осуществлен из любого места программы.

## Локальные переменные

*Локальные переменные* создаются внутри функции, и к ним имеется доступ только из кода этой функции. Обычно это временные переменные, которые используются до выхода из функции для хранения частично обработанных результатов.

Одним из наборов локальных переменных является перечень аргументов функции. В предыдущем разделе была определена функция, воспринимающая параметр по имени `$timestamp`. Значение этого параметра действительно только в теле функции, за пределами этой функции его значение нельзя ни получить, ни установить.

Чтобы привести еще один пример локальной переменной, рассмотрим функцию `longdate` еще раз в немного измененном варианте (пример 3.13).

**Пример 3.13.** Расширенная версия функции longdate

```
<?php
function longdate($timestamp)
{
    $temp = date("l F jS Y", $timestamp);
    return "Дата: $temp";
}
?>
```

В этом примере значение, возвращенное функцией date, присваивается временной переменной $temp, которая затем вставляется в строку, возвращаемую определяемой функцией. Как только будет осуществлен выход из функции, значение переменной $temp удаляется, как будто она вообще никогда не использовалась.

Теперь, чтобы посмотреть на области видимости переменных в действии, изучим похожий код, показанный в примере 3.14. Здесь переменная $temp была создана *еще до* вызова функции longdate.

**Пример 3.14.** Неудачная попытка получить доступ к переменной $temp в функции longdate

```
<?php
$temp = "Дата: ";
echo longdate(time());

function longdate($timestamp)
{
    return $temp . date("l F jS Y", $timestamp);
}
?>
```

Поскольку переменная $temp не была создана внутри функции longdate, а также не была передана ей в качестве параметра, функция longdate не может получить к ней доступ. Поэтому этот фрагмент кода выведет только дату без предшествующего ей текста. На самом деле сначала будет отображено сообщение об ошибке, предупреждающее об использовании неопределенной переменной (Notice: Undefined variable: temp).

Причина в том, что по умолчанию переменные, созданные внутри функции, являются локальными для этой функции, а переменные, созданные за пределами любой функции, могут быть доступны только из того кода, который не входит в код ни одной из функций.

В примерах 3.15 и 3.16 показан ряд способов исправления кода, приведенного в примере 3.14.

**Пример 3.15.** Решить проблему можно путем переноса ссылки на переменную $temp в ее локальную область видимости

```
<?php
$temp = "Дата: ";
echo $temp . longdate(time());

function longdate($timestamp)
```

```
{
    return date("l F jS Y", $timestamp);
}
?>
```

В примере 3.15 ссылка на $temp перемещена за пределы функции. Эта ссылка появляется в той же области видимости, в которой была определена переменная.

**Пример 3.16.** Альтернативное решение: передача $temp в качестве аргумента

```
<?php
$temp = "Дата: ";
echo longdate($temp, time());

function longdate($text, $timestamp)
{
    return $text . date("l F jS Y", $timestamp);
}
?>
```

В примере 3.16 принято другое решение: передать значение переменной $temp функции longdate в качестве дополнительного аргумента. Функция longdate считывает это значение во временную переменную, которую она создает под именем $text, и выводит желаемый результат.

Программисты часто допускают ошибку, забывая об области видимости переменных, поэтому, если не помнить принципы ее работы, это поможет в отладке некоторых весьма неочевидных ошибок программного кода. Достаточно сказать, что, если вы не объявили переменную каким-нибудь особым образом, ее область видимости ограничена локальным пространством: либо в пределах кода текущей функции, либо в пределах кода, не принадлежащего никаким функциям, в зависимости от того, где эта переменная была впервые создана или где к ней впервые был получен доступ, внутри функции или за ее пределами.

## Глобальные переменные

Бывают случаи, когда требуется переменная, имеющая *глобальную* область видимости, поскольку нужно, чтобы к ней имелся доступ из всего кода программы. К тому же некоторые данные могут быть настолько объемными и сложными, что их не захочется передавать функциям в качестве аргументов.

Чтобы объявить переменную, имеющую глобальную область видимости, используется ключевое слово global. Предположим, что существует некий способ входа пользователей на ваш веб-сайт, и нужно, чтобы весь код программы знал, с кем он имеет дело — с зарегистрированным пользователем или с гостем. Один из способов решения этой задачи заключается в создании глобальной переменной $is_logged_in:

```
global $is_logged_in;
```

Теперь вашей функции входа в систему нужно лишь при удачной попытке входа на сайт присвоить этой переменной значение 1, а при неудачной попытке — значение 0. Поскольку переменная обладает глобальной областью видимости, доступ к ней может быть получен из любой строки кода вашей программы.

Но пользоваться глобальными переменными нужно с оглядкой. Я рекомендую создавать их только в том случае, если без них совершенно невозможно добиться нужного результата. Вообще-то программы, разбитые на небольшие фрагменты и отдельные данные, содержат меньше ошибок и проще в обслуживании. Если ваша программа состоит из нескольких тысяч строк кода (а когда-нибудь такое случится) и оказалось, что где-то глобальная переменная имеет неверное значение, то сколько времени уйдет на поиски кода, который присваивает ей это значение?

Кроме того, если используется слишком много глобальных переменных, возникает риск воспользоваться одним из их имен еще раз в локальном пространстве или, по крайней мере, полагать, что такая переменная применяется локально, хотя на самом деле она уже была объявлена в качестве глобальной. Из таких вот ситуаций и возникают разные непонятные ошибки.

Иногда я придерживаюсь соглашения о написании имен всех глобальных переменных в верхнем регистре (что совпадает с рекомендациями о написании в этом же регистре имен констант), чтобы можно было с первого взгляда определить область видимости переменной.

## Статические переменные

В пункте «Локальные переменные» выше было упомянуто, что значение переменной стирается сразу же после выхода из функции. Если функция вызывается многократно, она начинает свою работу со свежей копией переменной и ее прежние установки не имеют никакого значения.

Интересно, а что, если внутри функции есть такая локальная переменная, к которой не должно быть доступа из других частей программы, но значение которой желательно сохранять до следующего вызова функции? Зачем? Возможно, потому, что нужен некий счетчик, чтобы следить за количеством вызовов функции. Решение, показанное в примере 3.17, заключается в объявлении *статической переменной*.

**Пример 3.17.** Функция, использующая статическую переменную

```
<?php
function test()
{
    static $count = 0;
    echo $count;
    $count++;
}
?>
```

В этом примере в самой первой строке функции создается статическая переменная по имени `$count`, которой присваивается нулевое начальное значение. В следующей строке выводится значение переменной, а в последней строке это значение увеличивается на единицу.

При следующем вызове функции, поскольку переменная `$count` уже была объявлена, первая строка функции пропускается и до нового увеличения значения переменной `$count` отображается ее предыдущее значение.

Планируя использование статических переменных, следует учесть, что при их определении присвоить им результат какого-нибудь выражения невозможно. Они могут инициализироваться только предопределенными значениями (пример 3.18).

**Пример 3.18.** Допустимые и недопустимые объявления статических переменных

```
<?php
static $int = 0;          Допустимо
static $int = 1+2;        Недопустимо (вызовет ошибку синтаксического
                          разбора (Parse error))
static $int = sqrt(144);  Недопустимо
?>
```

## Суперглобальные переменные

Начиная с версии PHP 4.1.0, стали доступны некоторые предопределенные переменные. Они известны как *суперглобальные переменные*. Смысл этого названия заключается в том, что они предоставляются средой окружения PHP и имеют глобальную область видимости внутри программы, то есть доступны абсолютно из любого ее места.

В этих суперглобальных переменных содержится масса полезной информации о текущей работающей программе и ее окружении (табл. 3.6). Эти переменные имеют структуру ассоциативных массивов, которые будут рассмотрены в главе 6.

**Таблица 3.6.** Суперглобальные переменные PHP

| Имя суперглобальной переменной | Ее содержимое |
|---|---|
| $GLOBALS | Все переменные, которые на данный момент определены в глобальной области видимости сценария. Имена переменных служат ключами массива |
| $_SERVER | Информация о заголовках, путях, местах расположения сценариев. Элементы этого массива создаются веб-сервером, и это не дает гарантии, что каждый веб-сервер будет предоставлять какую-то часть информации или ее всю |
| $_GET | Переменные, которые передаются текущему сценарию методом HTTP GET |
| $_POST | Переменные, которые передаются текущему сценарию методом HTTP POST |
| $_FILES | Элементы, подгруженные к текущему сценарию методом HTTP POST |
| $_COOKIE | Переменные, переданные текущему сценарию посредством HTTP cookies |
| $_SESSION | Переменные сессии, доступные текущему сценарию |
| $_REQUEST | Содержимое информации, переданной от браузера; по умолчанию $_GET, $_POST и $_COOKIE |
| $_ENV | Переменные, переданные текущему сценарию методом environment |

В именах всех суперглобальных переменных (кроме первой) присутствует один знак подчеркивания и используются только заглавные буквы, поэтому, чтобы избежать путаницы, не следует присваивать своим переменным имена, оформленные в таком же стиле.

Для иллюстрации порядка применения суперглобальных переменных рассмотрим часть той информации, которая может быть использована сайтами. Среди многой другой интересной информации, предоставляемой суперглобальными переменными, есть и URL-адрес той страницы, с которой пользователь был перенаправлен на текущую веб-страницу. Эта информация может быть получена следующим образом:

```
$came_from = $_SERVER['HTTP_REFERRER'];
```

Как видите, ничего сложного. Если же пользователь зашел непосредственно на вашу страницу, к примеру набрав ее URL-адрес непосредственно в браузере, переменной `$came_from` будет присвоена пустая строка.

### Суперглобальные переменные и проблемы безопасности

Обратите внимание, что суперглобальные переменные часто используются злоумышленниками, пытающимися отыскать средства для атаки и вмешательства в работу вашего веб-сайта. Они загружают в `$_POST`, `$_GET` или в другие суперглобальные переменные вредоносный код, например команды Unix или MySQL, которые, если вы по незнанию к ним обратитесь, могут разрушить или отобразить незащищенные данные.

Именно поэтому перед применением суперглобальных переменных их всегда следует подвергать предварительной обработке. Для этого можно воспользоваться PHP-функцией `htmlentities`. Она занимается преобразованием всех символов в элементы HTML. Например, символы «меньше чем» и «больше чем» (< и >) превращаются в строки `<` и `>`, то же самое делается для перевода в безопасное состояние всех кавычек, обратных слешей и т. д.

Поэтому более подходящий способ доступа к `$_SERVER` (и другим суперглобальным переменным) выглядит следующим образом:

```
$came_from = htmlentities($_SERVER['HTTP_REFERRER']);
```

В этой главе были заложены надежные основы, необходимые для работы с PHP. В главе 4 мы приступим к практическому использованию изученного материала для построения выражений и управления ходом программы, иными словами, перейдем к реальному программированию.

Но перед изучением новой главы я рекомендую проверить свои знания, ответив на приведенные далее вопросы, чтобы убедиться в том, что вы полностью усвоили содержимое этой главы.

## Проверьте ваши знания

1. Какой тег PHP служит основанием для того, чтобы приступить к интерпретации программного кода? Как выглядит краткая форма этого тега?
2. Какие два вида тегов используются для добавления комментариев?
3. Какой символ должен стоять в конце каждой инструкции PHP?
4. Какой символ используется в начале имен всех переменных PHP?

5. Что может храниться в переменных?
6. В чем разница между выражениями `$variable = 1` и `$variable == 1`?
7. Как вы считаете, почему подчеркивание разрешено использовать в именах переменных (`$current_user`), а дефисы — нет (`$current-user`)?
8. Чувствительны ли имена переменных к регистру букв?
9. Можно ли в именах переменных использовать пробелы?
10. Как преобразовать значение одного типа переменной в значение другого типа (скажем, строку в число)?
11. В чем разница между `++$j` и `$j++`?
12. Являются ли взаимозаменяемыми операторы `&&` и `and`?
13. Как создается многострочный вывод: с использованием команды `echo` или присвоением многострочного значения?
14. Можно ли переопределить константу?
15. Как изменить исходное предназначение кавычки?
16. В чем разница между командами `echo` и `print`?
17. Каково назначение функций?
18. Как сделать переменную доступной для всего кода PHP-программы?
19. Какими двумя способами можно передать всей остальной программе данные, которые были созданы внутри функции?
20. Что является результатом объединения строки с числом?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 3».

# 4 Выражения и управление процессом выполнения программы в PHP

В предыдущей главе уже упоминались некоторые темы, которые более полно будут рассмотрены в данной главе, например выбор (ветвление) и создание сложных выражений. В главе 3 мне хотелось сконцентрировать внимание на наиболее общих вопросах синтаксиса и работы в PHP, но при этом невозможно было не затронуть темы более высокого уровня. А вот теперь можно преподнести вам основы, необходимые для полноценного использования всех сильных сторон PHP.

В этой главе будет заложен фундамент практики программирования на PHP и рассмотрены основные способы управления процессом выполнения программы.

## Выражения

Начнем с базовой части любого языка программирования — *выражения*.

Выражение представляет собой сочетание значений, переменных, операторов и функций, в результате вычисления которого выдается новое значение. Оно знакомо всем, кто когда-либо имел дело с обыкновенной школьной алгеброй:

```
y = 3(abs(2x) + 4)
```

что в PHP приобретает следующий вид:

```
$y = 3 * (abs(2*$x) + 4);
```

Возвращаемое значение (в данном случае y или $y) может быть числом, строкой или булевым значением (названным так в честь Джорджа Буля, английского математика и философа XIX века). Первые два типа значений вам уже должны быть знакомы, потому я объясню, что такое третий тип.

Элементарное булево значение может быть либо истинным — TRUE, либо ложным — FALSE. Например, выражение 20 > 9 (20 больше 9) является истинным (TRUE), а выражение 5 == 6 (5 равно 6) — ложным (FALSE). (Булевы, или логические, операции могут быть объединены путем использования таких операторов, как И, ИЛИ

и исключающее ИЛИ, то есть AND, OR и XOR, которые будут рассмотрены в этой главе.)

Обратите внимание на то, что для имен TRUE и FALSE я использую буквы верхнего регистра. Это обусловлено тем, что в PHP они являются предопределенными константами. При желании можно также применять и их версии, составленные из букв нижнего регистра, поскольку они также являются предопределенными константами. Кстати, версия, в которой задействуются буквы нижнего регистра, является более надежной, потому что PHP не допускает ее переопределения, а версия, использующая буквы верхнего регистра, может быть переопределена, и это нужно иметь в виду при импортировании чужого кода.

В примере 4.1 показаны некоторые простые выражения: два, о которых уже упоминалось, плюс еще два выражения. Для каждой строки выводится буква от a до d, за которой следуют двоеточие и результат выражения (тег <br /> используется в HTML для переноса и разбивает выходную информацию на четыре строки).

**Пример 4.1.** Четыре простых булева выражения

```
<?php
echo "a: [" . (20 > 9) . "]<br />";
echo "b: [" . (5 == 6) . "]<br />";
echo "c: [" . (1 == 0) . "]<br />";
echo "d: [" . (1 == 1) . "]<br />";
?>
```

Этот код выведет следующую информацию:

```
a: [1]
b: []
c: []
d: [1]
```

Обратите внимание на то, что результаты вычисления обоих выражений, a: и d:, являются истинными (TRUE), имеющими значение 1. А результаты вычисления выражений b: и c: ложны (FALSE) и вообще не показывают никакого значения, поскольку в PHP константа FALSE определена как NULL (ничто). Чтобы убедиться в этом, можно ввести код, показанный в примере 4.2.

**Пример 4.2.** Вывод значений TRUE и FALSE

```
<?php // test2.php
echo "a: ["   TRUE    "]<br />";
echo "b: [" . FALSE . "]<br />";
?>
```

Этот код выведет следующую информацию:

```
a: [1]
b: []
```

Кстати, в некоторых языках константа FALSE может быть определена как 0 или даже как -1, поэтому в каждом языке ее определение стоит проверить.

# Литералы и переменные

Простейшей формой выражения является *литерал*, означающий нечто, вычисляющееся само в себя, например число 73 или строка Hello. Выражение может также быть просто переменной, которая вычисляется в присвоенное этой переменной значение. Обе эти формы относятся к типам выражений, поскольку они возвращают значение.

В примере 4.3 показаны пять различных литералов, все они возвращают значения, хотя и разных типов.

**Пример 4.3.** Пять типов литералов

```
<?php
$myname = "Brian";
$myage = 37;
echo "a: " . 73         "<br />"; // Числовой литерал
echo "b: "   "Hello"    "<br />"; // Строковый литерал
echo "c: " . FALSE      "<br />"; // Литерал константы
echo "d: " . $myname .  "<br />"; // Литерал строковой переменной
echo "e: " . $myage  .  "<br />"; // Литерал числовой переменной
?>
```

Как и ожидалось, в выходной информации вы увидите возвращаемое значение всех этих выражений, за исключением выражения `c:`, результатом вычисления которого является `FALSE` и которое ничего не возвращает:

```
a: 73
b: Hello
c:
d: Brian
e: 37
```

Объединив простейшие выражения с операторами, можно создать более сложные выражения, итогом вычисления которых являются какие-нибудь полезные результаты.

При объединении присваивания или управляющей конструкции с выражениями получается *инструкция*. В примере 4.4 показано по одной инструкции каждого вида. В первой из них осуществляется присваивание результата выражения `366 - $day_number` переменной `$days_to_new_year`, а во второй выводится приветственное сообщение, если выражение `$days_to_new_year < 30` вычисляется как `TRUE`.

**Пример 4.4.** Выражение и инструкция

```
<?php
$days_to_new_year = 366 - $day_number;  // Выражение
if ($days_to_new_year < 30)
{
    echo "Скоро Новый Год!";            // Инструкция
}
?>
```

# Операторы

В PHP имеется множество мощных операторов, от арифметических, строковых и логических до операторов присваивания, сравнения и многих других операторов (табл. 4.1).

**Таблица 4.1.** Типы операторов PHP

| Оператор | Описание | Пример |
|---|---|---|
| Арифметический | Элементарная математика | $a + $b |
| Для работы с массивом | Слияние массивов | $a + $b |
| Присваивания | Присваивание значений | $a = $b + 23 |
| Поразрядный | Манипуляция битами в байте | 12 ^ 9 |
| Сравнения | Сравнение двух значений | $a < $b |
| Выполнения | Выполнение содержимого, заключенного в обратные кавычки | 'ls -al' |
| Инкремента/декремента | Добавление или вычитание единицы | $a++ |
| Логический | Выполнение булевых сравнений | $a and $b |
| Строковый | Объединение строк | $a . $b |

Различные типы операторов воспринимают разное количество операндов.

- *Унарные* операторы, такие как оператор инкремента (`$a++`) или изменения знака числа (`-$a`), воспринимают только один операнд.
- *Бинарные* операторы, представленные большим количеством операторов PHP, включая операторы сложения, вычитания, умножения и деления, воспринимают два операнда.
- Один *трехкомпонентный* оператор, имеющий форму `x ? y : z`. По сути, это состоящая из трех частей однострочная инструкция `if`, в которой осуществляется выбор между двумя выражениями, зависящий от результата вычисления третьего выражения. Этот условный оператор воспринимает три операнда.

# Приоритетность операторов

Если бы у всех операторов был один и тот же уровень приоритета, то они обрабатывались бы в том порядке, в котором встречались интерпретатору. Фактически многие операторы имеют одинаковый уровень приоритета, что и показано в примере 4.5.

**Пример 4.5.** Три эквивалентных выражения

```
1 + 2 + 3 - 4 + 5
2 - 4 + 5 + 3 + 1
5 + 2 - 4 + 1 + 3
```

Из примера видно, что, несмотря на перестановку чисел (и предшествующих им операторов), результат каждого выражения имеет значение 7, поскольку у операторов «плюс» и «минус» одинаковый уровень приоритета. Можно проделать то же самое с операторами умножения и деления (пример 4.6).

**Пример 4.6.** Три выражения, которые также являются эквивалентными

```
1 * 2 * 3 / 4 * 5
2 / 4 * 5 * 3 * 1
5 * 2 / 4 * 1 * 3
```

В этом примере получаемое значение всегда равно 7,5. Но все меняется, когда в выражении присутствуют операторы с *разными* уровнями приоритета, как в примере 4.7.

**Пример 4.7.** Три выражения, в которых присутствуют операторы с разными уровнями приоритета

```
1 + 2 * 3 - 4 * 5
2 - 4 * 5 * 3 + 1
5 + 2 - 4 + 1 * 3
```

Если бы не существовало приоритетности операторов, то в результате вычисления этих выражений получались бы числа 25, –29 и 12 соответственно. Но поскольку операторы умножения и деления имеют более высокий уровень приоритета по сравнению с операторами сложения и вычитания, вокруг частей выражения с их участием предполагается наличие скобок, и если их сделать видимыми, выражения будут выглядеть так, как показано в примере 4.8.

**Пример 4.8.** Три выражения, в которых отображены предполагаемые скобки

```
1 + (2 * 3) - (4 * 5)
2 - (4 * 5 * 3) + 1
5 + 2 - 4 + (1 * 3)
```

Очевидно, что PHP должен сначала вычислить подвыражения, заключенные в скобки, чтобы получились частично вычисленные выражения, показанные в примере 4.9.

**Пример 4.9.** Выражения после вычисления подвыражений в скобках

```
1 + (6) - (20)
2 - (60) + 1
5 + 2 - 4 + (3)
```

Окончательный результат вычисления этих выражений равен соответственно –13, –57 и 6 (что абсолютно отличается от результатов 25, –29 и 12, которые мы увидели бы при отсутствии приоритетности операторов).

Разумеется, исходную приоритетность операторов можно отменить, расставив собственные скобки, и принудительно получить результаты, показанные в самом начале, которые были бы получены в отсутствие приоритетности операторов (пример 4.10).

**Пример 4.10.** Принудительное выполнение вычислений справа налево

```
((1 + 2) * 3 - 4) * 5
(2 - 4) * 5 * 3 + 1
(5 + 2 - 4 + 1) * 3
```

Теперь, если скобки расставлены правильно, мы увидим значения 25, –29 и 12 соответственно.

В табл. 4.2 перечислены операторы PHP в порядке их приоритетности от самого высокого до самого низкого уровня.

**Таблица 4.2.** Операторы PHP, расположенные по уровню их приоритетности (сверху вниз)

| Оператор(ы) | Тип |
|---|---|
| () | Скобки |
| ++ -- | Инкремент/декремент |
| ! | Логический |
| * / % | Арифметические |
| + - | Арифметические и строковые |
| << >> | Побитовые |
| < <= > >= <> | Сравнения |
| == != === !== | Сравнения |
| & | Поразрядный (и ссылочный) |
| ^ | Поразрядный |
| \| | Поразрядный |
| && | Логический |
| \|\| | Логический |
| ? : | Трехкомпонентный |
| = += -= *= /= .= %= &= != ^= <<= >>= | Присваивания |
| and | Логический |
| xor | Логический |
| or | Логический |

# Взаимосвязанность операторов

Мы рассматривали обработку выражений слева направо, за исключением тех случаев, в которых вступала в силу приоритетность операторов. Но некоторые операторы могут также потребовать обработки справа налево. Направление обработки обусловливается *взаимосвязанностью* операторов.

Взаимосвязанность приобретает большое значение в тех случаях, когда вы явным образом не меняете приоритетности. В табл. 4.3 перечислены все операторы, имеющие взаимосвязанность справа налево.

**Таблица 4.3.** Операторы, имеющие взаимосвязанность справа налево

| Оператор | Описание |
|---|---|
| NEW | Создание нового объекта |
| ! | Логическое НЕ |
| ~ | Поразрядное НЕ |
| ++ -- | Инкремент и декремент |

*Продолжение* ⇩

**Таблица 4.3** *(продолжение)*

| Оператор | Описание |
|---|---|
| + - | Унарный плюс и изменение знака числа |
| (int) | Преобразование в целое число |
| (double) | Преобразование в число с плавающей точкой |
| (string) | Преобразование в строковое значение |
| (array) | Преобразование в массив |
| (object) | Преобразование в объект |
| @ | Подавление сообщения об ошибке |
| = | Присваивание |

Рассмотрим оператор присваивания, показанный в примере 4.11, где всем трем переменным присваивается значение 0.

**Пример 4.11.** Оператор множественного присваивания

```
<?php
$level = $score = $time = 0;
?>
```

Такое множественное присваивание возможно только в том случае, если сначала вычисляется самая правая часть выражения, а затем процесс продолжается справа налево.

Новичкам следует научиться в процессе работы с PHP избегать потенциальных просчетов в вопросах взаимосвязанности операторов и всегда принудительно задавать порядок вычислений, заключая подвыражения в круглые скобки. Это поможет и другим программистам, которые будут обслуживать ваш код, понять, что в нем происходит.

# Операторы отношения

*Операторы отношения* проверяют значения двух операндов и возвращают логический результат, равный либо TRUE, либо FALSE. Существует три типа операторов отношения: операторы *равенства*, *сравнения* и *логические операторы*.

## Операторы равенства

С оператором равенства == (двойным знаком равенства) мы уже не раз встречались в этой книге. Его не следует путать с оператором присваивания = (одинарным знаком равенства). В примере 4.12 первый оператор присваивает значение, а второй проверяет его на равенство.

**Пример 4.12.** Присваивание значения и проверка его на равенство

```
<?php
$month = "Март";
if ($month == "Март") echo "Весна наступила";
?>
```

Как видно из примера, возвращая значение TRUE или FALSE, оператор сравнения позволяет проверять условия, используя инструкцию if. Но это еще не все, поскольку PHP является языком со слабой типизацией. Если два операнда выражения равенства имеют разные типы, PHP преобразует их к тому типу, который имеет для него наибольший смысл.

К примеру, любые строки, составленные полностью из цифр, при сравнении с числами будут преобразованы в числа. В примере 4.13 переменные $a и $b являются двумя разными строками, и поэтому вряд ли стоило ожидать, что какая-то из инструкций if выведет результат.

**Пример 4.13.** Операторы равенства и тождественности

```
<?php
$a = "1000";
$b = "+1000";
if ($a == $b) echo "1";
if ($a === $b) echo "2";
?>
```

Но если запустить этот пример, то он выведет число, что означает, что результат вычисления первой инструкции if является TRUE. Причина в том, что обе строки сначала конвертируются в числа, и 1000 имеет такое же числовое значение, что и +1000.

В отличие от первой, во второй инструкции if используется оператор тождественности — тройной знак равенства, который удерживает PHP от автоматического преобразования типов. Поэтому переменные $a и $b сравниваются как строки и теперь считаются отличающимися друг от друга, поэтому на экран ничего не выводится.

Как и в случае с принудительным заданием уровня приоритетности операторов, если возникнут сомнения в том, будет ли PHP конвертировать типы операндов, для отмены такого поведения интерпретатора можно воспользоваться оператором тождественности.

Аналогично применению оператора равенства, для определения равенства операндов можно проверить их на неравенство, используя оператор *неравенства* !=. Пример 4.14 является измененным примером 4.13, в котором операторы равенства и тождественности были заменены противоположными им операторами.

**Пример 4.14.** Операторы неравенства и нетождественности

```
<?php
$a = "1000";
$b = "+1000";
if ($a != $b) echo "1";
if ($a !== $b) echo "2";
?>
```

Как, наверное, и ожидалось, первая инструкция if не выводит на экран число 1, потому что в коде ставится вопрос о неравенстве числовых значений переменных $a и $b. Вместо этого будет выведено число 2, поскольку вторая инструкция if ставит вопрос о нетождественности прежнего типа операндов переменных $a и $b, и ответом будет TRUE, потому что они не тождественны.

## Операторы сравнения

Используя операторы сравнения, можно расширить круг проверок, не ограничивая его только равенством и неравенством. PHP предоставляет вам для этого операторы > (больше), < (меньше), >= (больше или равно) и <= (меньше или равно). В примере 4.15 показано использование этих операторов.

**Пример 4.15.** Четыре оператора сравнения

```
<?php
$a = 2; $b = 3;
if ($a > $b)  echo "$a больше $b<br />";
if ($a < $b)  echo "$a меньше $b<br />";
if ($a >= $b) echo "$a больше или равно $b<br />";
if ($a <= $b) echo "$a меньше или равно $b<br />";
?>
```

Этот пример, в котором переменная $a имеет значение 2, а переменная $b — значение 3, выведет на экран следующую информацию:

```
2 меньше 3
2 меньше или равно 3
```

Попробуйте самостоятельно запустить этот пример, меняя значения переменных $a и $b, чтобы увидеть результаты. Присвойте им одинаковые значения и посмотрите, что из этого получится.

## Логические операторы

Логические операторы выдают истинные или ложные результаты, поэтому их также называют булевыми операторами. Всего имеется четыре таких оператора (табл. 4.4).

**Таблица 4.4.** Логические операторы

| Логический оператор | Описание |
|---|---|
| AND | Возвращает истинное значение (TRUE), если оба операнда имеют истинные значения |
| OR | Возвращает истинное значение (TRUE), если любой из операндов имеет истинное значение |
| XOR | Возвращает истинное значение (TRUE), если один из двух операндов имеет истинное значение |
| NOT | Возвращает истинное значение (TRUE), если операнд имеет ложное значение, или ложное значение (FALSE), если он имеет истинное значение |

Использование этих операторов показано в примере 4.16. Обратите внимание, что PHP требует использовать вместо слова NOT символ !. Кроме того, операторы могут быть составлены из букв нижнего или верхнего регистра.

**Пример 4.16.** Использование логических операторов

```
<?php
$a = 1; $b = 0;
echo ($a AND $b)   "<br />";
echo ($a or $b)    "<br />";
```

```
echo ($a XOR $b) . "<br />";
echo !$a         "<br />";
?>
```

Этот пример выводит на экран NULL, 1, 1, NULL. Это значит, что только вторая и третья инструкции echo получают в результате вычисления значение TRUE. (Следует помнить, что NULL, или ничто, отображает значение FALSE.) Такой результат получается, потому что оператору AND, чтобы вернуть значение TRUE, нужно, чтобы оба операнда имели истинное значение, а четвертый оператор проводит над значением переменной $a операцию NOT, превращая его из TRUE (значения, равного единице) в FALSE. Если есть желание поэкспериментировать, запустите этот код, присваивая переменным $a и $b разные значения, выбранные из 1 и 0.

Занимаясь программированием, следует помнить, что у операторов AND и OR более низкий уровень приоритета, чем у других версий этих операторов — && и ||. Поэтому в сложных выражениях более безопасным будет, наверное, применение операторов && и ||.

Использование в инструкции if оператора OR может стать причиной непредвиденных проблем, поскольку второй операнд не будет вычисляться, если в результате вычисления первого операнда уже получено значение TRUE. В примере 4.17 функция getnext никогда не будет вызвана, если переменная $finished имеет значение 1.

**Пример 4.17.** Инструкция, использующая оператор OR

```
<?php
if ($finished == 1 OR getnext() == 1) exit;
?>
```

Если нужно, чтобы функция getnext вызывалась для каждой инструкции if, следует внести в код изменения, показанные в примере 4.18.

**Пример 4.18.** Изменения в инструкции if ... OR, гарантирующие вызов функции getnext

```
<?php
$gn = getnext();
if ($finished == 1 OR $gn == 1) exit;
?>
```

В этом случае код в функции getnext будет выполнен и возвращенное значение будет сохранено в переменной $gn еще до выполнения инструкции if.

Другое решение заключается в том, чтобы обеспечить выполнение функции getnext за счет простой перестановки условий местами, поскольку тогда вызов функции будет появляться в выражении первым.

В табл. 4.5 показаны все допустимые варианты использования логических операторов. Следует заметить, что !TRUE является эквивалентом FALSE, а !FALSE — эквивалентом TRUE.

**Таблица 4.5.** Все логические выражения, допустимые в PHP

| Входные данные | | Операторы и результаты | | |
|---|---|---|---|---|
| a | b | AND | OR | XOR |
| TRUE | TRUE | TRUE | TRUE | FALSE |
| TRUE | FALSE | FALSE | TRUE | TRUE |
| FALSE | TRUE | FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE | FALSE | FALSE |

# Условия

Условия изменяют процесс выполнения программы. Они позволяют задавать конкретные вопросы и по-разному реагировать на полученные ответы. Условия играют важную роль при разработке динамических веб-страниц — основной цели использования PHP, поскольку они облегчают создание разных вариантов выводимой при каждом просмотре веб-страницы информации.

Существует три типа нециклических условных инструкций: `if`, `switch` и `?`. Называя их нециклическими, я имел в виду, что после действий, инициированных инструкцией, процесс выполнения программы продолжается, а при использовании циклических условных инструкций (которые еще предстоит рассмотреть) код выполняется снова и снова до тех пор, пока не будет соблюдено определенное условие.

# Инструкция if

Процесс выполнения программы можно представить себе как езду на машине по однополосной магистрали. Эта магистраль большей частью прямолинейна, но иногда встречаются различные дорожные знаки, задающие направление движения.

Когда встречается инструкция `if`, можно представить, что машина подошла к знаку объезда, предписаниям которого необходимо следовать, когда определенные условия вычисляются как `TRUE`. При этом вы съезжаете с магистрали и следуете по объездному пути до тех пор, пока не вернетесь снова на магистраль и не продолжите движение по исходному маршруту. Или же, если условие не вычисляется как `TRUE`, вы игнорируете объезд и продолжаете ехать по магистрали как ни в чем не бывало (рис. 4.1).

Содержимым условной инструкции `if` может быть любое допустимое PHP-выражение, включая равенство, сравнение, проверку на нуль и `NULL` и даже значения, возвращаемые функциями (как встроенными, так и созданными самостоятельно).

Действия, предпринимаемые при вычислении условия в `TRUE`, помещаются, как правило, в фигурные скобки { }. Но эти скобки можно опустить, если нужно выполнить всего одну инструкцию. Тем не менее, если всегда использовать фигурные скобки, можно избежать «охоты» на трудно отслеживаемые ошибки, возникающие, к примеру, когда к условной инструкции добавляется еще одна строка, но забывается о необходимости добавить фигурные скобки, из-за чего строка не вычисляет-

ся. (Учтите, что в целях экономии места и доходчивости материала, если в примерах, приводимых в книге, была всего одна исполняемая инструкция, я не следовал этому совету и опускал фигурные скобки.)

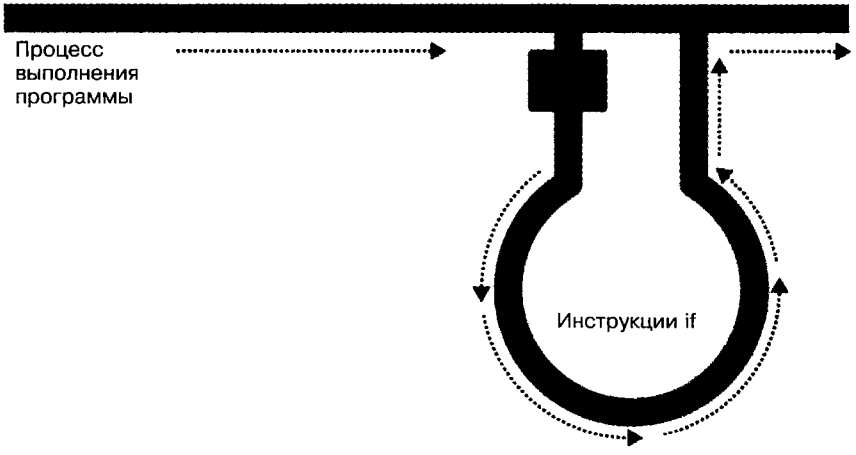



**Рис. 4.1.** Процесс выполнения программы похож на движение по однополосной магистрали

В примере 4.19 следует представить, что подошел конец месяца и нужно платить по всем счетам, поэтому вы проводите некоторые операции с банковским счетом.

**Пример 4.19.** Инструкция if, в которой используются фигурные скобки

```
<?php
if ($bank_balance < 100)
{
    $money        = 1000;
    $bank_balance += $money;
}
?>
```

В этом примере проверяется, не стал ли баланс ниже $100 (или 100 единиц другой используемой вами валюты). Если баланс стал ниже этой суммы, вы платите сами себе $1000, а затем прибавляете их к балансу. (Хорошо бы так просто зарабатывать деньги!)

Если баланс счета в банке равен $100 или превышает эту сумму, условные инструкции игнорируются и процесс выполнения программы переходит на следующую строку кода (которая здесь не показана).

Некоторые разработчики предпочитают ставить первую фигурную скобку справа от условного выражения, а некоторые начинают с нее новую строку. В этой книге открывающая фигурная скобка располагается обычно на новой строке. Подойдет любой из этих вариантов, поскольку PHP позволяет оставлять на ваше усмотрение какие угодно свободные пространства (пробелы, символы новых строк и табуляции). Но код будет легче читаться и отлаживаться, если у каждого уровня условий будет свой отступ, сформированный с помощью символа табуляции.

## Инструкция else

Бывают случаи, когда условие не вычисляется как TRUE, но вам не хочется сразу же продолжать выполнение основного кода программы, а вместо этого нужно сделать что-либо другое. В таком случае пригодится инструкция `else`. С ее помощью на вашей магистрали можно организовать второй объезд, показанный на рис. 4.2.

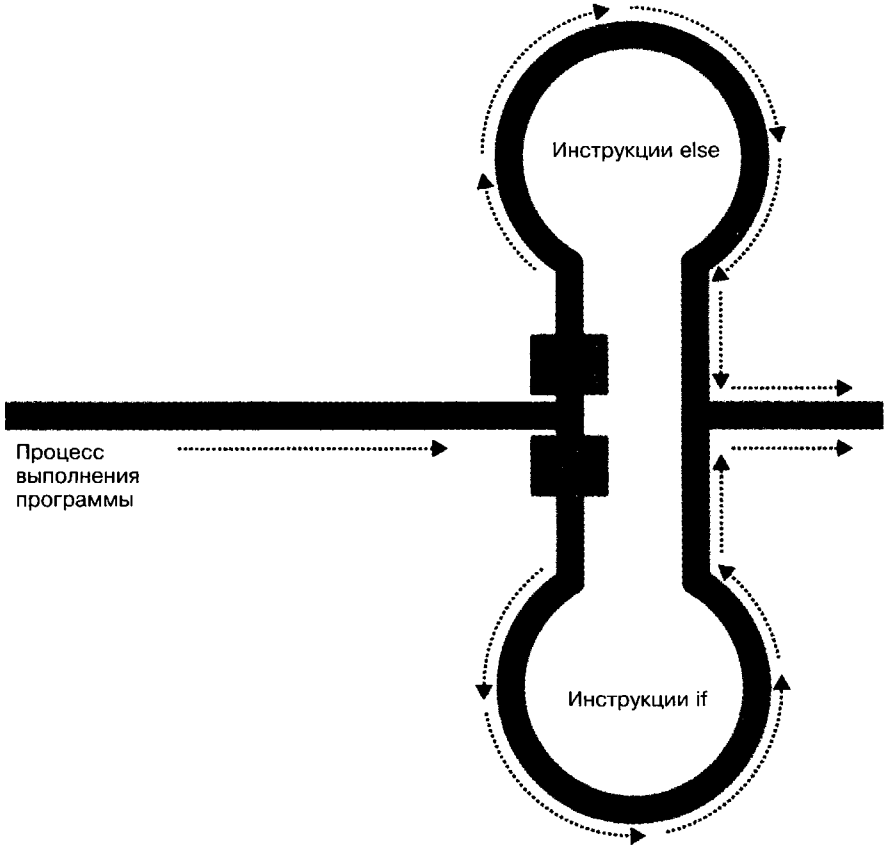



**Рис. 4.2.** Теперь у магистрали есть объезд if и объезд else

Если при использовании конструкции `if...else` условие вычисляется как TRUE, выполняется первая условная инструкция. Но если это условие вычисляется как FALSE, выполняется вторая условная инструкция. Для выполнения должна быть выбрана одна из этих двух инструкций, но обе сразу они не будут выполнены ни при каких условиях и обязательно будет выполнена хотя бы одна их них. Использование конструкции `if...else` показано в примере 4.20.

**Пример 4.20.** Конструкция `if...else`, в которой используются фигурные скобки

```
<?php
if ($bank_balance < 100)
```

```
{
    $money        = 1000;
    $bank_balance += $money;
}
else
{
    $savings      += 50;
    $bank_balance -= 50;
}
?>
```

Если в этом примере будет установлено, что в банке лежит более $100, выполняется инструкция else, с помощью которой часть этих денег перемещается на ваш сберегательный счет.

Точно так же, как и у if, если у инструкции else есть только одна условная инструкция, фигурные скобки можно не ставить. (Хотя фигурные скобки рекомендуется использовать в любом случае. Во-первых, при их наличии легче разобраться в коде, а во-вторых, они облегчают последующее добавление инструкций к этому ветвлению.)

## Инструкция elseif

Случается, что на основе последовательности условий нужно осуществить сразу несколько действий. Достичь желаемого результата можно, используя инструкцию elseif. Можно предположить, что она похожа на инструкцию else, за исключением того, что до кода условия вставляется еще одно условное выражение. Полноценная конструкция if...elseif...else показана в примере 4.21.

**Пример 4.21.** Конструкция if...elseif...else, в которой используются фигурные скобки

```
<?php
if ($bank_balance < 100)
{
    $money        = 1000;
    $bank_balance += $money;
}
elseif ($bank_balance > 200)
{
    $savings      += 100;
    $bank_balance -= 100;
}
else
{
    $savings      += 50;
    $bank_balance -= 50;
}
?>
```

В этом примере инструкция elseif была вставлена между инструкциями if и else. Она проверяет, не превышает ли баланс банковского счета сумму $200, и если превышает, принимается решение о том, что в этом месяце можно позволить себе положить на сберегательный счет $100.

Это все можно представить в виде набора объездов в нескольких направлениях (рис. 4.3).

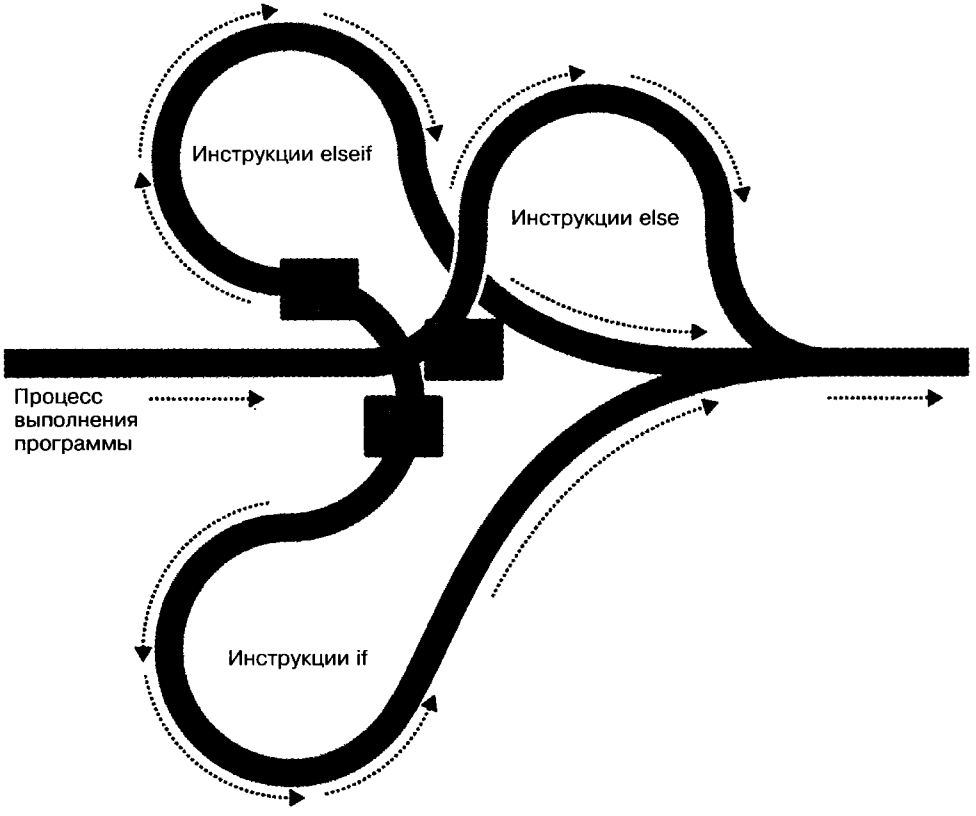



**Рис. 4.3.** Магистраль с объездами if, elseif и else

Инструкция else завершает либо конструкцию if...else, либо конструкцию if...elseif...else. Если она не нужна, финальную инструкцию else можно опустить, но ни одна из этих инструкций не должна стоять перед инструкцией elseif, точно так же как ни одна инструкция elseif не должна стоять перед инструкцией if.

Количество используемых инструкций `elseif` не ограничено. Но по мере роста количества этих инструкций будет лучше, наверное, обратиться к инструкции `switch`, если, конечно, она отвечает вашим запросам. Именно ее мы сейчас и рассмотрим.

## Инструкция switch

Инструкция `switch` применяется в тех случаях, когда у одной переменной или у результата вычисления выражения может быть несколько значений, каждое из кото-

Рассмотрим, например, управляемую кодом PHP систему меню, которая в соответствии с пожеланием пользователя передает отдельную строку коду основного меню. Предположим, что есть следующие варианты: Home, About, News, Login и Links, а переменная $page принимает одно из этих значений в соответствии с информацией, введенной пользователем.

Код реализации этого замысла с использованием конструкции if...elseif...else может иметь вид, показанный в примере 4.22.

**Пример 4.22.** Многострочная инструкция if...elseif

```
<?php
if     ($page == "Home")  echo "Вы выбрали Home";
elseif ($page == "About") echo "Вы выбрали About";
elseif ($page == "News")  echo "Вы выбрали News";
elseif ($page == "Login") echo "Вы выбрали Login";
elseif ($page == "Links") echo "Вы выбрали Links";
?>
```

Код, в котором используется инструкция switch, показан в примере 4.23.

**Пример 4.23.** Инструкция switch

```
<?php
switch ($page)
{
    case "Home":
   echo "Вы выбрали Home";
        break;
    case "About":
   echo "Вы выбрали About";
        break;
    case "News":
   echo "Вы выбрали News";
        break;
    case "Login":
   echo "Вы выбрали Login";
        break;
    case "Links":
   echo "Вы выбрали Links";
        break;
}
?>
```

Как видите, переменная $page используется только один раз — в начале инструкции switch. После этого все соответствия проверяются командой case. Когда найдено соответствие, выполняется его условная инструкция. Разумеется, в настоящей программе в этом месте будет применяться код отображения или перехода на страницу, а не простое сообщение пользователю о том, что именно он выбрал.

В инструкциях switch внутри команд case фигурные скобки не используются. Вместо этого инструкции начинаются с двоеточия и заканчиваются командой break. Тем не менее весь перечень команд case в инструкции switch заключается в фигурные скобки.

## Прекращение работы инструкции switch

Если нужно, чтобы инструкция switch прекратила свою работу из-за выполнения условия, используется команда break. Эта команда предписывает PHP прекратить работу инструкции switch и перейти к выполнению следующей инструкции.

Если в примере 4.23 не расставить команды break и результат вычисления команды case, проверяющей условие Home, получится TRUE, будут выполнены все пять условных инструкций, следующих за командами case. Или же, если переменная $page имела значение News, то, начиная с этого места, будут выполнены все оставшиеся команды case. Это сделано преднамеренно для расширения возможностей программирования, но в большинстве случаев не следует забывать ставить команду break во всех местах, где набор условных инструкций, следующих за командами case, завершает свою работу. Надо сказать, что случайный пропуск команд break является весьма распространенной ошибкой.

## Действие по умолчанию

Типичным требованием для инструкции switch является переход к действию по умолчанию, если не будет выполнено ни одно из условий, содержащихся в командах case. Например, к коду меню, показанному в примере 4.23, можно непосредственно перед закрывающей фигурной скобкой добавить код, показанный в примере 4.24.

**Пример 4.24.** Инструкция default для добавления к примеру 4.23

```
default: echo "Нераспознанный выбор";
         break;
```

Хотя здесь ставить команду break не требуется, поскольку default является заключительной внутренней инструкцией и процесс выполнения программы автоматически продолжится после закрывающей фигурной скобки, но если вы решите поставить инструкцию default выше этого места, ей определенно понадобится команда break, того чтобы процесс выполнения программы не затронул все стоящие ниже условные инструкции. Лучше перестраховаться и в конце этой инструкции всегда ставить команду break.

## Альтернативный синтаксис

Открывающую фигурную скобку инструкции switch можно заменить двоеточием, а закрывающую фигурную скобку — командой endswitch (пример 4.25).

Такой вариант используется довольно редко, и здесь он упоминается на тот случай, если придется столкнуться с ним в коде, созданном кем-нибудь другим.

**Пример 4.25.** Альтернативный синтаксис инструкции switch

```
<?php
switch ($page):
    case "Home":
        echo "Вы выбрали Home";
        break;

    // и т. д.

    case "Links":
```

```
        echo "Вы выбрали Links";
        break;
endswitch;
?>
```

## Оператор ?

Использование трехкомпонентного оператора ? позволяет избежать многословности инструкций `if` и `else`. Необычность этого оператора заключается в том, что он использует не два, как большинство других операторов, а три операнда.

В главе 3 уже состоялось краткое знакомство с этим оператором при выяснении разницы между `print` и `echo`, где он приводился в качестве примера оператора, который хорошо работает с `print`, но не работает с `echo`.

Оператору ? передаются выражение, которое он должен вычислить, и два выполняемых оператора: один для выполнения, когда результат вычисления выражения `TRUE`, а другой — когда `FALSE`.

В примере 4.26 показан код, который может использоваться для вывода предупреждения об уровне топлива в автомобиле на его панель приборов.

**Пример 4.26.** Использование оператора ?

```
<?php
echo $fuel <= 1 ? "Требуется дозаправка" : "Топлива еще достаточно";
?>
```

Если топлива остается всего 1 галлон[1] или меньше (иными словами, переменная `$fuel` имеет значение, равное единице или меньше ее), этот оператор возвращает предыдущей команде `echo` строку «Требуется дозаправка». В противном случае он возвращает строку «Топлива еще достаточно». Значение, возвращаемое оператором ?, можно также присвоить какой-нибудь переменной (пример 4.27).

**Пример 4.27.** Присваивание условного результата оператора ? переменной

```
<?php
$enough = $fuel <= 1 ? FALSE : TRUE;
?>
```

В этом примере переменной `$enough` будет присвоено значение `TRUE` только в том случае, если в баке более 1 галлона топлива, в противном случае ей будет присвоено значение `FALSE`.

Если вы считаете синтаксис оператора ? слишком запутанным, можете вместо него воспользоваться инструкцией `if`, но о нем все равно нужно знать, поскольку он может встретиться в программном коде, созданном другим программистом. Чтение кода, в котором используется этот оператор, может быть сильно затруднено из-за частого использования в нескольких местах одной и той же переменной. Например, весьма популярен код такого вида:

```
$saved = $saved >= $new ? $saved : $new;
```

---

[1] 1 галлон (американский) = 3,79 л. — *Примеч. ред.*

Понять, что он делает, можно только после тщательного разбора:

```
$saved =                  // Присваивание значения переменной $saved
       $saved >= $new     // Сравнение $saved и $new
    ?                     // Если сравнение выдает истинный результат ...
       $saved             // ... ей присваивается текущее значение $saved
                          // Если сравнение выдает ложный результат ...
       $new;              // ... ей присваивается значение переменной $new
```

Это весьма компактный способ отслеживания самого большого значения, которое может встретиться в процессе выполнения программы. Самое большое значение содержится в переменной `$saved` и при поступлении нового значения сравнивается со значением переменной `$new`. Программисты, освоившие оператор ?, считают, что для таких коротких сравнений его удобнее применять, чем инструкции `if`. Если этот оператор не используется для создания компактного кода, то он обычно применяется для принятия решений, умещающихся на одной строке, например для проверки того, установлено ли значение переменной, перед передачей ее функции.

# Организация циклов

Компьютеры славятся своей способностью быстро и неутомимо повторять вычисления. Зачастую от программы требуется снова и снова повторять одну и ту же последовательность кода, пока не произойдет какое-нибудь событие, например ввод значения пользователем или достижение программой своего естественного окончания. Имеющиеся в PHP разнообразные структуры организации циклов предоставляют великолепные способы решения подобных задач.

Чтобы представить, как это работает, посмотрите на рис. 4.4. Он очень похож на метафору с магистралью, которая использовалась для иллюстрации работы инструкции `if`, за исключением того, что у объезда также есть замкнутый участок, из которого машина может выйти только при соблюдении определенных программных условий.

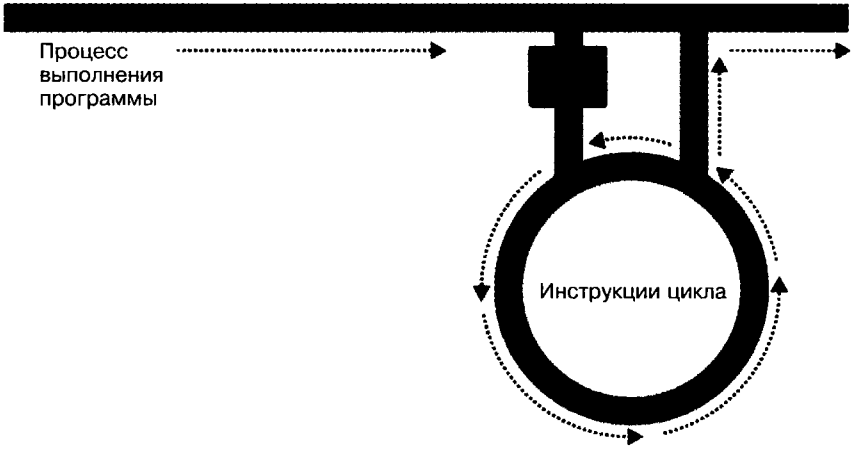



**Рис. 4.4.** Представление цикла как части программы магистральной разметки

## Циклы while

Давайте превратим автомобильную панель приборов из примера 4.26 в цикл, постоянно проверяющий уровень топлива при езде на машине, в котором используется инструкция цикла `while` (пример 4.28).

**Пример 4.28.** Цикл `while`

```
<?php
$fuel = 10;

while ($fuel > 1)
{
    // Продолжение поездки...
    echo "Топлива еще достаточно";
}
?>
```

Вообще-то вы можете предпочесть выводу текста горящий зеленый сигнал, но суть в том, что любая разновидность позитивной индикации об уровне топлива помещается в цикл `while`. Кстати, учтите, что если вы запустите этот пример на выполнение, то он будет постоянно выводить строку до тех пор, пока вы не остановите работу браузера.

Здесь, как и в случае с инструкциями if, для хранения инструкций внутри цикла while используются фигурные скобки, если только в этом цикле не задействована всего одна инструкция.

В примере 4.29 показан еще один вариант использования цикла `while`, в котором выводится таблица умножения на 12.

**Пример 4.29.** Цикл `while` для вывода таблицы умножения на 12

```
<?php
$count = 1;

while ($count <= 12)
{
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br />";
    ++$count;
}
?>
```

В этом примере переменной `$count` присваивается начальное значение 1, а затем запускается цикл `while`, в котором используется выражение сравнения `$count <= 12`. Этот цикл будет выполняться до тех пор, пока значение переменной не станет больше 12. Данный код выведет следующий текст:

```
Число 1, умноженное на 12, равно 12
Число 2, умноженное на 12, равно 24
Число 3, умноженное на 12, равно 36
```

*и т. д.*

Внутри цикла осуществляется вывод строки, а также значения переменной $count, умноженного на 12. Чтобы упорядочить вывод, после всего этого использован тег <br />, вызывающий переход на новую строку. Затем перед закрывающей фигурной скобкой, предписывающей PHP вернуться к началу цикла, значение переменной $count увеличивается на единицу.

Теперь значение переменной $count опять проверяется, чтобы узнать, не превышает ли оно число 12. Оно не превышает этого числа, но теперь оно равно 2, и после одиннадцати последующих прохождений цикла оно станет равно 13. Когда это произойдет, код, находящийся внутри цикла while, будет пропущен и станет выполняться код, следующий за циклом, в данном случае это будет завершение программы.

При отсутствии оператора ++$count (вместо которого с таким же успехом может быть применен оператор $count++) этот цикл будет похож на первый, показанный в этом разделе. Он никогда не закончится и будет снова и снова выводить один и тот же результат 1 × 12.

Но есть и более изящный способ написания этого цикла, который должен вам понравиться. Посмотрите на код примера 4.30.

**Пример 4.30.** Укороченная версия примера 4.29

```
<?php
$count = 0;
while (++$count <= 12)
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br />";
?>
```

В этом примере оператор ++$count был удален из тела цикла while и помещен непосредственно в выражение условия цикла. Теперь PHP вычисляет значение переменной $count в начале каждого прохода цикла (итерации) и, заметив, что перед именем переменной стоит оператор инкремента, сначала увеличивает значение переменной на 1 и только потом сравнивает его с числом 12. Следовательно, теперь переменной $count присваивается начальное значение 0, а не 1, поскольку это значение увеличивается сразу же, как только происходит вход в цикл. Если оставить начальное значение, равное 1, то будут выведены результаты для чисел между 2 и 12.

## Циклы do...while

Цикл do...while представляет собой небольшую модификацию цикла while, используемую в том случае, когда нужно, чтобы блок кода был исполнен хотя бы один раз, а условие проверялось только после этого.

В примере 4.31 показана модифицированная версия таблицы умножения на 12, в которой использован этот цикл.

**Пример 4.31.** Цикл do...while, используемый для вывода таблицы умножения на 12

```
<?php
$count = 1;
do
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br />";
while (++$count <= 12);
?>
```

Заметьте, что теперь мы вернулись к присваиванию переменной $count начального значения 1 (а не 0), потому что код выполняется сразу же, без увеличения значения переменной на 1. Во всем остальном этот код очень похож на показанный в примере 4.29.

Разумеется, если внутри цикла do...while находится несколько инструкций, то не следует забывать ставить вокруг них фигурные скобки, как показано в примере 4.32.

**Пример 4.32.** Расширенная версия примера 4.31, использующая фигурные скобки

```
<?php
$count = 1;
do {
    echo "Число $count, умноженное на 12, равно " . $count * 12;
    echo "<br />";
} while (++$count <= 12);
?>
```

## Циклы for

Цикл for, являющийся последней разновидностью инструкций цикла, к тому же еще и самый мощный из них, поскольку в нем сочетаются возможности установки значения переменных при входе в цикл, проверки соблюдения условия при каждом проходе цикла (итерации) и модификации значений переменных после каждой итерации.

В примере 4.33 продемонстрирована возможность вывода таблицы умножения с использованием цикла for.

**Пример 4.33.** Вывод таблицы умножения на 12 из цикла for

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
    echo "Число $count, умноженное на 12, равно " . $count * 12 . "<br />";
?>
```

Как видите, весь код сведен к одной инструкции for, в которой содержится одна условная инструкция. И вот что из этого получается. Каждая инструкция for воспринимает три параметра:

- выражение инициализации;
- выражение условия;
- выражение модификации.

Эти три выражения отделяются друг от друга точкой с запятой: for (выражение1 ; выражение2 ; выражение3). В начале первой итерации выполняется выражение инициализации. В нашем коде таблицы умножения переменная $count инициализируется значением 1. Затем при каждой итерации проверяется выражение условия (в данном случае $count <= 12), и выход из цикла осуществляется только в том случае, если результат вычисления условия будет TRUE. И наконец, в завершение каждой итерации выполняется выражение модификации. В случае с таблицей умножения значение переменной $count увеличивается на 1.

Эта структура в явном виде исключает любые требования по размещению управляющих элементов цикла в его собственном теле, освобождая его для инструкций, требующих циклического выполнения.

Если в теле цикла for содержится более одной инструкции, не забудьте воспользоваться фигурными скобками (пример 4.34).

**Пример 4.34.** Цикл for из примера 4.33 с добавлением фигурных скобок

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
{
    echo "Число $count, умноженное на 12, равно " . $count * 12;
    echo "<br />";
}
?>
```

Сравним условия, при которых следует использовать циклы for, с условиями, при которых следует использовать циклы while. Цикл for явно создавался под отдельное значение, изменяющееся на постоянную величину. Обычно мы имеем дело с увеличивающимся значением — это похоже на то, что вам был передан перечень того, что выбрал пользователь, и от вас требуется обработать каждый его выбор по очереди. Но переменную можно видоизменять по вашему усмотрению. Более сложная форма инструкции for позволяет даже осуществлять со всеми тремя параметрами сразу несколько операций:

```
for ($i = 1, $j = 1 ; $i + $j < 10 ; $i++ , $j++)
{
    // ...
}
```

Но новичкам использовать такую сложную форму не рекомендуется. Тут главное — отличать запятые от точки с запятой. Все три параметра должны быть отделены друг от друга точкой с запятой.

Несколько операторов внутри каждого параметра должны быть отделены друг от друга запятыми. Первый и третий параметры в предыдущем примере содержат по два оператора:

```
$i = 1, $j = 1   // Инициализация переменных $i и $j
$i + $j < 10     // Условие окончания работы цикла
$i++ , $j++      // Модификация $i и $j в конце каждой итерации
```

Главное, что следует уяснить из этого примера, — три секции параметров должны разделяться точкой с запятой, а не запятыми (которые должны использоваться только для разделения операторов внутри каждой секции параметров).

Тогда при каких условиях следует отдавать предпочтение инструкциям while перед инструкциями for? Когда ваше условие не зависит от простого изменения переменной на постоянной основе. Например, инструкция while применятся в том случае, если нужно проверить, не введено ли какое-то вполне определенное значение или не возникла ли какая-то конкретная ошибка, и завершить цикл сразу же, как только это произойдет.

## Прекращение работы цикла

Прекратить работу цикла for можно точно так же, как и работу рассмотренной уже инструкции switch, — используя команду break. К примеру, это может понадобиться, когда одна из ваших инструкций вернет ошибку и продолжать выполнение цикла станет небезопасно.

Один из таких случаев может произойти, когда при записи файла возникнет ошибка, возможно, из-за нехватки места на диске (пример 4.35).

**Пример 4.35.** Запись файла, использующая цикл for с перехватом ошибки

```
<?php
$fp = fopen("text.txt", 'wb');

for ($j = 0 ; $j < 100 ; ++$j)
{
    $written = fwrite($fp, "data");
    if ($written == FALSE) break;
}

fclose($fp);
?>
```

Это наиболее сложный из всех ранее приведенных фрагментов кода, но вы уже готовы к его пониманию. Команды обработки файлов будут рассмотрены в одной из следующих глав, а сейчас нужно лишь знать, что в первой строке кода открывается файл text.txt для записи в двоичном режиме, а затем переменной $fp возвращается указатель на него, который в дальнейшем используется для ссылки на этот открытый файл.

Затем осуществляется 100 проходов цикла (от 0 до 99), записывающих строку data в файл. После каждой записи функция fwrite присваивает переменной $written значение, представляющее собой количество успешно записанных символов. Но если происходит ошибка, функция fwrite присваивает этой переменной значение FALSE.

Поведение функции fwrite облегчает коду проверку переменной $written на наличие значения FALSE, и если она имеет такое значение, код прекращает работу цикла и передает управление инструкции, закрывающей файл.

При желании улучшить код можно упростить строку

```
if ($written == FALSE) break;
```

за счет использования оператора NOT:

```
if (!$written) break;
```

Фактически пара инструкций, находящихся внутри цикла, может быть сокращена до одной:

```
if (!fwrite($fp, "data")) break;
```

Но команда break обладает более широкими возможностями, чем можно было бы предположить, поскольку, если нужно прекратить работу кода, вложенного глубже

чем на один уровень, после команды break можно поставить число, показывающее, работу скольких уровней нужно прекратить, например:

```
break 2;
```

## Инструкция continue

Инструкция continue немного похожа на команду break, только она предписывает PHP остановить процесс текущего цикла и перейти непосредственно к его следующей итерации, то есть вместо прекращения работы всего цикла осуществляется выход только из текущей итерации.

Этот прием может пригодиться в тех случаях, когда известно, что нет смысла продолжать выполнение текущего цикла и нужно сберечь процессорное время или избежать ошибки путем перехода сразу к следующей итерации цикла. В примере 4.36 инструкция continue используется для того, чтобы избежать ошибки деления на ноль за счет ее вызова в тот момент, когда переменная $j имеет значение 0.

**Пример 4.36.** Перехват ошибки деления на ноль с помощью инструкции continue

```
<?php
$j = 10;

while ($j > -10)
{
  $j--;
  if ($j == 0) continue;
  echo (10 / $j) . "<br />";
}
>
```

Для всех значений переменной $j в диапазоне чисел между 10 и –10 за исключением 0 отображается результат деления числа 10 на значение переменной $j. Но для конкретного случая, когда значение $j равно 0, вызывается инструкция continue, и дальнейшее выполнение итерации сразу же пропускается с переходом к следующей итерации цикла.

# Неявное и явное преобразование типов

PHP является языком со слабой типизацией, который позволяет объявлять переменную и ее тип путем простого использования этой переменной. При необходимости он также осуществляет автоматическое преобразование одного типа в другой. Этот процесс называется *неявным преобразованием типов*.

Однако могут возникнуть ситуации, когда присущее PHP неявное преобразование типов станет совсем нежелательным действием. Рассматривая пример 4.37, обратите внимание на то, что входные данные для операции деления являются целыми числами. По умолчанию PHP осуществляет преобразование выходных данных к числу с плавающей точкой, чтобы получалось наиболее точное значение — 4,66 и 6 в периоде.

**Пример 4.37.** Этот пример возвращает число с плавающей точкой

```
<?php
$a = 56;
$b = 12;
$c = $a / $b;
echo $c;
?>
```

Но что делать, если вместо этого нужно получить значение переменной `$c` в виде целого числа? Этого можно добиться разными способами, одним из которых является принудительное преобразование результата `$a/$b` в целое число путем использования оператора преобразования `(int)`:

```
$c = (int) ($a / $b);
```

Этот способ называется *явным преобразованием типов*. Обратите внимание на то, что для обеспечения преобразования в целое число значения всего выражения это выражение помещено в круглые скобки. В противном случае преобразованию подверглось бы только значение переменной `$a`, что не имело бы никакого смысла, поскольку деление на значение переменной `$b` все равно вернуло бы результат в виде числа с плавающей точкой.

Можно провести явное преобразование значений в те типы, которые показаны в табл. 4.6, но обычно его можно избежать, используя преобразование за счет вызова одной из встроенных функций PHP. Например, для получения целочисленного значения можно использовать функцию `intval`. Этот раздел, как и многие другие в данной книге, предназначен в основном для того, чтобы помочь разобраться с чужим кодом, который может вам встретиться.

**Таблица 4.6.** Типы преобразований, доступных в PHP

| Тип преобразования | Описание |
|---|---|
| (int) (integer) | Преобразование в целое число путем отбрасывания десятичной части |
| (bool) (boolean) | Преобразование в булево значение |
| (float) (double) (real) | Преобразование в число с плавающей точкой |
| (string) | Преобразование в строку |
| (array) | Преобразование в массив |
| (object) | Преобразование в объект |

# Динамическое связывание в PHP

Поскольку PHP является языком программирования и получаемая в результате его работы выходная информация может быть совершенно разной для различных пользователей, есть возможность запускать целый веб-сайт с одной веб-страницы, созданной с помощью PHP. При каждом щелчке пользователя на каком-нибудь элементе подробности могут отправляться назад той же веб-странице, которая будет принимать решение, что делать дальше, в соответствии с различными объектами cookie и/или данными сессии, которые могут быть сохранены.

Но, несмотря на возможность создания таким способом целого веб-сайта, этого делать не рекомендуется, поскольку исходный код будет все время разрастаться и приобретет громадные размеры по мере того, как ему придется принимать во внимание разнообразные действия пользователя.

Будет куда более благоразумно разделить разработку веб-сайта на несколько разных частей. Например, один автономный процесс будет заниматься подпиской на веб-сайт со всеми вытекающими отсюда проверками допустимости адреса электронной почты, незадействованности имени пользователя и т. д.

Второй модуль неплохо было бы создать для регистрации пользователей, предшествующей их допуску к основной части вашего веб-сайта. Затем можно создать модуль вывода сообщений, в котором пользователи могли бы оставлять свои комментарии, модуль, содержащий ссылки и полезную информацию, еще один модуль, позволяющий загружать на сайт фотографии, и т. д.

Как только будет создано средство для отслеживания действий пользователя на вашем веб-сайте, использующее объекты cookie или переменные сессии (оба этих средства будут более подробно рассмотрены в следующих главах), можно разделить веб-сайт на удобные секции PHP-кода, каждая из которых будет независима от других. Таким образом, вы существенно облегчите себе будущую разработку каждого нового свойства и обслуживание уже имеющихся.

## Динамическое связывание в действии

Одним из наиболее популярных в настоящее время приложений, управляемых PHP, является платформа для ведения блогов WordPress (рис. 4.5). При ведении или чтении блога этого можно и не понять, но для каждой основной секции выделен свой основной PHP-файл, а огромное количество совместно используемых функций помещено в отдельные файлы, которые включаются основными PHP-страницами по мере необходимости.

Вся платформа держится на закулисном отслеживании сессии, поэтому вы вряд ли знаете о том, когда осуществляется переход от одной подчиненной секции к другой. Поэтому, если веб-разработчик хочет провести тонкую настройку WordPress, ему нетрудно найти конкретный файл, который для этого применяется, и выполнить его проверку и отладку, не теряя понапрасну времени на не связанные с ним части программы.

Когда в следующий раз будете использовать WordPress, проследите за адресной строкой своего веб-браузера, особенно при управлении блогом, и тогда вы сможете заметить обращения к некоторым из разнообразных PHP-файлов, которые используются в этом приложении.

В этой главе были рассмотрены обширные сведения, закладывающие основу для дальнейшего изучения материала книги. Теперь вы уже должны уметь составлять свои собственные небольшие PHP-программы. Но перед тем как перейти к следующей главе, посвященной функциям и объектам, можете проверить приобретенные знания, ответив на следующие вопросы.

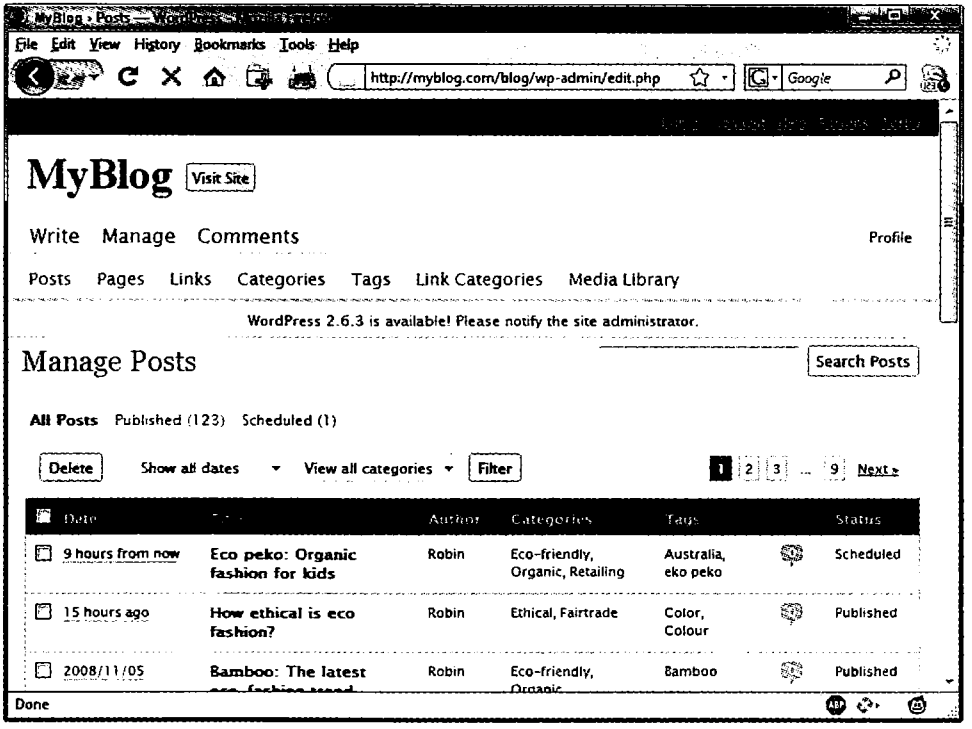



**Рис. 4.5.** Платформа WordPress, предназначенная для ведения блогов, написана на PHP

# Проверьте ваши знания

1. Какие основные значения представлены ключевыми словами TRUE и FALSE?
2. Что представляют собой две самые простые формы выражений?
3. В чем разница между унарными, бинарными и трехкомпонентными операторами?
4. В чем заключается наилучший способ установки собственной приоритетности операторов?
5. Что означает понятие взаимосвязанности операторов?
6. Когда следует использовать оператор идентичности (===)?
7. Назовите три типа условных инструкций.
8. Какую команду можно использовать для пропуска текущей итерации цикла и перехода к следующей итерации?
9. Почему цикл for считается более мощным, чем цикл while?
10. Как инструкции [illegible] интерпретируют условные выражения, составленные из разных типов данных?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 4».

# 5 Функции и объекты PHP

К основным требованиям к любому языку программирования относится наличие места для хранения данных, средств для направления процесса выполнения программы и ряда других мелочей, таких как вычисление выражений, управление файлами и вывод текста. PHP обладает всем этим, и вдобавок у него имеется облегчающий жизнь инструментарий наподобие инструкций `else` и `elseif`. Но даже если все это входит в наш набор инструментов, программирование может быть слишком нудным и утомительным занятием, особенно если регулярно будет возникать необходимость вновь и вновь набирать очень похожие друг на друга фрагменты кода.

И тут нам на помощь приходят функции и объекты. Нетрудно догадаться, что *функция* — это набор инструкций, который выполняет конкретную задачу и в дополнение к этому может вернуть какое-нибудь значение. Можно извлечь фрагмент кода, который используется более одного раза, поместить его в функцию и вызвать функцию по имени в тот момент, когда этот код нужно будет выполнить.

По сравнению с непрерывным линейным кодом у функций есть масса преимуществ.

- Экономия времени при наборе текста программы.
- Сокращение количества синтаксических и прочих ошибок программирования.
- Сокращение времени загрузки файлов программы.
- Сокращение времени выполнения, поскольку каждая функция компилируется только один раз, независимо от частоты ее вызовов.
- Возможность использовать функции как в рядовых, так в особенных случаях, поскольку они воспринимают аргументы.

Объекты являются дальнейшим развитием этой концепции. *Объект* объединяет одну или несколько функций и данные, которые ими используются, в единую структуру, которая называется *классом*.

В этой главе будет рассмотрено все, что касается использования функций, от их определения и вызова до различных способов передачи данных. Вооружившись этими знаниями, вы сможете создавать функции и использовать их в собственных объектах (в которых они будут упоминаться как *методы*).

# Функции PHP

PHP поставляется с несколькими сотнями готовых к работе встроенных функций, превращающих его в язык с очень богатыми возможностями. Чтобы воспользоваться функцией, ее нужно вызвать по имени. Посмотрим, например, как работает функция print:

```
print("print является функцией");
```

Круглые скобки сообщают PHP, что вы ссылаетесь на функцию. В противном случае будет считаться, что вы ссылаетесь на константу, и может быть выдано уведомление об использовании неопределенной константы:

```
Notice: Use of undefined constant fname - assumed 'fname'
```

за которым последует текстовая строка fname, согласно предположению, что вы, наверное, хотели поместить в код текстовую строку. (Ситуация запутается еще больше, если константа по имени fname будет существовать на самом деле и PHP в таком случае воспользуется ее значением.)

Собственно говоря, print является псевдофункцией, которая обычно называется конструкцией. Разница в том, что при ее использовании круглые скобки можно опустить:

print "print не требует использования круглых скобок";

А после любого другого имени вызываемой функции скобки нужно ставить всегда, даже если они останутся пустыми (в том случае, когда функции не передаются никакие аргументы).

Функции могут принимать любое количество аргументов, включая нулевое. Например, показанная ниже функция phpinfo, отображает массу информации о текущей установке PHP и не требует никаких аргументов:

```
phpinfo();
```

Результат вызова этой функции показан на рис. 5.1.

Функция phpinfo весьма полезна для получения информации о текущей установке PHP, но этой информацией могут воспользоваться и потенциальные злоумышленники. Поэтому никогда не оставляйте вызов этой функции в коде, подготовленном для работы в сети.

В примере 5.1 показан ряд встроенных функций, использующих один аргумент и более.

**Пример 5.1.** Три функции для работы со строками

```
<?php
echo strrev(" .dlrow olleH"); // Реверсирование строки
echo str_repeat("Hip ", 2);   // Повторение строки
echo strtoupper("hooray!");   // Преобразование символов строки в верхний
                              // регистр
?>
```

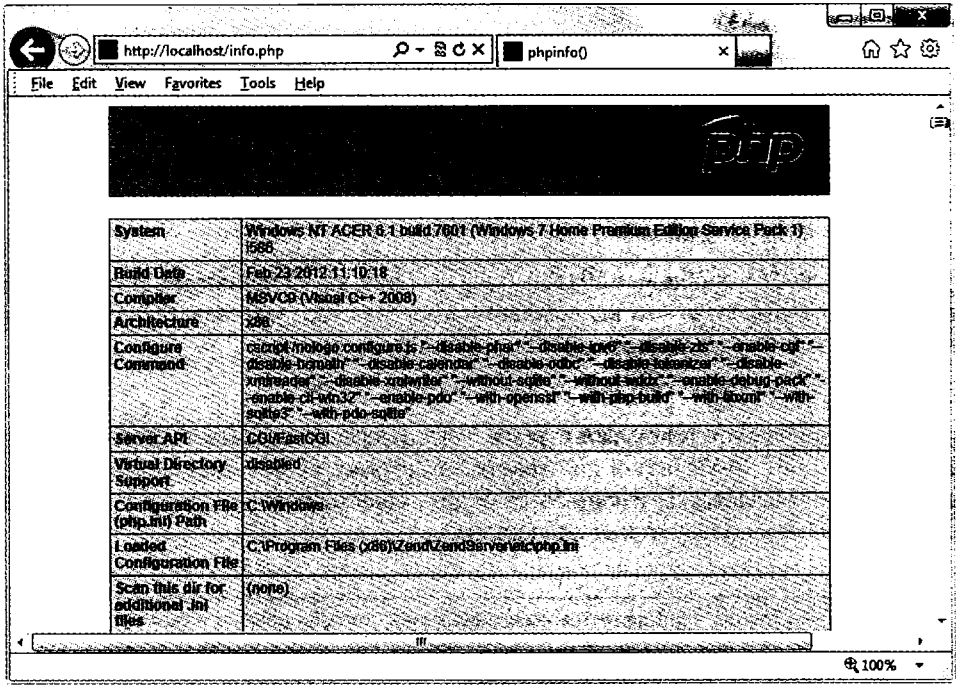



**Рис. 5.1.** Информация, выводимая встроенной в PHP функцией phpinfo

В этом примере используются три функции для обработки строк, выводящие следующий текст:

```
Hello world. Hip Hip HOORAY!
```

Как следует из результата, функция `strrev` реверсирует порядок символов в строке, функция `str_repeat` дважды повторяет строку `Hip` (в соответствии с требованием второго аргумента), а функция `strtoupper` переводит буквы в слове hoorау! в верхний регистр.

# Определение функции

В общем виде для функции используется следующий синтаксис:

```
function имя_функции([параметр [, ...]])
{
    // Инструкции
}
```

В первой строке синтаксиса показано следующее:

- определение начинается со слова `function`;
- за ним следует имя, которое должно начинаться с буквы или символа подчеркивания, за которыми может следовать любое количество букв, цифр или знаков подчеркивания;

- наличие круглых скобок обязательно;
- к необязательному элементу относится один или несколько параметров, разделенных запятыми (что отображено с помощью квадратных скобок, не являющихся частью синтаксиса функции).

Имена функций нечувствительны к регистру использующихся в них букв, поэтому все следующие строки могут ссылаться на одну и ту же функцию print: `PRINT`, `Print` и `PrInT`.

С открывающей фигурной скобки начинаются инструкции, которые будут выполнены при вызове функции; они должны завершаться закрывающей фигурной скобкой, составляющей пару первой скобке. В составе этих инструкций должна быть одна или несколько инструкций `return`, заставляющих функцию прекратить выполнение и вернуть управление вызывавшему функцию коду. Если инструкция `return` продолжена каким-нибудь значением, то вызывающий код может его извлечь, что мы сейчас и увидим.

## Возвращение значения

Рассмотрим простую функцию, преобразующую буквы чьих-нибудь полных имен в нижний регистр, а затем переводящую в верхний регистр первую букву каждого имени.

В примере 5.1 нам уже встречалась встроенная PHP-функция `strtoupper`. Для нашей текущей функции будет использована ее противоположность: функция `strtolower`:

```
$lowered = strtolower("люБОЕ нУжное Вам количество Букв и Знаков Пунктуации");
echo $lowered;
```

На выходе этого эксперимента получается следующая строка:

```
любое нужное вам количество букв и знаков пунктуации
```

Но нам не нужны имена, полностью состоящие из букв нижнего регистра, мы хотим, чтобы первые буквы были превращены в прописные. (Не будем в этом примере брать в расчет такие редкие имена, как Mary-Ann или Jo-En-Lai.) Нам и здесь сопутствует удача: PHP предоставляет также функцию `ucfirst`, которая переводит первую букву строки в верхний регистр:

```
$ucfixed = ucfirst("любое нужное вам количество букв и знаков пунктуации");
echo $ucfixed;
```

На выходе получается следующая строка:

```
Любое нужное вам количество букв и знаков пунктуации
```

Теперь мы можем внести свою первую лепту в конструирование программы: чтобы получить слово с первой прописной буквой, сначала для строки будет вызвана функция `strtolower`, а затем будет вызвана функция `ucfirst`. Для этого вызов функции `strtolower` будет вложен в вызов функции `ucfirst`. Давайте посмотрим, зачем это делается, потому что нам важно понять порядок вычисления кода.

Если воспользоваться следующим простым вызовом функции print:

```
print(5-8);
```

то сначала будет вычислено выражение 5-8 и на выходе будет получено число -3. (В предыдущей главе уже было показано, что PHP для отображения этого результата превращает его в строку.) Если выражение содержит функцию, то эта функция также в этот момент вычисляется:

```
print(abs(5-8));
```

Для выполнения этой короткой инструкции PHP совершает следующие действия.

1. Вычисляет 5-8, выдавая результат -3.
2. Использует функцию abs, превращая -3 в 3.
3. Превращает результат в строку и выводит его, используя функцию print.

Иными словами, PHP вычисляет каждый элемент, начиная с самого внутреннего и заканчивая тем, который находится снаружи. То же самое происходит и при обработке следующего вызова:

```
ucfirst(strtolower("люБОЕ, нУжное Вам количество Букв и Знаков Пунктуации"))
```

PHP передает нашу строку функции strtolower, а затем функции ucfirst, выдавая следующий результат (который мы уже видели, когда вызывали функции отдельно друг от друга):

```
Любое, нужное вам количество букв и знаков пунктуации
```

Теперь определим функцию (показанную в примере 5.2), которая берет три имени и переводит их буквы в нижний регистр, после чего превращает первую букву в прописную.

**Пример 5.2.** Приведение в порядок полного имени

```
<?php
echo fix_names("WILLIAM", "henry", "gatES");

function fix_names($n1, $n2, $n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));
    return $n1 . " " . $n2 . " " . $n3;
}
?>
```

Пользователи часто забывают вовремя выключить режим Caps Lock, случайно ставят прописные буквы не там, где нужно, и даже вообще забывают о них, от чего вы тоже не застрахованы. В результате выполнения кода этого примера будет выведен следующий текст:

```
William Henry Gates
```

## Возвращение массива

Выше была рассмотрена функция, возвращающая единственное значение. Но существуют также способы получения при выполнении функции сразу нескольких значений.

Самый подходящий из них возвращает эти значения в виде массива. В главе 3 уже было показано, что массив похож на связку переменных в одной строке. Использование массива для возвращения значений функции отображено в примере 5.3.

**Пример 5.3.** Возвращение нескольких значений в массиве

```
<?php
$names = fix_names("WILLIAM", "henry", "gatES");
echo $names[0] . " " . $names[1] . " " . $names[2];

function fix_names($n1, $n2, $n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));
    return array($n1, $n2, $n3);
}
?>
```

У этого метода есть преимущество, заключающееся в том, что все три имени содержатся по-отдельности, а не объединяются в одну строку, что дает возможность обращаться к любому пользователю просто по его имени или фамилии, не извлекая каждое имя из возвращаемой строки.

## Передача по ссылке

Когда в PHP перед именем переменной ставится символ &, парсер знает, что тем самым передается не сама переменная, а ссылка на нее. Это понятие может быть для вас сложным, поэтому вернемся к метафоре со спичечным коробком, которая использовалась в главе 3.

Представьте, что вы не вынимаете клочок бумаги из коробка, не читаете то, что на нем написано, не копируете эту надпись на другой клочок бумаги, не возвращаете оригинал в коробок и не передаете копию функции, а просто привязываете нитку к исходному клочку бумаги и передаете функции второй конец этой нитки (рис. 5.2).

Теперь, чтобы найти данные, к которым она обращается, функция может проследовать по нитке. Таким образом исключаются все издержки на создание копии переменной, предназначенной только для того, чтобы в функции можно было воспользоваться ее значением. Более того, теперь функция может изменить значение переменной.

Значит, пример 5.3 можно переписать: передать ссылки на все параметры, чтобы после этого функция напрямую смогла внести в них изменения (пример 5.4).

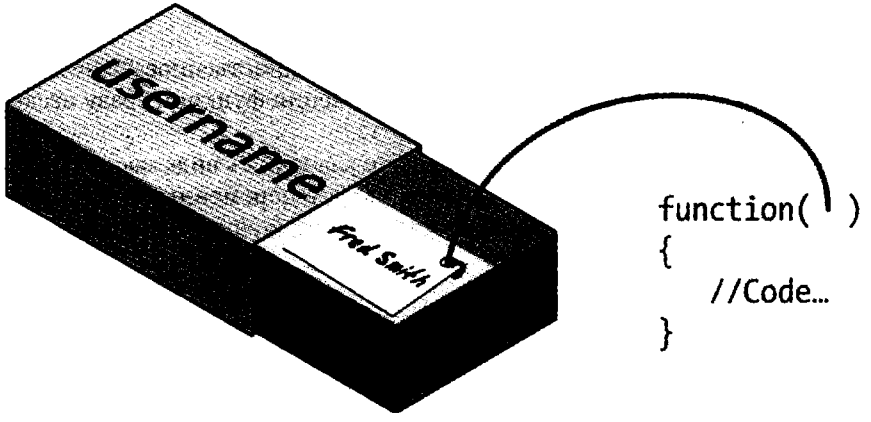



**Рис. 5.2.** Представление ссылки в виде нитки, привязанной к значению переменной

**Пример 5.4.** Возвращение значений из функции по ссылке

```
<?php
$a1 = "WILLIAM";
$a2 = "henry";
$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br />";
fix_names($a1, $a2, $a3);
echo $a1 . " " . $a2 . " " . $a3;

function fix_names(&$n1, &$n2, &$n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 =.ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));
}
?>
```

Вместо передачи строк непосредственно функции они сначала присваиваются в качестве значений переменным и выводятся на экран, чтобы посмотреть их состояние «до». Затем, как и раньше, вызывается функция, но теперь перед именем каждого параметра ставится символ &, предписывающий PHP передать функции только ссылки на значения переменных.

Теперь к переменным `$n1`, `$n2` и `$n3` привязаны «ниточки», ведущие к значениям переменных `$a1`, `$a2` и `$a3`. Иными словами, существует одна группа значений, но два набора имен переменных, позволяющих к ним обратиться.

Поэтому функции `fix_names` нужно только присвоить новые значения переменным `$n1`, `$n2` и `$n3`, чтобы обновить значения переменных `$a1`, `$a2` и `$a3`. В результате выполнения этого кода будут выведены следующие строки:

```
WILLIAM henry gatES
William Henry Gates
```

Как видите, в обеих инструкциях echo используются только значения переменных $a1, $a2 и $a3.

При передаче значений по ссылке следует соблюдать особую осторожность. Если нужно сохранить исходное значение, сделайте копии ваших переменных, а затем передайте значения этих копий по ссылке.

## Возвращение глобальных переменных

Можно также дать функции доступ к переменной, созданной за ее пределами, объявив ее глобальной прямо из тела функции. За ключевым словом global должно следовать имя переменной, тогда полный доступ к этой переменной можно будет получить из любой части вашего кода (пример 5.5).

**Пример 5.5.** Возвращение значений в глобальных переменных

```
<?php
$a1 = "WILLIAM";
$a2 = "henry";
$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br />";
fix_names();
echo $a1 . " " . $a2 . " " . $a3;

function fix_names()
{
    global $a1; $a1 = ucfirst(strtolower($a1));
    global $a2; $a2 = ucfirst(strtolower($a2));
    global $a3; $a3 = ucfirst(strtolower($a3));
}
?>
```

Теперь уже не нужно передавать функции параметры, и она не должна их принимать. После объявления эти переменные остаются глобальными и доступными коду всей остальной программы, включая ее функции.

По возможности в целях сохранения как можно большей локальной видимости переменных следует возвращать массивы или использовать переменные, переданные по ссылке. В противном случае начнут утрачиваться некоторые преимущества использования функций.

## И еще раз об области видимости переменных

Кратко напомню те сведения, которые были получены при изучении главы 3.

- Локальные переменные доступны лишь из той части кода, в которой они были определены. Если это произошло за пределами функции, доступ к переменным будет возможен из всего кода, находящегося за пределами функций, классов и т. д. Если переменная была определена внутри функции, значит, доступ к ней

может получить только код этой функции и ее значение теряется при выходе из функции.

- Глобальные переменные доступны из любых частей вашего кода.
- Статические переменные доступны только внутри функции, в которой они были объявлены, но при этом они сохраняют свое значение в процессе многократных вызовов функции.

# Включение и запрос файлов

По мере приобретения навыков программирования на PHP вы, скорее всего, приступите к созданию библиотеки, состоящей из функций, которые, по вашему мнению, смогут пригодиться в будущем. Кроме того, наверное, вы начнете пользоваться библиотеками, созданными другими программистами.

Копировать эти функции и вставлять их в свой код не имеет никакого смысла. Можно сохранить эти функции в отдельных файлах и воспользоваться командами для их извлечения. Для этого существуют две команды: `include` (включить) и `require` (затребовать).

## Инструкция include

При использовании инструкции `include` можно потребовать у PHP извлечения конкретного файла и загрузки всего его содержимого. Это равносильно вставке включаемого файла в данное место текущего файла. В примере 5.6 показано, как нужно включать файл под названием `library.php`.

**Пример 5.6.** Включение файла PHP

```
<?php
include "library.php";

// Сюда помещается ваш код
?>
```

## Использование инструкции include_once

При каждом использовании директивы `include` она снова вставляет требуемый файл, даже если он уже был вставлен. Предположим, к примеру, что в библиотеке `library.php` содержится масса полезных функций. Вы включаете ее в свой файл, но, кроме нее, включаете также еще одну библиотеку, которая содержит `library.php`. Из-за этой вложенности вы непреднамеренно вставляете `library.php` дважды. В результате будут появляться сообщения об ошибках, потому что будет предпринята попытка несколько раз объявить одну и ту же константу или функцию. Поэтому вместо данной директивы нужно использовать инструкцию `include_once` (пример 5.7).

**Пример 5.7.** Однократное включение файла PHP

```
<?php
include_once "library.php";

// Сюда помещается ваш код
?>
```

Теперь, если встретится еще одна инструкция include или include_once, похожая на ту, которая уже была выполнена, она будет полностью проигнорирована. Чтобы определить, был ли файл уже включен, абсолютный путь к нему сравнивается со всеми раскрытыми относительными путями и файлом, найденным в пути, который указан в вашей инструкции include.

Вообще-то, наверное, лучше будет придерживаться использования инструкции include_once и не применять инструкцию include. Тогда у вас никогда не будет проблем с тем, что файлы вставляются по нескольку раз.

## Использование инструкций require и require_once

Потенциальная проблема, возникающая при использовании инструкций include и include_once, состоит в том, что для вставки нужного файла PHP предпримет всего одну попытку. Выполнение программы продолжится даже в том случае, если файл не будет найден.

Когда вставка файла имеет принципиальную важность, его нужно затребовать, то есть применить инструкцию require. По тем же причинам, которые излагались при рассмотрении использования инструкции include_once, я рекомендую, чтобы вы, когда нужно затребовать файл, придерживались главным образом использования инструкции require_once (пример 5.8).

**Пример 5.8.** Однократное востребование файла PHP

```
<?php
require_once "library.php";

// Сюда помещается ваш код
?>
```

# Совместимость версий PHP

PHP продолжает совершенствоваться и существует в нескольких версиях. Если нужно проверить доступность в вашем коде какой-нибудь конкретной функции, можно воспользоваться функцией function_exists, которая проверяет все предопределенные и созданные пользователем функции.

В примере 5.9 проверяется доступность функции array_combine, которая имеется в PHP версии 5.

**Пример 5.9.** Проверка существования функции

```
<?php
if (function_exists("array_combine"))
{
    echo "Функция существует";
}
else
{
    echo "Функция не существует, желательно создать ее самостоятельно";
}
?>
```

Используя подобный код, можно определить доступность любых функциональных возможностей в новых версиях PHP, которые вам придется смоделировать, если нужно будет, чтобы ваш код работал и в более ранних версиях. Ваши функции могут работать медленнее встроенных, но код по крайней мере будет обладать более широкой переносимостью.

Чтобы определить версию PHP, под которой запущен ваш код, можно также воспользоваться функцией `phpversion`. Возвращаемый результат в зависимости от версии будет иметь следующий вид:

```
5.2.8
```

# Объекты PHP

Практически так же, как применение функций стало фактором существенного увеличения эффективности программирования на заре развития вычислительной техники (когда лучшим из доступных средств программной навигации порой была самая элементарная инструкция `GOTO` или `GOSUB`), объектно-ориентированное программирование (ООП) подняло использование функций на совершенно новый уровень.

Как только у вас появится навык сведения повторно используемых фрагментов кода в функции, останется сделать еще один небольшой шаг и присмотреться к связыванию функций и данных, которыми они оперируют, в объекты.

Рассмотрим сайт социальной сети, состоящий из множества различных частей. Одна из таких частей управляет всеми пользовательскими функциями: ее код позволяет новым пользователям записаться, а уже записавшимся — изменить свои личные данные. В стандартном PHP можно создать для управления всеми этими действиями ряд функций и встроить несколько вызовов к базе данных MySQL, чтобы вести данные по всем пользователям.

А теперь вообразите, насколько проще будет создать объект, представляющий текущего пользователя. Для этого можно создать класс по имени `User`, в котором будут содержаться весь код, необходимый для обслуживания пользователей, и все переменные, требующиеся для работы с данными внутри класса. Затем, когда понадобится управлять пользовательскими данными, можно будет просто создать новый объект класса `User`.

Этот новый объект можно будет рассматривать в качестве настоящего пользователя. Например, объекту можно передать имя, пароль и адрес электронной почты, спросить его о том, существует ли уже такой пользователь, и если нет, заставить его создать нового пользователя с данными атрибутами. Можно даже иметь объект мгновенных сообщений или объект, позволяющий учитывать дружеские отношения между двумя пользователями.

## Терминология

При создании программы, рассчитанной на использование объектов, нужно сконструировать некую совокупность данных и кода, называемую *классом*. Каждый новый объект, основанный на этом классе, называется *экземпляром* (или *случаем употребления*) этого класса.

Данные, связанные с объектом, называются его *свойствами*, а используемые им функции — *методами*. При определении класса задаются имена его свойств и код для его методов. На рис. 5.3 показана метафора объекта в виде музыкального автомата. Компакт-диски в его карусели можно рассматривать в качестве его свойств, а метод их проигрывания заключается в нажатии кнопки на передней панели. Есть также щель для опускания монет (метод, используемый для активации объекта) и устройство чтения компакт-дисков (метод, используемый для извлечения музыки, или свойств, с компакт-дисков).

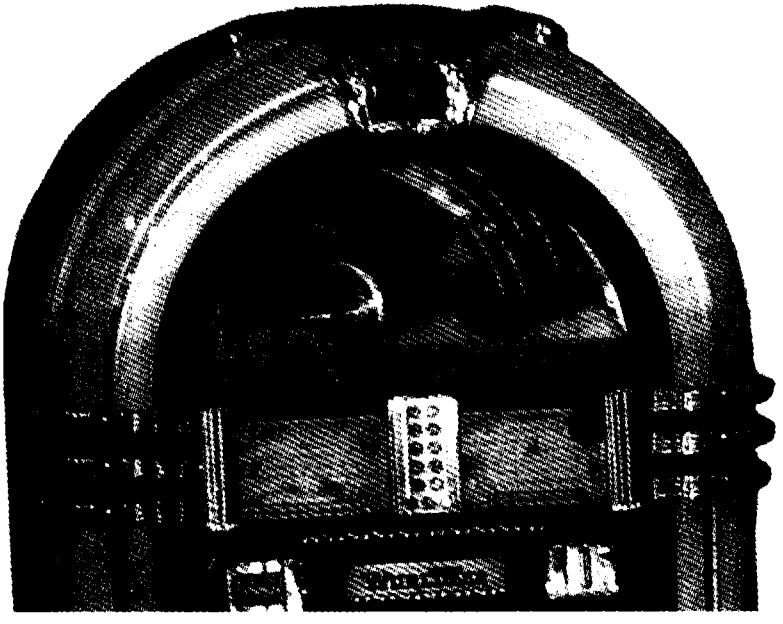


**Рис. 5.3.** Музыкальный автомат как подходящий пример автономного объекта

При создании объектов предпочтительно воспользоваться инкапсуляцией или создавать класс таким образом, чтобы с его свойствами могли работать только его

собственные методы. Иными словами, нужно запретить внешнему коду непосредственный доступ к данным объекта. Предоставляемые объектом методы известны как *интерфейс* объекта.

Такой подход упрощает отладку: дефектный код придется исправлять только в пределах класса.

Кроме того, когда нужно будет обновить программу, при использовании надлежащей инкапсуляции и поддержке одинакового интерфейса можно будет просто разработать новые классы для замены старых, полностью их отладить, а затем заменить ими старые классы. Если они будут в чем-то неработоспособными, можно будет вернуть назад старые классы для немедленного устранения проблемы перед дальнейшей отладкой новых классов.

Как только класс будет создан, может выясниться, что нужен еще один, похожий на него, но все же несколько отличающийся класс. Быстрее и проще всего будет определить новый класс, воспользовавшись *наследованием*. При этом ваш новый класс сохранит все свойства, присущие тому классу, чьим наследником он является. Исходный класс теперь будет называться *суперклассом*, а новый класс — *подклассом* (или *производным* классом).

Вернемся к примеру с музыкальным автоматом. Если вы изобретаете новый музыкальный автомат, который наряду с музыкой может воспроизводить и видеоклипы, то можете сохранить все свойства и методы исходного музыкального автомата и добавить ряд новых свойств (видеоклипов) и новых методов (видеоплееров).

Существенным преимуществом этой системы является то, что, если вы увеличили скорость работы или улучшили другие аспекты работы суперкласса, его подклассы пользуются теми же самыми усовершенствованиями.

## Объявление класса

Перед тем как получить возможность использования объекта, нужно определить класс с помощью ключевого слова `class`. Определение класса включает в себя имя класса (чувствительное к регистру букв), его свойства и методы. В примере 5.10 дается определение класса `User`, имеющего два свойства: `$name` и `$password` (которые обозначены ключевым словом `public` — см. подраздел «Область видимости свойств и методов в PHP 5» данного раздела). В нем также создается новый экземпляр этого класса (по имени `$object`).

**Пример 5.10.** Объявление класса и проверка объекта

```
<?php
$object = new User;
print_r($object);

class User
{
    public $name, $password;

    function save_user()
    {
        echo "Сюда помещается код, сохраняющий данные пользователя";
```

```
    }
}
?>
```

Здесь также задействована поистине бесценная функция под названием `print_r`. Она требует от PHP отобразить информацию о переменной в удобной для восприятия человеком форме, о чем говорит элемент `_r` в ее имени (означающий readable, «читаемый»). Для нового объекта `$object` эта функция выводит следующую информацию:

```
User Object
(
    [name] =>
    [password] =>
)
```

Но браузер сжимает все пустые пространства, поэтому выводимая в нем информация читается немного сложнее:

```
User Object ( [name] => [password] => )
```

В любом случае выведенная информация свидетельствует о том, что `$object` является объектом, определенным пользователем, и содержит свойства `name` и `password`.

## Создание объекта

Для создания объекта определенного класса используется ключевое слово `new`, применяемое в выражении: *$объект* = `new` *Класс*. Вот два способа создания объектов:

```
$object = new User;
$temp = new User('name', 'password');
```

В первой строке мы просто назначаем объект классу `User`. А во второй строке передаем вызову параметры.

Класс может требовать или запрещать аргументы; он также может разрешать, но не требовать их.

## Доступ к объектам

Добавим к примеру 5.10 еще несколько строк и проверим результаты. В примере 5.11 предыдущий код расширяется за счет установки свойств объекта и вызова метода.

**Пример 5.11.** Создание объекта и взаимодействие с ним

```
<?php
$object = new User;
print_r($object); echo "<br />";

$object->name = "Joe";
```

```
$object->password = "mypass";
print_r($object); echo "<br />";

$object->save_user();

class User
{
    public $name, $password;

    function save_user()
    {
        echo "Сюда помещается код, сохраняющий данные пользователя";
    }
}
?>
```

Из примера видно, что для доступа к свойству объекта используется следующий синтаксис: $*объект->свойство*. Похожим образом можно вызвать и метод: $*объект->метод*().

Можно было заметить, что перед именами свойств и методов отсутствуют символы доллара ($). Если на первой позиции имен поставить символ $, то код не будет работать, поскольку он попробует обратиться к значению, хранящемуся в переменной. Например, выражение `$object->$property` будет пытаться найти значение, присвоенное переменной по имени `$property` (скажем, это значение является строкой brown), а затем обратиться к свойству `$object->brown`. Если переменная `$property` не определена, то будет предпринята попытка обратиться к свойству `$object->NULL`, что спровоцирует возникновение ошибки.

Если организовать просмотр, используя имеющееся в браузере средство для просмотра исходного кода, то код примера 5.11 выведет следующую информацию:

```
User Object
(
    [name]     =>
    [password] =>
)
User Object
(
    [name]     => Joe
    [password] => mypass
)
Сюда помещается код, сохраняющий данные пользователя
```

Здесь также используется функция `print_r`, которая предоставляет содержимое переменной `$object` до и после присваивания свойству значения. В дальнейшем я не буду использовать инструкцию `print_r`, но если материал этой книги будет прорабатываться на вашем разработочном сервере, вы сможете поместить в код несколько таких инструкций, чтобы иметь полное представление о том, что происходит.

Можно было также заметить, что благодаря вызову метода save_user был выполнен код этого метода, который вывел строку, напоминающую о том, что нужно создать некий код.

Определения функций и классов можно помещать в любое место вашего кода, до или после инструкций, в которых они используются. Но правилом хорошего тона считается помещать их ближе к концу файла.

## Клонирование объектов

Если объект уже создан, то в качестве параметра он передается по ссылке. Если воспользоваться метафорой спичечного коробка, то это похоже на привязывание сразу нескольких ниток к объекту, хранящемуся в коробке, что позволяет получить к нему доступ, следуя по любой из привязанных ниток.

Иными словами, присваивание объектов не приводит к их полному копированию.

Как это работает, показано в примере 5.12, где определяется очень простой пользовательский класс User, который не имеет методов и содержит всего лишь одно свойство name.

**Пример 5.12.** Копирование объекта?

```
<?php
$object1       = new User();
$object1->name = "Alice";
$object2       = $object1;
$object2->name = "Amy";
echo "object1 name = " . $object1->name . "<br />";
echo "object2 name = " . $object2->name;

class User
{
    public $name;
}
?>
```

Мы создали объект $object1 и присвоили свойству name значение Alice. Затем создали $object2, присвоили ему значение $object1 и присвоили значение Amy непосредственно свойству name объекта $object2 — или подумали, что присвоили. Но этот код выдаст следующую информацию:

```
object1 name = Amy
object2 name = Amy
```

Что же произошло? И $object1, и $object2 ссылаются на *один и тот же* объект, поэтому изменение свойства name, принадлежащего $object2, на Amy устанавливает такое же значение и для свойства, принадлежащего $object1.

Во избежание подобной путаницы следует использовать инструкцию clone, которая создает новый экземпляр класса и копирует значения свойств из исходного класса в новый экземпляр. Применение этой инструкции показано в примере 5.13.

**Пример 5.13.** Клонирование объекта

```
<?php
$object1       = new User();
$object1->name = "Alice";
$object2       = clone $object1;
$object2->name = "Amy";
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;

class User
{
    public $name;
}
?>
```

Вот и все. Этот код выдает то, что нам требовалось получить с самого начала:

```
object1 name = Alice
object2 name = Amy
```

# Конструкторы

При создании нового объекта вызываемому классу можно передать перечень аргументов. Они передаются специальному методу внутри класса, который называется *конструктором* и занимается инициализацией различных свойств.

В прежние времена этому методу обычно давалось имя класса, как в примере 5.14.

**Пример 5.14.** Создание метода-конструктора

```
<?php
class User
{
    function User($param1, $param2)
    {
        // Сюда помещаются инструкции конструктора
        public $username = "Guest";
    }
}
?>
```

В примере 5.15 показано, что PHP 5 предоставляет более логичный подход к присвоению имени конструктору, при котором функции присваивается имя `__construct` (то есть к слову `construct` спереди добавляются два символа подчеркивания).

**Пример 5.15.** Создание метода-конструктора в PHP 5

```
<?php
class User
{
    function __construct($param1, $param2)
```

```
    {
        // Сюда помещаются инструкции конструктора
        public $username = "Guest";
    }
}
?>
```

## Деструкторы в PHP 5

Еще одним нововведением в PHP 5 стала возможность создания методов-деструкторов. Эта возможность подходит для тех случаев, когда код ссылается на объект в последний раз или когда сценарий подошел к концу. В примере 5.16 показано, как создается метод-деструктор.

**Пример 5.16.** Создание в PHP 5 метода-деструктора

```
<?php
class User
{
    function __destruct()
    {
        // Сюда помещается код деструктора
    }
}
?>
```

# Написание методов

Как видите, объявление метода похоже на объявление функции, но есть некоторые отличия. Например, имена методов, начинающиеся с двойного подчеркивания (__), являются зарезервированными словами и вы не должны больше создавать ничего подобного.

Кроме того, существует специальная переменная `$this`, которая может использоваться для доступа к свойствам текущего объекта. Чтобы понять, как это работает, посмотрите на код примера 5.17, содержащий еще один метод из определения класса `User`, который называется `get_password`.

**Пример 5.17.** Использование в методе переменной `$this`

```
<?php
class User
{
    public $name, $password;

    function get_password()
    {
        return $this->password;
    }
}
?>
```

Метод получения пароля — `get_password` — применяет переменную `$this` для доступа к текущему объекту, а затем возвращает значение свойства `password`,

принадлежащего этому объекту. Обратите внимание на то, как при использовании оператора -> в имени свойства $password опускается первый символ $. Если оставить его на прежнем месте, особенно при первом применении этого свойства, будет допущена весьма типичная ошибка.

Класс, определенный в примере 5.17, нужно использовать следующим образом:

```
$object           = new User;
$object->password = "secret";
echo $object->get_password();
```

Этот код выводит пароль secret.

## Статические методы в PHP 5

При работе в PHP 5 можно также определить метод как *статический*, что означает возможность его вызова в классе, но не в объекте. Статический метод не имеет доступа ни к одному из свойств объекта, а создание такого метода и доступ к нему показаны в примере 5.18.

**Пример 5.18.** Создание статического метода и доступ к нему

```
<?php
User::pwd_string();

class User
{
    static function pwd_string()
    {
        echo "Пожалуйста, введите свой пароль";
    }
}
?>
```

Обратите внимание на то, как наряду со статическим методом вызывается сам класс и как при этом вместо оператора -> используется оператор двойного двоеточия (::), также известный как оператор разрешения области видимости. Статические функции полезны для совершения действий, относящихся к самому классу, но не к конкретным экземплярам этого класса. Еще один пример статического метода показан в примере 5.21 далее.

При попытке получить доступ к свойству текущего объекта с помощью выражения $this->property или получить доступ к другим свойствам объекта внутри статической функции будет выдано сообщение об ошибке.

# Объявление свойств

В явном объявлении свойств внутри классов нет необходимости, поскольку они могут быть определены неявным образом при первом же их использовании. Для иллюстрации этой особенности класс User в примере 5.19 не имеет ни свойств, ни методов, но при этом в коде его определения нет ничего противозаконного.

**Пример 5.19.** Неявное объявление свойства

```
<?php
$object1       = new User();
$object1->name = "Alice";
echo $object1->name;
class User {}
?>
```

Этот код вполне корректно и без проблем выведет строку `Alice`, поскольку PHP неявным образом объявит для вас переменную `$object1->name`. Но такой стиль программирования может привести к ошибкам, найти которые будет невероятно трудно, поскольку свойство `name` было объявлено за пределами класса.

Чтобы не создавать трудностей ни себе, ни тому, кто впоследствии будет обслуживать ваш код, я советую выработать привычку всегда объявлять свойства внутри класса в явном виде. И поверьте, вы об этом никогда не пожалеете.

К тому же, когда свойство объявляется внутри класса, ему можно присвоить значение по умолчанию. Используемое вами значение должно быть константой, а не результатом вызова функции или вычисления выражения. Несколько допустимых и недопустимых присваиваний показано в примере 5.20.

**Пример 5.20.** Допустимые и недопустимые объявления свойств

```
<?php
class Test
{
    public $name     = "Paul Smith"; // Допустимое
    public $age      = 42;           // Допустимое
    public $time     = time();       // Недопустимое – вызывает функцию
    public $score    = $level * 2;   // Недопустимое – использует выражение
}
?>
```

## Объявление констант

По аналогии с созданием глобальных констант внутри определения функций можно определять константы и внутри классов. Чтобы константы выделялись на общем фоне, обычно для их имен используют буквы верхнего регистра (пример 5.21).

**Пример 5.21.** Определение констант внутри класса

```
<?php
Translate::lookup();

class Translate
{
    const ENGLISH = 0;
    const SPANISH = 1;
    const FRENCH  = 2;
    const GERMAN  = 3;
    // ...

    function lookup()
```

```
    {
        echo self::SPANISH;
    }
}
?>
```

К константам можно обращаться напрямую, с помощью ключевого слова `self` и оператора двойного двоеточия. Обратите внимание на то, что этот код, в первой строке которого используется оператор двойного двоеточия, вызывает класс напрямую, без предварительного создания его экземпляра. Как и ожидалось, значение, выводимое при запуске этого кода на выполнение, будет равно 1.

Запомните, что константа после ее определения не может быть изменена.

## Область видимости свойств и методов в PHP 5

PHP 5 предоставляет три ключевых слова для управления областью видимости свойств и методов.

- `public` (открытые). Свойства с этой областью видимости получаются по умолчанию при объявлении переменной с помощью ключевых слов `var` или `public` или когда переменная объявляется неявно при первом же ее использовании.

  Ключевые слова `var` и `public` являются взаимозаменяемыми. Хотя сейчас использование `var` не приветствуется, оно сохранено для совместимости с предыдущими версиями PHP. Методы считаются открытыми по умолчанию.
- `protected` (защищенные). На свойства и методы с этой областью видимости можно ссылаться только через принадлежащие объектам методы класса и такие же методы любых подклассов.
- `private` (закрытые). К представителям класса с этой областью видимости можно обращаться через методы этого же класса, но не через методы его подклассов.

Решение о том, какую область видимости применить, принимается на основе следующих положений.

- Открытую (public) область видимости следует применять, когда к представителю класса нужен доступ из внешнего кода и когда расширенные классы должны его наследовать.
- Защищенную (protected) область видимости необходимо использовать, когда к представителю класса не должно быть доступа из внешнего кода, но расширенные классы все же должны его наследовать.
- Закрытую (private) область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода и когда расширенные классы не должны его наследовать.

Применение этих ключевых слов показано в примере 5.22.

**Пример 5.22.** Изменение области видимости свойства и метода

```
<?php
class Example
```

```
{
    var $name   = "Michael"; // Не рекомендуемая форма, аналогичная public
    public $age = 23;        // Открытое свойство
    protected $usercount;    // Защищенное свойство

    private function admin() // Закрытый метод
    {
        // Сюда помещается код метода admin
    }
}
?>
```

## Статические свойства и методы

Большинство данных и методов применяются в экземплярах класса. Например, в классе User следует установить конкретный пароль пользователя или проверить, когда пользователь был зарегистрирован.

Эти факты и операции имеют особое отношение к каждому конкретному пользователю и поэтому применяют специфические для экземпляра свойства и методы.

Но время от времени возникает потребность обслуживать данные, относящиеся целиком ко всему классу. Например, для отчета о том, сколько пользователей зарегистрировалось, будет храниться переменная, имеющая отношение ко всему классу User. Для таких данных PHP предоставляет статические свойства и методы.

В примере 5.18, приведенном выше, было показано, что объявление представителей класса статическими делает их доступными и без создания экземпляров класса. Свойство, объявленное статическим, не может быть доступно непосредственно из экземпляра класса, но может быть доступно из статического метода.

В примере 5.23 определяется класс по имени Test, в котором содержатся статическое свойство и открытый метод.

**Пример 5.23.** Определение класса со статическим свойством

```
<?php
$temp = new Test();
echo "Test A: "   Test::$static_property   "<br />";
echo "Test B: " . $temp->get_sp()          "<br />";
echo "Test C: " . $temp->static_property   "<br />";

class Test
{
    static $static_property = "Это статическое свойство";

    function get_sp()
    {
        return self::$static_property;
    }
}
?>
```

Когда код будет запущен на выполнение, он выдаст следующую информацию:

```
Test A: Это статическое свойство
Test B: Это статическое свойство

Notice: Undefined property: Test::$static_property
Test C:
```

В этом примере показано, что на свойство `$static_property` можно ссылаться напрямую из самого класса, используя в `Test A` оператор двойного двоеточия. `Test B` также может получить его значение путем вызова метода `get_sp` объекта `$temp`, созданного из класса `Test`. Но `Test C` терпит неудачу, потому что статическое свойство `$static_property` недоступно объекту `$temp`.

Обратите внимание на то, как метод `get_sp` получает доступ к свойству `$static_property`, используя ключевое слово `self`. Именно таким способом можно получить непосредственный доступ к статическому свойству или константе внутри класса.

## Наследование

Как только класс будет создан, из него можно будет получить подкласс. Это сэкономит массу времени: вместо скрупулезного переписывания кода можно будет взять класс, похожий на тот, который следует создать, распространить его на подкласс и просто внести изменения в те места, которые будут иметь характерные особенности. Это достигается за счет использования инструкции `extends`.

В примере 5.24 класс `Subscriber` объявляется подклассом `User` путем использования инструкции `extends`.

**Пример 5.24.** Наследование и распространение класса

```
<?php
$object           = new Subscriber;
$object->name     = "Fred";
$object->password = "pword";
$object->phone    = "012 345 6789";
$object->email    = "fred@bloggs.com";
$object->display();

class User
{
    public $name, $password;

    function save_user()
    {
        echo "Сюда помещается код, сохраняющий данные пользователя";
    }
}

class Subscriber extends User
```

```
{
    public $phone, $email;

    function display()
    {
        echo "Name:  " . $this->name       "<br />";
        echo "Pass:  " . $this->password . "<br />";
        echo "Phone: " . $this->phone      "<br />";
        echo "Email: " . $this->email;
    }
}
?>
```

У исходного класса User имеются два свойства — $name и $password, а также метод для сохранения данных текущего пользователя в базе данных. Подкласс Subscriber расширяет этот класс за счет добавления еще двух свойств — $phone и $email и включения метода, отображающего свойства текущего объекта, который использует переменную $this. Данная переменная ссылается на текущее значение объекта, к которому осуществляется доступ. Этот код выведет следующую информацию:

```
Name:  Fred
Pass:  pword
Phone: 012 345 6789
Email: fred@bloggs.com
```

## Инструкция parent

Когда в подклассе создается метод с именем, которое уже фигурирует в его родительском классе, его инструкции переписывают инструкции из родительского класса. Иногда такое поведение идет вразрез с вашими желаниями и вам нужно получить доступ к родительскому методу. Для этого можно воспользоваться инструкцией parent, которая показана в примере 5.25.

**Пример 5.25.** Переписывание метода и использование инструкции parent

```
<?php
$object = new Son;
$object->test();
$object->test2();

class Dad
{
    function test()
    {
        echo "[Class Dad] Я твой отец<br />";
    }
}

class Son extends Dad
{
    function test()
```

```
    {
        echo "[Class Son] Я Лука<br />";
    }

    function test2()
    {
        parent::test();
    }
}
?>
```

Этот код создает класс по имени Dad (отец), а затем подкласс по имени Son (сын), который наследует свойства и методы родительского класса, а затем переписывает метод test. Поэтому, когда во второй строке кода вызывается метод test, выполняется новый метод. Единственный способ выполнения переписанного метода test в том варианте, в котором он существует в классе Dad, заключается в использовании инструкции parent, как показано в функции test2 класса Son. Этот код выведет следующую информацию:

```
[Class Son] Я Лука
[Class Dad] Я твой отец
```

Если нужно обеспечить вызов метода из текущего класса, можно воспользоваться ключевым словом self:

```
self::method();
```

## Конструкторы подкласса

При распространении класса и объявлении собственного конструктора вы должны знать, что PHP не станет автоматически вызывать метод-конструктор родительского класса. Чтобы обеспечивалось выполнение всего кода инициализации, подкласс, как показано в примере 5.26, всегда должен вызывать родительские конструкторы.

**Пример 5.26.** Вызов конструктора родительского класса

```
<?php
$object = new Tiger();
echo "У тигров есть...<br>";
echo "Мех: " . $object->fur . "<br />";
echo "Полосы: " . $object->stripes;

class Wildcat
{
    public $fur; // У диких кошек есть мех

    function __construct()
    {
        $this->fur = "TRUE";
    }
}

class Tiger extends Wildcat
```

```
{
    public $stripes; // У тигров есть полосы

    function __construct()
    {
        parent::__construct(); // Первоочередной вызов родительского
                               // конструктора
        $this->stripes = "TRUE";
    }
}
?>
```

В этом примере используются обычные преимущества наследования. В классе `Wildcat` (Дикая кошка) создается свойство `$fur` (мех), которое хотелось бы использовать многократно, потому мы создаем класс `Tiger` (Тигр), наследующий свойство `$fur`, и дополнительно создаем еще одно свойство — `$stripes` (полосы). Чтобы проверить вызов обоих конструкторов, программа выводит следующую информацию:

```
У тигров есть...
Мех: TRUE
Полосы: TRUE
```

## Методы Final

При необходимости помешать подклассу переписать метод суперкласса можно воспользоваться ключевым словом `final`. Как это делается, показано в примере 5.27.

**Пример 5.27.** Создание метода final

```
<?php
class User
{
    final function copyright()
    {
        echo "Этот класс был создан Джо Смитом ";
    }
}
?>
```

Усвоив содержание этой главы, вы должны приобрести твердое представление о том, что PHP может для вас сделать. Вы сможете без особого труда воспользоваться функциями и при необходимости создать объектно-ориентированный код. В главе 6 мы завершим начальное исследование PHP и рассмотрим работу с массивами.

# Проверьте ваши знания

1. Каково основное преимущество, получаемое при использовании функции?
2. Сколько значений может вернуть функция?
3. В чем разница между доступом к переменной по имени и по ссылке?

4. Что в PHP означает термин «область видимости»?
5. Как можно включить один файл PHP в другой?
6. Чем объект отличается от функции?
7. Как в PHP создаются новые объекты?
8. Какой синтаксис используется для создания подкласса из существующего класса?
9. Как можно вызвать инициализирующую часть кода при создании объекта?
0. Почему объявлять свойства внутри класса лучше явным образом?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 5».

# 6 Массивы в PHP

В главе 3 у нас уже состоялось краткое знакомство с массивами в PHP, позволившее составить первичное представление об их возможностях. В данной главе будет показан большой арсенал приемов работы с массивами, некоторые из них при наличии у вас опыта работы с языками со строгой типизацией, например C, могут удивить своей простотой и изяществом.

Массивы — одна из составляющих популярности PHP. Кроме того, что они не дают умереть со скуки при создании кода для работы со сложными структурами данных, они также предоставляют множество невероятно быстрых способов доступа к данным.

## Основные подходы к массивам

Массивы уже рассматривались в виде группы склеенных вместе спичечных коробков. Их можно представить также в виде нитки бус, где бусины обозначают переменные, которые могут быть числовыми, строковыми и даже другими массивами. Массивы похожи на нитки бус, потому что каждый элемент имеет собственное место и у каждого элемента (кроме первого и последнего) с обеих сторон есть другие элементы.

Часть массивов использует ссылки по числовым индексам, другая — позволяет работать с буквенно-цифровыми идентификаторами. Встроенные функции дают возможность проводить сортировку, добавлять и удалять отрезки и перебирать элементы для обработки каждого из них, используя специальный вид цикла. А за счет размещения одного или нескольких массивов внутри других массивов можно создавать массивы любой размерности.

## Массивы с числовой индексацией

Представим, что перед вами поставлена задача создать простой веб-сайт для компании по поставке товаров для офиса и сейчас вы ведете разработку его раздела, в котором представлены различные сорта бумаги. Как вариант, различные единицы хранения этой категории можно поместить в числовой массив, чтобы получить возможность управлять ими. Простейший способ реализации этого подхода показан в примере 6.1.

**Пример 6.1.** Добавление элементов в массив

```
<?php
$paper[] = "Copier";
$paper[] = "Inkjet";
$paper[] = "Laser";
$paper[] = "Photo";

print_r($paper);
?>
```

В этом примере при каждом присваивании массиву `$paper` значения для хранения последнего используется первое же свободное место, а значение существующего в PHP внутреннего указателя увеличивается на единицу, чтобы указывать на свободное место, готовое для следующей вставки значения. Уже известная нам функция `print_r` (которая выводит на экран содержимое переменной, массива или объекта) применяется для проверки правильности заполнения массива. Результат ее работы имеет следующий вид:

```
Array
(
    [0] => Copier
    [1] => Inkjet
    [2] => Laser
    [3] => Photo
)
```

Предыдущий код может быть написан и так, как показано в примере 6.2, где для каждого элемента указывается точное место в массиве. Но, как видите, такой подход требует набора лишних символов и усложняет обслуживание кода в том случае, если будет необходимо вставлять товары в массив или удалять их оттуда. Поэтому, если не нужно указывать какой-нибудь другой порядок размещения элементов в массиве, лучше все же позволить PHP самостоятельно заниматься их расстановкой.

**Пример 6.2.** Добавление в массив элементов с конкретным указанием их мест

```
<?php
$paper[0] = "Copier";
$paper[1] = "Inkjet";
$paper[2] = "Laser";
$paper[3] = "Photo";

print_r($paper);
?>
```

Этот пример выведет такую же информацию, как и предыдущий, но в разрабатываемом веб-сайте вы вряд ли будете пользоваться функцией `print_r`, поэтому в примере 6.3 показано, как с помощью цикла можно распечатать сведения о различных типах бумаги, предлагаемых на веб-сайте.

**Пример 6.3.** Добавление элементов в массив и извлечение их из массива

```
<?php
$paper[] = "Copier";
```

```
$paper[] = "Inkjet";
$paper[] = "Laser";
$paper[] = "Photo";

for ($j = 0 ; $j < 4 ; ++$j)
    echo "$j: $paper[$j]<br>";
?>
```

Этот пример выведет следующую информацию:

```
0: Copier
1: Inkjet
2: Laser
3: Photo
```

Итак, вы увидели два способа добавления элементов к массиву и один из способов ссылки на них, но PHP предлагает и много других способов, на которых я кратко остановлюсь в дальнейшем. Сначала рассмотрим другой тип массива.

## Ассоциативные массивы

Конечно, можно отслеживать элементы массива по индексам, но тогда придется помнить, какой именно номер на какой товар ссылается. Кроме того, за вашим кодом трудно будет уследить другим программистам.

Самое время обратиться к ассоциативным массивам. Использование этих массивов позволяет ссылаться на элементы массива по именам, а не по номерам. В примере 6.4 приводится расширенная версия предыдущего кода, где каждому элементу массива дается имя для идентификации и более длинное и информативное строковое значение.

**Пример 6.4.** Добавление элементов к ассоциативному массиву и извлечение этих элементов

```
<?php
$paper['copier'] = "Copier & Multipurpose";
$paper['inkjet'] = "Inkjet Printer";
$paper['laser']  = "Laser Printer";
$paper['photo']  = "Photographic Paper";

echo $paper['laser'];
?>
```

Теперь у каждого элемента вместо числа (не содержащего никакой полезной информации, кроме позиции элемента в массиве) имеется уникальное имя, по которому на него можно сослаться где-нибудь в другом месте, как в случае с инструкцией echo, которая выводит на экран `Laser Printer`. Имена (`copier`, `inkjet` и т. д.) называются *индексами*, или *ключами*, а присвоенные им элементы (например, `Laser Printer`) — *значениями*.

Это весьма мощное свойство PHP часто используется при извлечении информации из кода XML и HTML. Например, HTML-парсер, используемый в поисковой системе, может помещать все элементы веб-страницы в ассоциативный массив, имена которого отображают структуру страницы:

```
$html['title'] = "Моя веб-страница";
$html['body']  = "... тело веб-страницы ...";
```

Вполне вероятно, что программа разобьет все найденные на странице ссылки и поместит их в другой массив, а все заголовки и подзаголовки — еще в один массив. При использовании ассоциативных, а не числовых массивов код, ссылающийся на все эти элементы, проще будет создавать и отлаживать.

## Присваивание, использующее ключевое слово array

Мы уже видели, как элементам массива присваиваются значения путем последовательного добавления к этому массиву новых элементов. Но это слишком затянутый процесс, независимо от того, что при этом происходит: вы определяете ключи, числовые идентификаторы или позволяете PHP неявным образом заниматься присваиванием числовых идентификаторов. Есть более краткий и быстрый способ присваивания значений с использованием ключевого слова array. В примере 6.5 показаны оба массива, числовой и ассоциативный, значения которым присваиваются именно этим способом.

**Пример 6.5.** Добавление элементов к массиву с использованием ключевого слова array

```
<?php
$p1 = array("Copier", "Inkjet", "Laser", "Photo");

echo "Элемент массива p1: " . $p1[2] . "<br>";

$p2 = array('copier' => "Copier & Multipurpose",
            'inkjet' => "Inkjet Printer",
            'laser'  => "Laser Printer",
            'photo'  => "Photographic Paper");

echo "Элемент массива p2: " . $p2['inkjet'] . "<br>";
?>
```

В первой части этого кодового фрагмента массиву $p1 присваивается старое, укороченное описание товара. Здесь используются четыре элемента, поэтому они занимают позиции от 0 до 3. Инструкция echo выводит следующий текст:

```
Элемент массива p1: Laser
```

Во второй части кода массиву $p2 присваиваются ассоциативные идентификаторы и сопутствующие им длинные описания товаров. Для этого используется формат *индекс* => *значение*. Применение оператора => похоже на использование простого оператора присваивания =, за исключением того, что значение присваивается *индексу*, а не *переменной*. После этого индекс приобретает неразрывную связь с этим значением до тех пор, пока ему не будет присвоено другое значение. Поэтому команда echo выводит следующий текст:

```
Элемент массива p2: Inkjet Printer
```

В том, что $p1 и $p2 принадлежат к разным типам массивов, можно убедиться, если вставить в код две следующие команды, вызывающие ошибку неопределенного индекса (undefined index) или ошибку неопределенного смещения (undefined offset), поскольку для каждого из массивов используется неподходящий идентификатор:

```
echo $p1['inkjet']; // Неопределенный индекс
echo $p2['3'];      // Неопределенное смещение
```

# Цикл foreach...as

Создатели PHP постарались сделать этот язык простым в использовании. Поэтому они не остановились на уже имеющихся структурах организации цикла, а добавили еще одну структуру, специально предназначенную для массивов, — цикл foreach...as. Используя этот цикл, можно поочередно перебрать все элементы массива и произвести с ними какие-нибудь действия.

Процесс начинается с первого элемента и заканчивается последним, поэтому вам даже не нужно знать, сколько элементов присутствует в массиве. В примере 6.6 показано, как цикл foreach может использоваться для переписывания кода примера 6.3.

**Пример 6.6.** Последовательный перебор элементов числового массива, использующий цикл foreach...as

```
<?php
$paper = array("Copier", "Inkjet", "Laser", "Photo");
$j = 0;

foreach ($paper as $item)
{
    echo "$j: $item<br>";
    ++$j;
}
?>
```

Когда PHP встречает инструкцию foreach, он извлекает первый элемент массива и помещает его значение в переменную, указанную после ключевого слова as, и при каждом возвращении управления инструкции foreach в эту переменную помещается значение следующего элемента массива. В данном случае переменной $item присваиваются по очереди все четыре значения, хранящиеся в массиве $paper. Как только будут использованы все значения, выполнение цикла завершается. Этот код выводит точно такую же информацию, что и код примера 6.3.

Теперь посмотрим, как foreach работает с ассоциативным массивом. В примере 6.7 переписана вторая часть примера 6.5.

**Пример 6.7.** Последовательный перебор элементов ассоциативного массива, использующий цикл foreach...as

```
<?php
$paper = array('copier' => "Copier & Multipurpose",
```

```
                'inkjet' => "Inkjet Printer",
                'laser'  => "Laser Printer",
                'photo'  => "Photographic Paper");

foreach ($paper as $item => $description)
    echo "$item: $description<br>";
?>
```

Вспомним, что ассоциативным массивам не требуются числовые индексы, поэтому переменная `$j` в данном примере не используется. Вместо этого каждый элемент массива `$paper` вводится в пару «ключ — значение», представленную переменными `$item` и `$description`, из которых эта пара выводится на экран в следующем виде:

```
copier: Copier & Multipurpose
inkjet: Inkjet Printer
laser: Laser Printer
photo: Photographic Paper
```

В качестве альтернативы синтаксису `foreach...as` можно воспользоваться функцией `list` в сочетании с функцией `each` (пример 6.8).

**Пример 6.8.** Последовательный перебор элементов ассоциативного массива с помощью функций each и list

```
<?php
$paper = array('copier' => "Copier & Multipurpose",
               'inkjet' => "Inkjet Printer",
               'laser'  => "Laser Printer",
               'photo'  => "Photographic Paper");

while (list($item, $description) = each($paper))
    echo "$item: $description<br>";
?>
```

В этом примере организуется цикл `while`, который будет продолжать работу до тех пор, пока функция `each` не вернет значение `FALSE`. Функция `each` ведет себя как `foreach`: она возвращает из массива `$paper` массив, содержащий пару «ключ — значение», а затем перемещает встроенный указатель на следующую пару в исходном массиве. Когда возвращать станет нечего, функция `each` возвращает значение `FALSE`.

Функция `list` в качестве аргументов принимает массив (в данном случае пару «ключ — значение», возвращенную функцией `each`), а затем присваивает значения массива переменным, перечисленным внутри круглых скобок.

Лучше понять работу функции `list` можно из примера 6.9, где массив создается из двух строк — `Alice` и `Bob`, а затем передается функции `list`, которая присваивает эти строки переменным `$a` и `$b`.

**Пример 6.9.** Использование функции list

```
<?php
list($a, $b) = array('Alice', 'Bob');
echo "a=$a b=$b";
?>
```

Этот код выводит следующий текст:

```
a=Alice b=Bob
```

Итак, для перебора элементов массива можно использовать различные подходы. Можно воспользоваться конструкцией foreach...as для создания цикла, извлекающего значения в переменную, которая следует за as, или воспользоваться функцией each и создать собственную систему циклической обработки.

# Многомерные массивы

Простая конструктивная особенность синтаксиса массивов PHP позволяет создавать массивы более чем с одним измерением. Фактически можно создавать массивы какой угодно размерности (хотя приложения редко нуждаются в массивах с размерностью больше трех).

Эта особенность заключается в возможности включать целый массив в состав другого массива, а также делать это снова и снова, как в старом стишке про блох, которых кусают другие блохи поменьше, а тех, в свою очередь, кусают свои блохи, и так до бесконечности.

Рассмотрим, как это работает, для чего возьмем ассоциативный массив из предыдущего примера и расширим его (пример 6.10).

**Пример 6.10.** Создание многомерного ассоциативного массива

```
<?php
$products = array(
    'paper' =>     array(
        'copier' => "Copier & Multipurpose",
        'inkjet' => "Inkjet Printer",
        'laser'  => "Laser Printer",
        'photo'  => "Photographic Paper"),

    'pens' => array(
        'ball'   => "Ball Point",
        'hilite' => "Highlighters",
        'marker' => "Markers"),

    'misc' => array(
        'tape'   => "Sticky Tape",
        'glue'   => "Adhesives",
        'clips'  => "Paperclips") );

echo "<pre>";
foreach ($products as $section => $items)
    foreach ($items as $key => $value)
        echo "$section:\t$key\t($value)<br>";
echo "</pre>";
?>
```

Чтобы упростить понимание начинающего разрастаться кода, я переименовал часть элементов. Например, рассматривавшийся ранее массив `$paper` стал всего лишь подразделом более крупного массива, а главный массив теперь называется `$products`. В этом массиве присутствуют три элемента: бумага — `paper`, ручки — `pens` и разные товары — `misc`, и каждый из них содержит другой массив, состоящий из пар «ключ — значение».

При необходимости эти подмассивы могут содержать другие массивы. Например, элемент шариковые ручки — `ball` может содержать множество типовых и цветовых решений этого товара, имеющихся в интернет-магазине. Но пока я ограничил код глубиной в два измерения.

После присвоения массивам данных для вывода различных значений я воспользовался парой вложенных циклов `foreach...as`. Внешний цикл извлекает из верхнего уровня массива основные разделы, а внутренний цикл извлекает для категорий в каждом разделе пары «ключ — значение».

Если вспомнить, что все уровни массива работают одинаково (являясь парой «ключ — значение»), можно без особого труда создать код для доступа к любому элементу на любом уровне.

В инструкции `echo` используется управляющий символ PHP `\t`, который выводит знак табуляции.

Хотя знаки табуляции для веб-браузеров, как правило, ничего не значат, я использовал их в разметке, применив теги `<pre>...</pre>`, которые предписывают веб-браузеру форматировать текст с сохранением предварительного формата и фиксированной ширины и не игнорировать неотображаемые символы вроде знаков табуляции и переводов строки. Текст, выводимый этим кодом, будет иметь следующий вид:

```
paper:    copier    (Copier & Multipurpose)
paper:    inkjet    (Inkjet Printer)
paper:    laser     (Laser Printer)
paper:    photo     (Photographic Paper)
pens:     ball      (Ball Point)
pens:     hilite    (Highlighters)
pens:     marker    (Markers)
misc:     tape      (Sticky Tape)
misc:     glue      (Adhesives)
misc:     clips     (Paperclips)
```

Непосредственный доступ к конкретному элементу массива можно получить, используя квадратные скобки:

```
echo $products['misc']['glue'];
```

Этот код выводит значение `Adhesives`.

Можно также создать числовой многомерный массив, непосредственный доступ к элементам которого можно будет получать по индексам, а не по буквенно-цифровым идентификаторам. В примере 6.11 создается шахматная доска с фигурами на исходных позициях.

**Пример 6.11.** Создание многомерного числового массива

```
<?php
$chessboard = array(
    array('r', 'n', 'b', 'k', 'q', 'b', 'n', 'r'),
    array('p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array('P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'),
    array('R', 'N', 'B', 'K', 'Q', 'B', 'N', 'R'));

echo "<pre>";
foreach ($chessboard as $row)
{
    foreach ($row as $piece)
        echo "$piece ";
    echo "<br />";
}
echo "</pre>";
?>
```

В этом примере буквы в нижнем регистре представляют собой черные фигуры, а верхнем регистре — белые. Используются следующие обозначения: r — *rook* (ладья), n — *knight* (конь), b — *bishop* (слон), k — *king* (король), q — *queen* (ферзь) и p — *pawn* (пешка). Для последовательного перебора массива и демонстрации его содержимого снова используется пара вложенных циклов foreach...as. Внешний цикл обрабатывает каждую горизонталь и помещает ее в переменную $row, которая сама по себе является массивом, поскольку для каждой горизонтали массив шахматной доски — $chessboard использует подмассив. В этом цикле используются две инструкции, поэтому они заключены в фигурные скобки.

Внутренний цикл обрабатывает каждую клетку горизонтали, выводя хранящийся в ней символ ($piece), за которым следует пробел (чтобы выводимый текст имел форму шахматной доски). У этого цикла одна инструкция, которую не нужно заключать в фигурные скобки. Теги <pre> и </pre> обеспечивают правильную форму выводимого текста:

```
r n b k q b n r
p p p p p p p p



P P P P P P P P
R N B K Q B N R
```

Используя квадратные скобки, можно получить непосредственный доступ к любому элементу этого массива:

```
echo $chessboard[7][4];
```

Эта инструкция выведет букву Q в верхнем регистре, которая является значением восьмого вниз по вертикали и пятого по горизонтали элемента (следует помнить, что индексы массива начинаются с нуля, а не с единицы).

# Использование функций для работы с массивами

С функциями `list` и `each` вы уже знакомы, но в PHP имеется множество других функций, предназначенных для работы с массивами. Их полный перечень представлен по адресу http://tinyurl.com/phparrayfuncs. Но некоторые из этих функций играют настолько важную роль в программировании на PHP, что мы изучим их подробнее.

## is_array

Массивы и переменные используют одно и то же пространство имен. Это означает, что нельзя иметь строковую переменную по имени `$fred` и массив, который также называется `$fred`. Если есть сомнения и в коде программы нужно проверить, является ли переменная массивом, можно воспользоваться функцией `is_array`:

```
echo (is_array($fred)) ? "Это массив" : "Это не массив";
```

Заметьте, что переменной `$fred` не присвоено никакого значения, поэтому будет выведено сообщение о неопределенной переменной — Undefined variable.

## count

Несмотря на то что функция `each` и структура организации цикла `foreach...as` предоставляют отличные способы последовательного перебора всего содержимого массива, иногда нужно точно знать, сколько элементов содержится в вашем массиве, особенно если вы будете обращаться к ним напрямую. Для подсчета всех элементов на верхнем уровне массива используется следующая команда:

```
echo count($fred);
```

Если нужно узнать, сколько всего элементов содержится в многомерном массиве, можно воспользоваться следующей инструкцией:

```
echo count($fred, 1);
```

Второй параметр является необязательным и устанавливает режим использования. Он может иметь либо нулевое значение, чтобы ограничить подсчет только верхним уровнем, либо единичное — для принудительного включения рекурсивного подсчета еще и всех элементов, содержащихся в подмассивах.

## sort

Сортировка является настолько распространенной операцией, что PHP предоставляет для нее встроенную функцию. В наипростейшей форме ее можно использовать следующим образом:

```
sort($fred);
```

В отличие от некоторых других функций, сортировка будет работать непосредственно с предоставленным ей массивом, а не возвращать новый массив с отсортированными элементами. Вместо этого она вернет значение TRUE при успешном выполнении сортировки и FALSE — в случае возникновения ошибки. Эта функция поддерживает также несколько флагов. Основные два, которые вам могут пригодиться, предписывают проведение либо числовой, либо строковой сортировки:

```
sort($fred, SORT_NUMERIC);
sort($fred, SORT_STRING);
```

Массив можно также отсортировать в обратном порядке, воспользовавшись функцией rsort:

```
rsort($fred, SORT_NUMERIC);
rsort($fred, SORT_STRING);
```

## shuffle

Иногда, например при создании игры или при игре в карты, требуется, чтобы элементы массива располагались в случайном порядке:

```
shuffle($cards);
```

Как и функция sort, функция shuffle работает непосредственно с предоставленным ей массивом и возвращает значение TRUE в случае успешного завершения работы и FALSE — при возникновении ошибки.

## explode

Это очень полезная функция, позволяющая взять строку, содержащую несколько элементов, отделенных друг от друга одиночным символом (или строкой символов), а затем поместить каждый из этих элементов в массив. В примере 6.12 показан один из случаев полезного применения этой функции, который заключается в разбиении предложения на слова и помещении всех слов, из которого оно состоит, в массив.

**Пример 6.12.** Извлечение слов из строки в массив с использованием пробелов

```
<?php
$temp = explode(' ', "Это предложение из пяти слов");
print_r($temp);
?>
```

Этот пример выводит следующую информацию (которая при просмотре в браузере будет отображена в одной строке):

```
Array
(
    [0] => Это
    [1] => предложение
    [2] => из
```

```
    [3] => пяти
    [4] => слов
)
```

Первый параметр, разделитель, не обязательно должен быть пробелом или даже одиночным символом. В примере 6.13 показан этот же код в несколько измененном виде.

**Пример 6.13.** Извлечение слов, разделенных символами ***, из строки в массив

```
<?php
$temp = explode('***', "Это***предложение***со***звездочками");
print_r($temp);
?>
```

Код примера 6.13 выводит следующую информацию:

```
Array
(
    [0] => Это
    [1] => предложение
    [2] => со
    [3] => звездочками
)
```

## extract

Иногда бывает удобно превратить пары «ключ — значение» из массива в переменные PHP. Один из таких случаев — это обработка переменных $_GET или $_POST, отправленных формой сценарию PHP.

Когда форма передается через Интернет, веб-сервер распаковывает переменные и помещает их в глобальный массив, предназначенный для сценария PHP. Если переменные были отправлены методом GET, они будут помещены в ассоциативный массив $_GET, а при отправке методом POST они будут помещены в ассоциативный массив $_POST.

Разумеется, можно перебрать все элементы этих ассоциативных массивов, воспользовавшись уже рассмотренными в этой главе способами. Но иногда нужно лишь сохранить отправленные значения в переменных для дальнейшего использования. В таком случае можно заставить PHP проделать эту работу за вас в автоматическом режиме:

```
extract($_GET);
```

Таким образом, к примеру, если параметр строки запроса q отправлен сценарию PHP наряду со связанным с ним значением Hi there, будет создана новая переменная по имени $q, которой будет присвоено это значение.

Но к этому подходу нужно относиться осторожно, поскольку если какие-нибудь извлекаемые переменные конфликтуют с уже определенными переменными, то существующие переменные будут переписаны. Чтобы избежать этого, можно воспользоваться одним из многих дополнительных параметров, доступных в данной функции:

```
extract($_GET, EXTR_PREFIX_ALL, 'fromget');
```

В этом случае имена всех новых переменных будут начинаться с заданного строкового префикса, за которым следует символ подчеркивания, в результате чего $q превратится в $fromget_q. Я настоятельно рекомендую при обработке массивов $_GET и $_POST или любого другого массива, ключи которого могут контролироваться пользователем, использовать именно эту версию функции. Поскольку злоумышленники могут отправлять ключи, специально подобранные для того, чтобы переписать переменные с часто используемыми именами и таким образом угрожать вашему веб-сайту.

## compact

Иногда нужно воспользоваться функцией compact, которая является противоположностью функции extract, чтобы создать массив из переменных и их значений. Применение этой функции показано в примере 6.14.

**Пример 6.14.** Использование функции compact

```
<?php
$fname   = "Elizabeth";
$sname   = "Windsor";
$address = "Buckingham Palace";
$city    = "London";
$country = "United Kingdom";

$contact = compact('fname', 'sname', 'address', 'city', 'country');
print_r($contact);
?>
```

В результате запуска кода из примера 6.14 будет выведена следующая информация:

```
Array
(
    [fname] => Elizabeth
    [sname] => Windsor
    [address] => Buckingham Palace
    [city] => London
    [country] => United Kingdom
)
```

Обратите внимание на то, что функции compact нужны имена переменных, стоящие в кавычках и не содержащие начального символа $. Причина заключается в том, что функция compact определяет массив с именами переменных.

Эту функцию можно использовать также для отладки, когда нужно быстро просмотреть несколько переменных вместе с их значениями, как в примере 6.15.

**Пример 6.15.** Использование функции compact для отладки программы

```
<?php
$j       = 23;
$temp    = "Hello";
$address = "1 Old Street";
```

```
$age     = 61;

print_r (compact (explode (' '.'j temp address age')));
?>
```

Работа примера основана на использовании функции explode для извлечения всех слов из строки в массив, который затем передается функции compact, а она, в свою очередь, возвращает массив функции print_r, показывающей его содержимое.

Если скопировать и вставить строку кода, содержащую вызов функции print_r, то в ней нужно будет лишь изменить имена переменных, чтобы быстро вывести группу их значений. В этом примере выводимая информация будет иметь следующий вид:

```
Array
(
    [j] => 23
    [temp] => Hello
    [address] => 1 Old Street
    [age] => 61
)
```

## reset

Когда с помощью конструкции foreach...as или функции each осуществляется последовательный перебор элементов массива, они перемещают внутренний указатель PHP, который показывает, какой из элементов массива следует извлечь в следующий раз. Если коду программы понадобится вернуться к началу массива, то можно воспользоваться функцией reset, которая к тому же вернет значение элемента, на котором остановился указатель. Эта функция может быть использована следующим образом:

```
reset($fred);          // Отбрасывание возвращаемого значения
$item = reset($fred); // Сохранение первого элемента массива
                       // в переменной $item
```

## end

Можно также переместить внутренний указатель элемента массива PHP на самый последний элемент, воспользовавшись для этого функцией end, которая, кроме этого, возвращает значение элемента и может быть использована следующим образом:

```
end($fred);
$item = end($fred);
```

В этой главе завершается введение в основы PHP. Теперь, используя приобретенные навыки, вы должны справиться с написанием довольно сложных программ. В следующей главе будет рассмотрено применение PHP для решения наиболее распространенных практических задач.

# Проверьте ваши знания

1. В чем разница между числовым и ассоциативным массивом?
2. Каковы основные преимущества использования ключевого слова `array`?
3. В чем разница между `foreach` и `each`?
4. Как создается многомерный массив?
5. Как определить количество элементов в массиве?
6. Каково назначение функции `explode`?
7. Как вернуть внутренний указатель элемента массива PHP на его первый элемент?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 6».

# 7 Практикум по программированию на PHP

В предыдущих главах рассматривались элементы языка PHP. А эта глава предназначена для приобретения навыков программирования в процессе решения типовых, но тем не менее важных практических задач.

Здесь будут представлены лучшие способы обработки строк, позволяющие получить вполне понятный и лаконичный код, включающий усовершенствованное управление отображением даты и времени, демонстрируемый веб-браузерами в точном соответствии с вашими желаниями. Вы также узнаете о создании и разнообразных способах изменения файлов, включая файлы, выложенные на сайт пользователями.

В этой главе будет также предоставлено полноценное введение в XHTML, язык разметки, похожий на HTML, и предназначенный для его замены (согласующийся с синтаксисом XML, который используется для хранения данных, например RSS-потоков). В совокупности все это расширит вашу осведомленность как в области практического программирования на PHP, так и в сфере развития международных веб-стандартов.

## Использование функции printf

Ранее нам уже встречались функции `print` и `echo`, которые использовались для простого вывода текста в браузер. Но существует намного более мощная функция `printf`, управляющая форматом выводимых данных путем вставки в строку специальных форматирующих символов. Функция `printf` ожидает, что для каждого форматирующего символа будет предоставлен аргумент, который будет отображаться с использованием заданного формата. Например, в следующем фрагменте применяется спецификатор преобразования `%d`, чтобы значение 3 отображалось в виде десятичного числа:

```
printf("В вашей корзине находится %d покупки", 3);
```

Если заменить `%d` на `%b`, значение 3 будет отображено в виде двоичного числа (11). В табл. 7.1 показаны поддерживаемые функцией спецификаторы преобразования.

**Таблица 7.1.** Спецификаторы преобразования, используемые в функции printf

| Спецификатор | Преобразование, осуществляемое с аргументом arg | Пример (для arg, имеющего значение 123) |
|---|---|---|
| % | Отображение символа % (аргументы не требуются) | % |
| b | Отображение arg в виде двоичного целого числа | 1111011 |
| c | Отображение ASCII-символа с кодом, содержащимся в arg | { |
| d | Отображение arg в виде целого десятичного числа со знаком | 123 |
| e | Отображение arg с использованием научной формы записи | 1.23000e+2 |
| f | Отображение arg в виде числа с плавающей точкой | 123.000000 |
| o | Отображение arg в виде восьмеричного целого числа | 173 |
| s | Отображение arg в виде строки | 123 |
| u | Отображение arg в виде беззнакового десятичного числа | 123 |
| x | Отображение arg в виде шестнадцатеричного числа с символами в нижнем регистре | 7b |
| X | Отображение arg в виде шестнадцатеричного числа с символами в верхнем регистре | 7B |

В функции `printf` можно использовать любое количество спецификаторов, если им передается соответствующее количество аргументов и если каждый спецификатор предваряется символом `%`. Поэтому следующий код имеет вполне допустимую форму и выводит предложение: «Меня зовут Симон. Мне 33 года, то есть 21 в шестнадцатеричном представлении»:

```
printf("Меня зовут %s. Мне %d года, то есть %X в шестнадцатеричном представлении",
    'Simon', 33, 33);
```

Если пропустить какой-нибудь аргумент, то будет получена ошибка синтаксического разбора, информирующая о том, что правая круглая скобка, `)`, была встречена в неожиданном месте.

Более полезный с практической точки зрения пример использования функции `printf` устанавливает цвета в коде HTML, используя для этого десятичные числа. Предположим, к примеру, что вам нужен цвет, составленный из трех значений: 65 для красного, 127 для зеленого и 245 для синего цвета, но вам не хочется самостоятельно переводить эти числа в шестнадцатеричный формат. Для этого есть более простое решение:

```
printf("<font color='#%X%X%X'>Привет</font>", 65, 127, 245);
```

Тщательно разберитесь с цветовой спецификацией, которая заключена в апострофы (`''`). Сначала ставится знак решетки (#), ожидаемый в цветовой спецификации. Затем следуют три форматирующие спецификации `%X`, по одной для каждого из ваших чисел. В результате эта команда выдаст следующий текст:

```
<font color='#417FF5'>Привет</font>
```

Обычно представляется удобным в качестве аргументов printf использовать переменные или выражения. Например, если значения для цветового решения хранятся в трех переменных — $r, $g и $b, то более темный оттенок можно получить с помощью выражения:

```
printf("<font color='#%X%X%X'>Привет</font>", $r-20, $g-20, $b-20);
```

## Настройка представления данных

Можно указать не только тип преобразования, но и точность отображаемого результата. Например, суммы в валюте отображаются, как правило, с точностью до двух цифр. Но после вычисления значение может иметь более высокую точность (например, если разделить $123,42 на 12, получится $10,285). Чтобы обеспечить правильное внутреннее хранение таких значений, но при этом организовать их отображение с точностью только до двух цифр, можно между символом % и спецификатором преобразования вставить строку .2:

```
printf("Результат: $%.2f", 123.42 / 12);
```

Эта команда выводит следующий текст:

```
Результат: $10.29
```

Но доступные средства управления на этом не заканчиваются, потому что можно также указать, где и чем — нулями или пробелами — дополнить выводимый текст, поставив перед спецификатором соответствующие значения. В примере 7.1 показаны пять возможных комбинаций.

**Пример 7.1.** Настройка преставления данных точности

```
<?php
echo "<pre>"; // Тег, позволяющий отображать все пустые пространства

// Дополнение пробелами до 15 знако-мест
   printf("Результат равен $%15f\n", 123.42 / 12);

// Дополнение нулями до 15 знако-мест
   printf("Результат равен $%015f\n", 123.42 / 12);

// Дополнение пробелами до 15 знако-мест и вывод с точностью до двух
// десятичных знаков
   printf("Результат равен $%15.2f\n", 123.42 / 12);

// Дополнение нулями до 15 знако-мест и вывод с точностью до двух
// десятичных знаков
   printf("Результат равен $%015.2f\n", 123.42 / 12);

// Дополнение символами # до 15 знако-мест и вывод с точностью до двух
   десятичных знаков
   printf("Результат равен $%'#15.2f\n", 123.42 / 12);
?>
```

Этот пример выводит следующий текст:

```
Результат равен $      10.285000
Результат равен $00000010.285000
Результат равен $         10.29
Результат равен $000000000010.29
Результат равен $#########10.29
```

Проследить работу спецификатора проще, если изучать его слева направо (табл. 7.2). Обратите внимание на следующие моменты.

- Самым правым символом спецификатора преобразования в данном случае является f, означающий преобразование в число с плавающей точкой.
- Если сразу же перед спецификатором преобразования стоит сочетание точки и числа, значит, этим числом указана точность выводимой информации.
- Независимо от присутствия спецификатора точности, если в общем спецификаторе есть число, то оно представляет собой количество знако-мест, выделяемых под выводимую информацию. В предыдущем примере это количество равно 15. Если выводимая информация уже равна количеству выделяемых знако-мест или превышает его, данный аргумент игнорируется.
- Перед самым левым символом % разрешается поставить символ 0, который игнорируется, если не указано количество выделяемых знако-мест. Если это количество указано, то вместо пробелов дополнение производится нулями. Если нужно, чтобы пустующие знако-места заполнялись не нулями или пробелами, а каким-нибудь другим символом, то можно выбрать любой символ, поставив перед ним одинарную кавычку: '#.
- В левой части спецификатора ставится символ %, с позиции которого и начинается преобразование.

**Таблица 7.2.** Компоненты спецификатора преобразования

| Начало преобразования | Дополняющий символ | Количество дополняющих символов | Точность отображения | Спецификатор преобразования | Примеры |
|---|---|---|---|---|---|
| % | | 15 | | f | 10.285000 |
| % | 0 | 15 | .4 | f | 000000000010.29 |
| % | '# | 15 | .2 | f | ########10.2850 |

# Дополнение строк

Дополнить до требуемой длины можно не только числа, но и строки, выбирая для этого различные дополняющие символы и даже левую или правую границы выравнивания. Возможные варианты показаны в примере 7.2.

**Пример 7.2.** Дополнение строк

```
<?php
echo "<pre>"; // Тег, позволяющий отображать все пустые пространства
```

```
$h = 'House';

printf("[%s]\n",        $h); // Стандартный вывод строки
printf("[%10s]\n",      $h); // Выравнивание пробелами по правому краю
printf("[%-10s]\n",     $h); // Выравнивание пробелами по левому краю
printf("[%010s]\n",     $h); // Дополнение нулями
printf("[%'#10s]\n\n",  $h); // Использование специально выбранного символа
                             // дополнения '#'

$d = 'Doctor House';

printf("[%10.8s]\n",    $d); // Выравнивание по правому краю с усечением
                             // до 8 символов
printf("[%-10.6s]\n",   $d); // Выравнивание по левому краю с усечением
                             // до 6 символов
printf("[%-'@10.6s]\n", $d); // Выравнивание по левому краю, дополнение
                             // символом '@', усечение до 6 символов
?>
```

Обратите внимание на то, что для получения нужной разметки на веб-странице я воспользовался HTML-тегом <pre>, который оставляет нетронутыми все пустые пространства и после каждой отображаемой строки выводит на экран символ новой строки \n. В этом примере выводится следующий текст:

```
[House]
[     House]
[House     ]
[00000House]
[#####House]
[  Doctor H]
[Doctor    ]
[Doctor@@@@]
```

Если при указании количества знако-мест длина строки уже равна этому количеству или превышает его, это указание будет проигнорировано, *если только* заданное количество символов, до которого нужно усечь строку, не будет меньше указанного количества знако-мест.

В табл. 7.3 показаны спецификаторы преобразования строки, разложенные на компоненты.

**Таблица 7.3.** Компоненты спецификаторов преобразования строки

| Начало преобразования | Выравнивание по левому или по правому краю | Дополняющий символ | Количество дополняющих символов | Усечение | Спецификатор преобразования | Примеры |
|---|---|---|---|---|---|---|
| % | | | | | s | [House] |
| % | – | | 10 | | s | [House ] |
| % | | '# | 8 | .4 | s | [####Hous] |

## Использование функции sprintf

Зачастую результат преобразования нужно не выводить на экран, а использовать в самом коде программы. Для этого предназначена функция `sprintf`. Она позволяет не отправлять выходную информацию браузеру, а присваивать ее какой-нибудь переменной.

Функции `sprintf` можно использовать просто для преобразования, возвращающего шестнадцатеричное строковое значение для цветового сочетания RGB 65, 127, 245, которое присваивается переменной `$hexstring`:

```
$hexstring = sprintf("%X%X%X", 65, 127, 245);
```

Или же она может пригодиться для сохранения выходной информации, которую нужно будет вывести на экран чуть позже:

```
$out = sprintf("Результат: $%.2f", 123.42 / 12);
echo $out;
```

# Функции даты и времени

Для отслеживания даты и времени в PHP используются стандартные отметки времени Unix, представляющие собой простое количество секунд, прошедших с начала отсчета — 1 января 1970 года. Для определения текущей отметки времени можно воспользоваться функцией `time`:

```
echo time();
```

Поскольку значение хранится в секундах, для получения метки времени ровно через неделю можно воспользоваться следующим выражением, в котором к возвращаемому значению прибавляется 7 дней × 24 часа × 60 минут × 60 секунд:

```
echo time() + 7 * 24 * 60 * 60;
```

Если нужно получить отметку времени для заданной даты, можно воспользоваться функцией `mktime`. Она выводит отметку времени `946684800` для первой секунды первой минуты первого часа первого дня 2000 года:

```
echo mktime(0, 0, 0, 1, 1, 2000);
```

Этой функции передаются следующие параметры (слева направо):

- количество часов (0–23);
- количество минут (0–59);
- количество секунд (0–59);
- номер месяца (1–12);
- номер дня (1–31);
- год (1970–2038 или 1901–2038 при использовании PHP 5.1.0 + 32-разрядной системы со знаком числа).

Вы можете спросить, а почему годы ограничены отрезком с 1970-го по 2038-й? Причина в том, что разработчики первой версии Unix выбрали 1970 год в качестве начала отсчета времени, опускаться ниже которого не понадобится ни одному программисту. К счастью, благодаря тому, что PHP, начиная с версии 5.1.0, поддерживает системы, использующие 32-разрядные целые числа со знаком, в нем разрешается применение дат с 1901 до 2038 года. Но второе ограничение еще хуже первого и обусловлено тем, что разработчики Unix также решили, что по прошествии 70 лет никто уже не будет пользоваться их системой, и поэтому они были уверены, что для хранения отметки времени им вполне хватит 32-разрядного значения, которое будет вмещать даты только до 19 января 2038 года. Это ограничение вызовет сбой, известный как Y2K38 (очень похожий на проблему 2000 года, которая была вызвана тем, что года хранились в виде значений из двух цифр, и также требовала своего решения). Остается только надеяться на то, что ближе к этой дате будет найден приемлемый выход из положения.

Для отображения даты используется функция `date`, поддерживающая множество настроек форматирования, которые позволяют выводить дату любым желаемым способом. Эта функция имеет следующий синтаксис:

```
date($format, $timestamp);
```

Параметр `$format` должен быть строкой, в которой содержатся спецификаторы форматирования, подробно описанные в табл. 7.4, а параметр `$timestamp` должен быть отметкой времени в стандарте Unix. Полный перечень спецификаторов приведен по адресу http://tinyurl.com/phpdatefuncs. Следующая команда выведет текущее время и дату в формате «Thursday April 15th, 2010 - 1:38pm»:

```
echo date("l F jS, Y - g:ia", time());
```

**Таблица 7.4.** Основные спецификаторы формата, использующиеся в функции date

| Формат | Описание | Возвращаемое значение |
|---|---|---|
| *Спецификаторы дня* | | |
| d | День месяца, две цифры с лидирующими нулями | От 01 до 31 |
| D | День недели, составленный из трех букв | От Mon до Sun |
| j | День месяца без лидирующих нулей | От 1 до 31 |
| l | День недели полностью | От Sunday до Saturday |
| N | День недели, число, от понедельника до воскресенья | От 1 до 7 |
| S | Суффикс для дня месяца (пригодится в сочетании со спецификатором j) | st, nd, rd или th |
| w | День недели, число, от воскресенья до субботы | От 0 до 6 |
| z | День года | От 0 до 365 |
| *Спецификатор недели* | | |
| W | Номер недели в году | От 01 до 52 |
| *Спецификаторы месяца* | | |
| F | Название месяца | От January до December |
| m | Номер месяца с лидирующими нулями | От 01 до 12 |
| M | Название месяца, составленное из трех букв | От Jan до Dec |
| n | Номер месяца без лидирующих нулей | От 1 до 12 |
| t | Количество дней в заданном месяце | 28, 29, 30 или 31 |

| Формат | Описание | Возвращаемое значение |
|---|---|---|
| *Спецификаторы года* | | |
| L | Високосный год | 1 — Да, 0 — Нет |
| Y | Год, четыре цифры | От 0000 до 9999 |
| y | Год, две цифры | От 00 до 99 |
| *Спецификаторы времени* | | |
| a | До или после полудня, в нижнем регистре | am или pm |
| A | До или после полудня, в верхнем регистре | AM или PM |
| g | Час суток, 12-часовой формат без лидирующих нулей | От 1 до 12 |
| G | Час суток, 24-часовой формат без лидирующих нулей | От 1 до 24 |
| h | Час суток, 12-часовой формат с лидирующими нулями | От 01 до 12 |
| H | Час суток, 24-часовой формат с лидирующими нулями | От 01 до 24 |
| i | Минуты с лидирующими нулями | От 00 до 59 |
| s | Секунды с лидирующими нулями | От 00 до 59 |

# Константы, связанные с датами

Существуют полезные константы, которые можно использовать с командами, связанными с датами, для того чтобы они вернули дату в определенном формате. Например, `date(DATE_RSS)` возвращает текущую дату и время в формате, который используется в RSS-потоке. Наиболее часто используются следующие константы.

- `DATE_ATOM` — формат для потоков Atom. PHP-формат имеет вид «Y-m-d\TH:i:sP», а выводимая информация — «2012-08-16T12:00:00+0000».
- `DATE_COOKIE` — формат для cookie, устанавливаемый веб-сервером или JavaScript. PHP-формат имеет вид «l, d-M-y H:i:s T», а выводимая информация — «Thu, 16-Aug-2012 12:00:00 UTC».
- `DATE_RSS` — формат для потоков RSS. PHP-формат имеет вид «D, d M Y H:i:s T», а выводимая информация — «Thu, 16 Aug 2012 12:00:00 UTC».
- `DATE_W3C` — формат для консорциума Всемирной паутины, World Wide Web Consortium. PHP-формат имеет вид «Y-m-d\TH:i:sP», а выводимая информация — «2012-08-16T12:00:00+0000».

Полный перечень приведен по адресу http://tinyurl.com/phpdates.

# Использование функции checkdate

Как отобразить допустимую дату в различных форматах, вы уже видели, а как проверить, что пользователь передал такую дату вашей программе? Нужно передать месяц, день и год функции `checkdate`, которая вернет значение `TRUE`, если ей передана допустимая дата, или значение `FALSE` — в противном случае.

Например, если введена дата 30 февраля любого года, то она в любом случае будет недопустимой. В примере 7.3 показан код, который можно использовать для этой цели. В данных условиях он признает заданную дату недопустимой.

**Пример 7.3.** Проверка допустимости даты

```
<?php
$month = 9;      // Сентябрь (в котором только 30 дней)
$day   = 31;     // 31-е
$year  = 2012;   // 2012

if (checkdate($month, $day, $year)) echo "Допустимая дата";
else echo "Недопустимая дата";
?>
```

# Работа с файлами

При всех своих достоинствах MySQL не является единственным (или самым лучшим) способом хранения всех данных на веб-сервере. Иногда бывает быстрее и удобнее обращаться непосредственно к файлам, хранящимся на диске. Это может потребоваться при изменении изображений, например выложенных пользователями аватаров, или файлов регистрационных журналов, требующих обработки.

Прежде всего следует упомянуть об именах файлов. Если создается код, который может использоваться на различных установках PHP, то узнать о том, чувствительна система к регистру букв или нет, практически невозможно. Например, имена файлов в Windows и Mac OS X нечувствительны к регистру, а в Linux и Unix — чувствительны. Поэтому нужно принять за основу то, что система чувствительна к регистру, и придерживаться соглашения о присваивании файлам имен в нижнем регистре.

## Проверка существования файла

Чтобы проверить факт существования файла, можно воспользоваться функцией `file_exists`, которая возвращает либо `TRUE`, либо `FALSE` и используется следующим образом:

```
if (file_exists("testfile.txt")) echo "Файл существует";
```

## Создание файла

В данный момент файла `testfile.txt` не существует, поэтому создадим его и запишем в него несколько строк. Наберите код, показанный в примере 7.4, и сохраните его под именем `testfile.php`.

**Пример 7.4.** Создание простого текстового файла

```
<?php // testfile.php
$fh = fopen("testfile.txt", 'w') or die("Создать файл не удалось");
$text = <<<_END
```

```
Строка 1
Строка 2
Строка 3

_END;
fwrite($fh, $text) or die("Сбой записи файла");
fclose($fh);
echo "Файл 'testfile.txt' записан успешно ";
?>
```

Если этот код будет запущен через браузер, то при его успешном выполнении будет получено следующее сообщение: «Файл 'testfile.txt' записан успешно». Если будет выведено сообщение об ошибке, значит, на диске недостаточно свободного места или, что более вероятно, отсутствует разрешение на создание файла или на запись в этот файл. В таком случае нужно изменить атрибуты папки назначения в соответствии с требованиями вашей операционной системы. Если все обойдется без ошибки, то файл `testfile.txt` попадет в ту же папку, где был сохранен программный файл `testfile.php`. Если открыть файл в текстовом или программном редакторе, в нем будет следующее содержимое:

```
Строка 1
Строка 2
Строка 3
```

В этом простом примере показана последовательность работы со всеми файлами.

1. Все начинается с открытия файла с помощью вызова функции `fopen`.
2. После этого можно вызывать другие функции. В данном случае в файл велась запись (`fwrite`), но можно также читать данные из уже существующего файла (`fread` или `fgets`) и осуществлять с ним другие действия.
3. Работа завершается закрытием файла (`fclose`). Хотя программа перед завершением своей работы делает это за вас, но все же вы должны удостовериться в том, что по окончании работы с файлом он будет закрыт.

Каждому открытому файлу требуется файловый ресурс, чтобы PHP-программа могла к нему обращаться и им управлять. В предыдущем примере переменной `$fh` (которую я выбрал в качестве описателя файла) присваивается значение, возвращаемое функцией `fopen`. После этого каждой функции обработки файла, которая получает к нему доступ, например `fwrite` или `fclose`, в качестве параметра должна быть передана переменная `$fh`, чтобы идентифицировать обрабатываемый файл. Интересоваться содержимым переменной `$fh` не стоит, это всего лишь номер, используемый PHP для ссылки на внутреннюю информацию о файле. Данная переменная используется только для передачи другим функциям.

В случае сбоя функция `fopen` возвращает значение `FALSE`. В предыдущем примере показан простой способ перехвата управления и реакции на сбой: в нем вызывается функция `die`, которая завершает программу и выдает пользователю сообщение об ошибке. Это упрощенный способ выхода подходит лишь для наших учебных программ, а выходить с его помощью из веб-приложения не следует ни в коем случае (вместо этого нужно создать веб-страницу с сообщением об ошибке).

Обратите внимание на второй параметр, используемый в вызове функции fopen. Это символ w, предписывающий функции открыть файл для записи. Если такого файла нет, то он будет создан. Применять эту функцию следует с оглядкой: если файл уже существует, параметр режима работы w заставит функцию fopen удалить все его прежнее содержимое (даже если в него не будет записано ничего нового!).

В табл. 7.5 перечислены различные параметры режима работы, которые могут быть использованы при вызове этой функции.

**Таблица 7.5.** Режимы работы, поддерживаемые функцией fopen

| Режим | Действие | Описание |
|---|---|---|
| 'r' | Чтение с начала файла | Открытие файла только для чтения; установка указателя файла на его начало. Возвращение FALSE, если файла не существует |
| 'r+' | Чтение с начала файла с возможностью записи | Открытие файла для чтения и записи; установка указателя файла на его начало. Возвращение FALSE, если файла не существует |
| 'w' | Запись с начала файла с усечением его размера | Открытие файла только для записи; установка указателя файла на его начало и сокращение размера файла до нуля. Если файла не существует, попытка его создания |
| 'w+' | Запись с начала файла с усечением его размера и возможностью чтения | Открытие файла для чтения и записи; установка указателя файла на его начало и сокращение его размера до нуля. Если файла не существует, попытка его создания |
| 'a' | Добавление к концу файла | Открытие файла только для записи; установка указателя файла на его конец. Если файла не существует, попытка его создания |
| 'a+' | Добавление к концу файла с возможностью чтения | Открытие файла для чтения и записи; установка указателя файла на его конец. Если файла не существует, попытка его создания |

# Чтение из файлов

Проще всего прочитать текстовый файл, извлекая из него всю строку целиком, для чего, как в примере 7.5, используется функция fgets (последняя буква s в названии функции означает string — «строка»).

**Пример 7.5.** Чтение файла с помощью функции fgets

```
<?php
$fh = fopen("testfile.txt", 'r') or
    die("Файл не существует, или вы не обладаете правами на его открытие");
$line = fgets($fh);
fclose($fh);
echo $line;
?>
```

Если используется файл, созданный кодом из примера 7.4, будет получена первая строка:

```
Строка 1
```

Можно также извлечь из файла сразу несколько строк или фрагменты строк, воспользовавшись функцией fread, как показано в примере 7.6.

**Пример 7.6.** Чтение файла с помощью функции fread

```
<?php
$fh = fopen("testfile.txt", 'r') or
    die("Файл не существует, или вы не обладаете правами на его открытие");
$text = fread($fh, 3);
fclose($fh);
echo $text;
?>
```

При вызове функции fread было запрошено чтение трех символов, поэтому программа отобразит следующий текст:

```
Стр
```

Функция fread обычно используется для чтения двоичных данных. Но если она используется для чтения текстовых данных объемом более одной строки, следует брать в расчет символы новой строки.

## Копирование файлов

Попробуем создать клон нашего файла testfile.txt, воспользовавшись PHP-функцией copy. Наберите текст примера 7.7 и сохраните его в файле copyfile.php, а затем вызовите программу через браузер.

**Пример 7.7.** Копирование файла

```
<?php // copyfile.php
copy('testfile.txt', 'testfile2.txt') or die("Копирование невозможно");
echo "Файл успешно скопирован в 'testfile2.txt'";
?>
```

Если еще раз проверить содержимое вашей папки, в ней окажется новый файл testfile2.txt. Кстати, если вам не нужно, чтобы программа завершала свою работу после неудачной попытки копирования, можно воспользоваться другим вариантом синтаксиса, который показан в примере 7.8.

**Пример 7.8.** Альтернативный синтаксис для копирования файла

```
<?php // copyfile2.php
if (!copy('testfile.txt', 'testfile2.txt')) echo " Копирование невозможно";
else echo "Файл успешно скопирован в 'testfile2.txt'";
?>
```

## Перемещение файла

Для перемещения файла его следует переименовать, как показано в примере 7.9.

**Пример 7.9.** Перемещение файла

```
<?php // movefile.php
if (!rename('testfile2.txt', 'testfile2.new'))
```

```
    echo "Переименование невозможно";
else echo "Файл успешно переименован в 'testfile2.new'";
?>
```

Функцию переименования можно применять и к каталогам. Чтобы избежать предупреждений при отсутствии исходных файлов, сначала для проверки факта их существования можно вызвать функцию `file_exists`.

## Удаление файла

Для удаления файла из файловой системы достаточно, как показано в примере 7.10, воспользоваться функцией `unlink`, позволяющей сделать это.

**Пример 7.10.** Удаление файла

```
<?php // deletefile.php
if (!unlink('testfile2.new')) echo "Удаление невозможно ";
else echo "Файл 'testfile2.new' удален успешно";
?>
```

При непосредственном доступе к файлам на жестком диске нужна гарантия того, что ваша файловая система не будет поставлена под угрозу. Например, при удалении файла на основе введенной пользователем информации нужно быть абсолютно уверенными в том, что этот файл может быть удален без ущерба безопасности системы и что пользователю разрешено удалять его.

В данном случае, как и при операции перемещения, если файла с таким именем не существует, будет выведено предупреждение, появления которого можно избежать, если использовать функцию `file_exists` для проверки его существования перед вызовом функции `unlink`.

## Обновление файлов

Довольно часто возникает потребность добавлять к сохраненному файлу дополнительные данные, для чего существует множество способов. Можно воспользоваться одним из режимов добавления данных (см. табл. 7.5) или же задействовать режим, поддерживающий запись, и просто открыть файл для чтения и записи и переместить указатель файла в то место, с которого необходимо вести запись.

*Указатель файла* — это позиция внутри файла, с которой будет осуществлен очередной доступ к файлу при чтении или записи. Его не следует путать с *описателем файла* (который в примере 7.4 хранился в переменной `$fh`), содержащим сведения о том файле, к которому осуществляется доступ.

Если набрать код, показанный в примере 7.11, сохранить его в файле `update.php`, а затем вызвать его из своего браузера, то можно увидеть работу указателя.

**Пример 7.11.** Обновление файла

```
<?php // update.php
$fh = fopen("testfile.txt", 'r+') or die("Сбой открытия файла");
$text = fgets($fh);
```

```
fseek($fh, 0, SEEK_END);
fwrite($fh, "$text") or die("Сбой записи в файл");
fclose($fh);
echo "Файл 'testfile.txt' успешно обновлен";
?>
```

Эта программа открывает файл testfile.txt для чтения и записи, для чего указывается режим работы '+r', в котором указатель устанавливается в самое начало файла. Затем используется функция fgets, с помощью которой из файла считывается одна строка (до встречи первого символа перевода строки). После этого вызывается функция fseek, чтобы переместить указатель файла в самый конец, куда затем добавляется строка, которая была извлечена из начала файла (и сохранена в переменной $text), после чего файл закрывается. Получившийся в итоге файл имеет следующий вид:

```
Строка 1
Строка 2
Строка 3
Строка 1
```

Первая строка была успешно скопирована, а затем добавлена в конец файла.

В данном примере функции fseek, кроме описателя файла $fh, были переданы еще два параметра, 0 и SEEK_END. Параметр SEEK_END предписывает функции переместить указатель файла в его конец, а параметр 0 показывает, на сколько позиций нужно вернуться назад из этой позиции. В примере 7.11 используется значение 0, потому что указатель должен оставаться в конце файла.

С функцией fseek можно использовать еще два режима установки указателя: SEEK_SET и SEEK_CUR. Режим SEEK_SET предписывает функции установку указателя файла на конкретную позицию, заданную предыдущим параметром. Поэтому в следующем примере указатель файла перемещается на позицию 18:

```
fseek($fh, 18, SEEK_SET);
```

Режим SEEK_CUR приводит к установке указателя файла на позицию, которая смещена от текущей позиции за заданное значение. Если в данный момент указатель файла находится на позиции 18, то следующий вызов функции переместит его на позицию 23:

```
fseek($fh, 5, SEEK_CUR);
```

Делать это без особой надобности не рекомендуется, но таким образом даже текстовые файлы (с фиксированной длиной строк) можно использовать в качестве простых неструктурированных баз данных. В этом случае ваша программа может использовать функцию fseek для перемещения в обе стороны по такому файлу для извлечения, обновления существующих и добавления новых записей. Записи также могут удаляться путем их перезаписи нулевыми символами и т. д.

## Блокирование файлов при коллективном доступе

Веб-программы довольно часто вызываются многими пользователями в одно и то же время. Когда одновременно предпринимается попытка записи в файл более чем

одним пользователем, файл может быть поврежден. А когда один пользователь ведет в него запись, а другой считывает из него данные, с файлом ничего не случится, но читающий может получить весьма странные результаты.

Чтобы обслужить сразу несколько одновременно обращающихся к файлу пользователей, нужно воспользоваться функцией блокировки файла `flock`. Эта функция ставит в очередь все другие запросы на доступ к файлу до тех пор, пока ваша программа не снимет блокировку. Когда ваши программы обращаются к файлу, который может быть одновременно доступен нескольким пользователям, с намерением произвести в него запись, к коду нужно также добавлять задание на блокировку файла, как в примере 7.12, который является обновленной версией примера 7.11.

**Пример 7.12.** Обновление файла с использованием блокировки

```
<?php
$fh = fopen("testfile.txt", 'r+') or die("Сбой открытия файла");
$text = fgets($fh);

if (flock($fh, LOCK_EX))
{
    fseek($fh, 0, SEEK_END);
fwrite($fh, "$text") or die("Сбой записи в файл");
    flock($fh, LOCK_UN);
}

fclose($fh);
echo "Файл 'testfile.txt' успешно обновлен";
?>
```

При блокировке файла для посетителей вашего веб-сайта нужно добиться наименьшего времени отклика: блокировку нужно ставить непосредственно перед внесением изменений в файл и снимать ее сразу же после их внесения. Блокировка файла на более длительный период приведет к неоправданному замедлению работы приложения. Поэтому в примере 7.12 функция `flock` вызывается непосредственно до и после вызова функции `fwrite`.

При первом вызове `flock` с помощью параметра `LOCK_EX` устанавливается эксклюзивная блокировка того файла, ссылка на который содержится в переменной `$fh`:

```
flock($fh, LOCK_EX);
```

С этого момента и далее никакой другой процесс не может осуществлять не только запись, но даже чтение файла до тех пор, пока блокировка не будет снята с помощью передачи функции параметра `LOCK_UN`:

```
flock($fh, LOCK_UN);
```

Как только блокировка будет снята, другие процессы снова получат возможность доступа к файлу. Это одна из причин, по которой необходимо заново обращаться к нужному месту в файле при каждом чтении или записи данных: со времени последнего обращения к нему другой процесс мог внести в этот файл изменения.

Кстати, вы заметили, что вызов с требованием эксклюзивной блокировки вложен в структуру инструкции `if`? Дело в том, что `flock` поддерживается не на всех систе-

мах, и поэтому перед внесением изменений есть смысл проверить успешность установки блокировки, так как известно, что некоторые системы на это не способны.

Следует также принять во внимание, что действия функции `flock` относятся к так называемой *рекомендательной* блокировке. Это означает, что блокируются только те процессы, которые вызывают эту функцию. Если есть код, который действует напрямую и изменяет файлы, не блокируя их с помощью `flock`, он всегда сможет обойти блокировку и внести хаос в ваши файлы.

Если в каком-то кодовом фрагменте заблокировать файл, а затем по рассеянности забыть его разблокировать, это может привести к ошибке, которую будет очень трудно обнаружить.

Функция flock не будет работать в сетевой файловой системе NFS и во многих других файловых системах, основанных на применении сетей. Не стоит полагаться на flock и при использовании многопоточных серверов типа ISAPI, потому что она не защитит файлы от доступа из кода PHP-сценариев, запущенных в параллельных потоках на том же физическом сервере. Кроме того, flock не поддерживается на любых системах, использующих устаревшую файловую систему FAT, например на устаревших версиях Windows.

## Чтение всего файла целиком

Для чтения целиком всего файла без использования описателей файлов можно воспользоваться очень удобной функцией `file_get_contents`. Она очень проста в применении, о чем свидетельствует код примера 7.13.

**Пример 7.13.** Использование функции `file_get_contents`

```
<?php
echo "<pre>";  // Тег, позволяющий отображать переводы строк
echo file_get_contents("testfile.txt");
echo "</pre>"; // Прекращение действия тега pre
?>
```

Но эту функцию можно использовать и с большей пользой. С ее помощью можно извлечь файл с сервера через Интернет. В примере 7.14 показан запрос кода HTML с главной страницы веб-сайта O'Reilly с последующим ее отображением, как при обычном переходе на саму веб-страницу. Полученный результат будет похож на копию страницы, приведенную на рис. 7.1.

**Пример 7.14.** Захват главной страницы веб-сайта O'Reilly

```
<?php
echo file_get_contents("http://oreilly.com");
?>
```

## Загрузка файлов на веб-сервер

Загрузка файлов на веб-сервер вызывает затруднения у многих пользователей, но сделать этот процесс еще проще, чем он есть на самом деле, не представляется возможным. Для загрузки файла из формы нужно лишь выбрать специальный тип кодировки, который называется `multipart/form-data`, а все остальное сделает ваш

веб-браузер. Чтобы увидеть этот процесс в работе, наберите программу, представленную в примере 7.15, и сохраните ее в файле под именем upload.php. Когда этот файл будет запущен, в браузере появится форма, позволяющая загружать на сервер любой выбранный файл.


**Рис. 7.1.** Главная страница веб-сайта O'Reilly, захваченная с помощью функции file_get_contents

**Пример 7.15.** Программа для загрузки изображений, хранящаяся в файле upload.php

```
<?php // upload.php
echo <<<_END
<html><head><title>PHP-форма для загрузки файлов на сервер</title></head><body>
<form method='post' action='upload.php' enctype='multipart/form-data'>
Выберите файл: <input type='file' name='filename' size='10' />
<input type='submit' value='Загрузить' />
</form>
_END;

if ($_FILES)
{
    $name = $_FILES['filename']['name'];
    move_uploaded_file($_FILES['filename']['tmp_name'], $name);
    echo "Загружаемое изображение '$name'<br /><img src='$name' />";
}

echo "</body></html>";
?>
```

Проанализируем программу по блокам. В первой строке многострочной инструкции echo задается начало HTML-документа, отображается заголовок, а затем начинается тело документа.

Далее идет форма, для передачи содержимого которой выбран метод POST, задается предназначение всех отправляемых программе upload.php (то есть самой нашей программе) данных и указывается веб-браузеру на то, что отправляемые данные должны быть закодированы с использованием типа содержимого multipart/form-data.

Для подготовки формы в следующих строках задается отображение приглашения Выберите файл, а затем дважды запрашивается пользовательский ввод. Сначала от пользователя требуется указать файл. В параметрах ввода задаются тип вводимой информации — input type, в качестве которого указан файл — file, имя — name, в качестве которого определено имя файла — filename, а также размер поля ввода — size, в качестве которого указана ширина поля, составляющая десять символов.

Затем от пользователя требуется ввести команду на отправку данных формы, для чего служит кнопка с надписью Загрузить (эта надпись заменяет текст, используемый по умолчанию, — Submit Query, что означает «Отправить запрос»). После этого форма закрывается.

В этой небольшой программе показана весьма распространенная технология веб-программирования, в которой одна и та же программа вызывается дважды: один раз при первом посещении страницы, а второй — когда пользователь щелкает на кнопке отправки формы.

PHP-код, предназначенный для приема загружаемых данных, предельно прост, поскольку все загружаемые на сервер файлы помещаются в ассоциативный системный массив $_FILES. Поэтому для установки факта отправки пользователем файла достаточно проверить, есть ли у массива $_FILES хоть какое-нибудь содержимое. Эта проверка осуществляется с помощью инструкции if ($_FILES).

При первом посещении страницы пользователем, которое происходит еще до загрузки файла, массив $_FILES пуст, поэтому программа пропускает этот блок кода. Когда пользователь загружает файл, программа запускается еще раз и обнаруживает присутствие элемента в массиве $_FILES.

Когда программа обнаружит, что файл был загружен, его имя, каким оно было прочитано из компьютера, занимавшегося загрузкой, извлекается и помещается в переменную $name. Теперь нужно только переместить файл из временного места, где PHP хранит загруженные файлы, в постоянное место хранения. Это делается с помощью функции move_uploaded_file, которой передается исходное имя файла, сохраняемого в текущем каталоге.

И наконец, загруженное на сервер изображение отображается путем помещения его имени в тег <IMG>. Возможный результат показан на рис. 7.2.

Если при запуске программы в ответ на вызов функции move_uploaded_file будет получено предупреждение об отсутствии прав доступа — Permission denied, значит, у вас нет права на доступ к папке, из которой запущена программа.

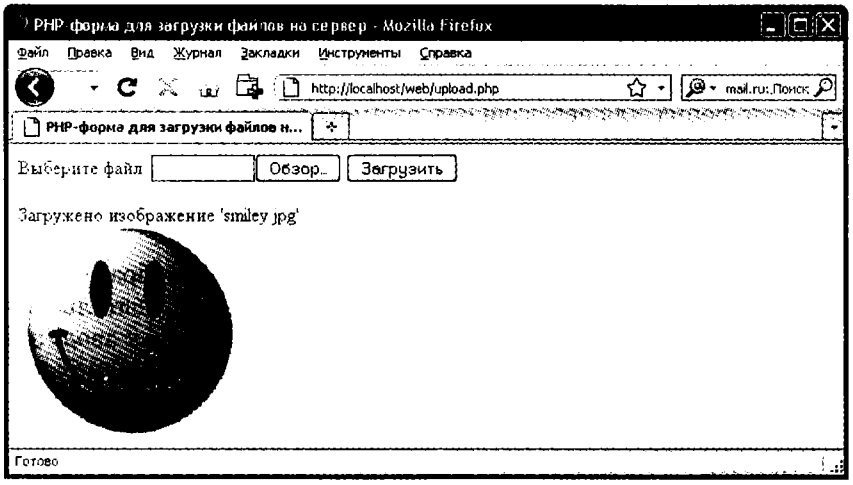



**Рис. 7.2.** Загрузка изображения с помощью формы данных

## Использование массива $_FILES

При загрузке файла на сервер в массиве `$_FILES` сохраняются пять элементов, показанных в табл. 7.6 (где используется загружаемый файл, имя которого предоставляется отправляемой серверу формой).

**Таблица 7.6.** Содержимое массива $_FILES

| Элемент массива | Содержимое |
|---|---|
| $_FILES['file']['name'] | Имя загруженного файла (например, smiley.jpg) |
| $_FILES['file']['type'] | Тип содержимого файла (например, image/jpeg) |
| $_FILES['file']['size'] | Размер файла в байтах |
| $_FILES['file']['tmp_name'] | Имя временного файла, сохраненного на сервере |
| $_FILES['file']['error'] | Код ошибки, получаемый после загрузки файла |

Типы содержимого обычно называли MIME-типами (Multipurpose Internet Mail Extension – многоцелевые почтовые расширения в Интернете). Но поскольку позже они были распространены на все виды передаваемой через Интернет информации, то теперь их часто называют типами информации, используемой в Интернете (Internet media types). В табл. 7.7 показаны некоторые из наиболее часто используемых типов, которые появляются в элементе массива `$_FILES['file']['type']`.

**Таблица 7.7.** Некоторые наиболее распространенные типы информации, используемой в Интернете

| | | | |
|---|---|---|---|
| application/pdf | image/gif | multipart/form-data | text/xml |
| application/zip | image/jpeg | text/css | video/mpeg |
| audio/mpeg | image/png | text/html | video/mp4 |
| audio/x-wav | image/tiff | text/plain | video/quicktime |

## Проверка допустимости

Надеюсь, что не нужно говорить (хотя я все равно это сделаю) о крайней важности проверки допустимости присланных формой данных, обусловленной существующей для пользователей возможностью взломать ваш сервер.

Помимо проверки вредоносности введенных данных, нужно также проверить, был ли файл получен, и если он получен, то был ли отправлен правильный тип данных.

С учетом всего этого программа upload.php была превращена в программу upload2.php, показанную в примере 7.16.

**Пример 7.16.** Более безопасная версия upload.php

```
<?php // upload2.php
echo <<<_END
<html><head><title>PHP-форма для загрузки файлов на сервер</title></head><body>
<form method='post' action='upload2.php' enctype='multipart/form-data'>
Выберите файл с расширением JPG, GIF, PNG или TIF:
<input type='file' name='filename' size='10' />
<input type='submit' value='Загрузить' /></form>
_END;

if ($_FILES)
{
    $name = $_FILES['filename']['name'];

    switch($_FILES['filename']['type'])
    {
        case 'image/jpeg': $ext = 'jpg'; break;
        case 'image/gif':  $ext = 'gif'; break;
        case 'image/png':  $ext = 'png'; break;
        case 'image/tiff': $ext = 'tif'; break;
        default:           $ext = '';    break;
    }
    if ($ext)
    {
        $n = "image.$ext";
        move_uploaded_file($_FILES['filename']['tmp_name'], $n);
        echo "Загружено изображение '$name' под именем '$n':<br />";
        echo "<img src='$n' />";
    }
    else echo "'$name' – неприемлемый файл изображения";
}
else echo "Загрузки изображения не произошло";

echo "</body></html>";
?>
```

Блок, не содержащий HTML-кода, был расширен, и теперь вместо шести строк примера 7.15 в нем содержится 20 строк, начиная с `if ($_FILES)`.

Как и в предыдущей версии, в этой строке `if` выполняется проверка факта отправки данных, но теперь у этой инструкции ближе к концу программы есть

и соответствующая ей инструкция `else`, которая выводит на экран сообщение о том, что загрузки изображения не произошло.

В теле инструкции `if` переменной `$name` присваивается значение имени файла, полученное (как и прежде) от загружающего компьютера, но на этот раз мы не полагаемся на то, что пользователь отправил нам приемлемые данные. Вместо этого используется инструкция `switch`, предназначенная для проверки соответствия типа загружаемого контекста четырем типам изображений, которые поддерживаются этой программой. При обнаружении соответствия переменной `$ext` присваивается трехсимвольное расширение имени файла, относящееся к этому типу. Если соответствие не обнаружится, значит, загруженный файл не относится к приемлемому типу и переменной `$ext` будет присвоена пустая строка `""`.

В следующем блоке кода проверяется, содержит ли переменная `$ext` строку, и в случае положительного ответа в переменной `$n` создается новое имя файла, составленное из основы *image* и расширения, сохраненного в переменной `$ext`. Это означает, что программа полностью контролирует имя создаваемого файла и этим именем может быть только одно из следующих: `image.jpg`, `image.gif`, `image.png` или `image.tif`.

Поскольку программе больше ничего не угрожает, остальной PHP-код похож на код предыдущей версии. Он перемещает загруженное временное изображение на его новое место, затем выводит его на экран, а вместе с ним отображает старое и новое имена изображения.

Об удалении временного файла, созданного PHP в процессе загрузки, беспокоиться не стоит, поскольку, если файл не был перемещен или переименован, он будет удален автоматически, как только программа завершит свою работу.

Когда по условию инструкции `if` произойдет переход к инструкции `else`, которая выполняется только в том случае, если загружен неподдерживаемый тип изображения, программа выводит сообщение об ошибке.

Я настоятельно рекомендую применить такой же подход и использовать заранее подобранные имена и места для загружаемых файлов, когда вы будете создавать собственную программу загрузки. Тогда будут исключены любые попытки добавления к используемым переменным каких-нибудь других путевых имен и других данных, способных нанести вред. Если подразумевается, что несколько пользователей могут загружать файл с одним и тем же именем, то такие файлы можно снабжать префиксами, представляющими собой имена пользователей, или сохранять их в отдельных папках, созданных для каждого пользователя.

Но если нужно использовать предоставленное имя файла, его следует обезвредить, разрешив применение только буквенно-цифровых символов и точки, что можно сделать с помощью следующей команды, использующей регулярное выражение (см. главу 17) для осуществления поиска и замены символов в значении переменной `$name`:

```
$name = ereg_replace("[^A-Za-z0-9.]", "", $name);
```

Эта команда оставляет в строковой переменной `$name` только символы A–Z, a–z, 0–9 и точку, а прочие символы удаляет.

Для обеспечения работы своей программы на всех системах, независимо от их чувствительности к регистру букв, стоит воспользоваться другой командой, которая одновременно с предыдущими действиями переводит все символы верхнего регистра в нижний:

```
$name = strtolower(ereg_replace("[^A-Za-z0-9.]", "", $name));
```

Иногда можно встретить тип содержимого image/pjpeg, который служит признаком прогрессивного JPEG-формата. Этот тип можно без лишних опасений добавить к вашему коду в качестве альтернативы для image/jpeg:

```
case 'image/pjpeg':
case 'image/jpeg': $ext = 'jpg'; break;
```

# Системные вызовы

Иногда функцию для осуществления конкретного действия можно найти не в PHP, а в операционной системе, под управлением которой запущен этот язык. В таком случае для выполнения задачи можно применить системный вызов exec.

Например, для быстрого просмотра содержимого текущего каталога можно воспользоваться программой, показанной в примере 7.17. В процессе работы в системе Windows она не потребует изменений и задействует Windows-команду dir. В Linux, Unix или Mac OS X нужно будет закомментировать или удалить первую строку и убрать символы комментария из второй строки, чтобы применить системную команду ls. При желании можете набрать текст этой программы, сохранить его как exec.php и вызвать из своего браузера.

**Пример 7.17.** Выполнение системной команды

```
<?php // exec.php
$cmd = "dir";   // Windows
// $cmd = "ls"; // Linux, Unix & Mac

exec(escapeshellcmd($cmd), $output, $status);

if ($status) echo "Команда exec не выполнена";
else
{
    echo "<pre>";
    foreach($output as $line) echo "$line\n";
}
?>
```

В зависимости от используемой системы в результате запуска этой программы будет выведена следующая информация (полученная при использовании Windows-команды dir):

```
Volume in drive C is HP
Volume Serial Number is E67F-EE11
Directory of C:\web
```

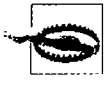
```
20/01/2011  10:34
.
    20/01/2011  10:34
..
        19/01/2011  16:26               236 maketest.php
        20/01/2011  10:47               198 exec.php
        20/01/2011  08:04            13.741 smiley.jpg
        19/01/2011  18:01                54 test.php
        19/01/2011  16:59                35 testfile.txt
        20/01/2011  09:35               886 upload.php
                       6 File(s)     15.150 bytes
                       2 Dir(s)  382.907.748.352 bytes free
```

Функция exec воспринимает три аргумента.

1. Саму команду (в предыдущем случае это `$cmd`).
2. Массив, в который система поместит информацию, получаемую в результате выполнения команды (в предыдущем случае это `$output`).
3. Переменную для хранения возвращаемого статуса вызова (в предыдущем случае это `$status`).

При желании параметры `$output` и `$status` можно опустить, но тогда ничего не будет известно ни о выходной информации, созданной в результате вызова, ни даже о том, насколько успешен был сам вызов.

Обратите внимание также на применение функции `escapeshellcmd`. Желательно выработать привычку постоянно использовать эту функцию при вызове функции exec, поскольку она обезвреживает содержимое командной строки, предотвращая выполнение случайных команд в том случае, если пользователю предоставляется возможность их ввода.



Как правило, функции системных вызовов на веб-хостах общего пользования запрещены как представляющие угрозу системе безопасности. По возможности все задачи нужно стараться решить средствами PHP и обращаться к системе напрямую только в случае крайней необходимости. Кроме того, вы должны знать, что обращение к системе выполняется довольно медленно, и если приложение рассчитано на запуск как в Windows, так и в Linux/Unix, для него следует создавать две реализации вызова.

# XHTML

Некоторые элементы XHTML (eXtensible Hypertext Markup Language — расширяемый язык гипертекстовой разметки) уже встречались в этой книге, хотя вы могли этого и не заметить. Например, вместо простого HTML-тега `<br>` я использовал его XHTML-версию `<br />`. В чем же разница между этими двумя языками разметки?

На первый взгляд, они не слишком отличаются друг от друга, но XHTML облагораживает HTML, проясняя множество мелких непоследовательных моментов, усложняющих обработку. Для HTML требуется очень сложный и весьма лояльный синтаксический анализатор (парсер), а для XHTML, в котором используется стан-

дартный синтаксис, больше напоминающий XML (eXtensible Markup Language — расширяемый язык разметки), нужна совсем несложная обработка с использованием очень простого парсера, то есть той части кода, которая обрабатывает теги и команды и определяет их предназначение.

## Преимущества XHTML

Документы XHTML могут довольно быстро обрабатываться любой программой, способной работать с XML-файлами. При неизменном росте количества устройств, приобретающих способность обработки веб-контента (примером которых могут послужить iPhone и BlackBerry, а также телефоны, работающие под управлением Android и Windows, не говоря уже о множестве новых планшетных устройств), обеспечение пристойного внешнего вида контента на этих устройствах, как, впрочем, и в веб-браузерах стационарных компьютеров и ноутбуков, приобретает все большее значение. Более компактный синтаксис, требующийся для XHTML, является весьма весомым фактором, содействующим такой кросс-платформенной совместимости.

Уже настали времена, когда разработчики браузеров, чтобы получить возможность поставлять все более быстрые и мощные программы, стараются подтолкнуть веб-разработчиков к использованию XHTML. И в итоге случится такое, когда на смену HTML придет XHTML, поэтому его использование лучше не откладывать в долгий ящик.

## Версии XHTML

Стандарт XHTML постоянно развивается и использовались сразу несколько версий этого языка, но по тем или иным причинам единственной версией, с которой вам нужно разобраться, осталась XHTML 1.0.

Хотя были и другие версии XHTML (например, 1.1, 1.2 и 2.0), которые дошли до стадии рекомендованных и даже начали использоваться, ни одна из них не нашла достаточной поддержки у веб-разработчиков, что еще больше упрощает нашу с вами задачу, сводя ее к освоению только одной версии.

## Отличие XHTML от HTML

Отличия XHTML от HTML описываются следующими правилами.

- Каждый тег должны быть закрыт другим тегом. В случае отсутствия соответствующего закрывающего тега тег должен закрывать сам себя, используя пробел, за которым следуют символы / и >. Так, к примеру, тег `<input type='submit'>` необходимо изменить, чтобы получился тег `<input type='submit' />`. Кроме того, все открывающие теги `<p>` теперь требуют закрывающих тегов `</p>`, которые нельзя заменить тегами `<p />`.
- Все теги должны быть правильно вложены. Поэтому строка `<b>Меня зовут <i>Робин </b></i>` недопустима, поскольку открытый тег `<b>` закрывается до закрытия тега `<i>`. Правильная версия строки — `<b>Меня зовут <i>Робин</i></b>`.

- Все атрибуты тегов должны быть взяты в кавычки. Вместо использования таких тегов, как <form method=post action=post.php>, нужно применять теги <form method='post' action='post.php'>. Можно также задействовать двойные кавычки: <form method="post" action="post.php">.
- Символ амперсанда (&) не может использоваться сам по себе. Например, строка Batman & Robin должна быть заменена строкой Batman & Robin. Это означает, что URL-адреса также требуют модификации: HTML-синтаксис <a href="index.php?page=12&item=15"> необходимо заменить на <a href="index.php?page=12&item=15">.
- Теги XHTML чувствительны к регистру, и для них нужно использовать символы нижнего регистра. Поэтому такой HTML-код, как <BODY><DIV ID="heading">, должен быть заменен следующим синтаксисом: <body> .
- Теперь атрибуты не могут быть минимизированы, поэтому такие теги, как <option name="bill" selected>, должны быть заменены присваиваемым значением: <option name="bill" selected="selected">. Все остальные атрибуты, например checked и disabled, также должны быть заменены на checked="checked", disabled="disabled" и т. д.
- Документы XHTML должны начинаться с помещаемого в самую первую строку XML-объявления. Эти объявления должны быть похожи на следующее: <?xml version="1.0" encoding="UTF-8"?>.
- Изменилось объявление DOCTYPE.
- Теперь тег <html> требует использования атрибута xmlns.

Рассмотрим документ, приведенный в примере 7.18, который оформлен в стандарте XHTML 1.0.

**Пример 7.18.** Документ XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <meta http-equiv="Content-Type"
            content="text/html; charset=utf-8" />
        <title>Документ XHTML 1.0 </title>
    </head>
    <body>
        <p>Это пример документа XHTML 1.0 </p>
        <h1>Это заголовок</h1>
        <p>Это просто текст</p>
    </body>
</html>
```

Как уже упоминалось, документ начинается с XML-объявления, за которым следуют объявление DOCTYPE и тег <html> с атрибутом xmlns. Далее все выглядит как обычный код HTML, за исключением того, что тег meta закрыт, как и полагается, символами />.

## Типы документов HTML 4.01

Чтобы дать браузеру точную информацию о том, как необходимо обрабатывать документ, используется объявление DOCTYPE, которое определяет допустимый синтаксис. Как следует из приводимых далее примеров, HTML 4.01 поддерживает три определения типа документа (Document Type Declarations (DTD)). Строгий DTD в примере 7.19 требует полного соблюдения синтаксиса HTML 4.01.

**Пример 7.19.** Строгий DTD в HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

Нестрогий DTD в примере 7.20 разрешает использование некоторых устаревших элементов и не рекомендуемых атрибутов. (В стандартах, размещенных по адресу http://w3.org/TR/xhtml1, объясняется, какие элементы не рекомендуется использовать.)

**Пример 7.20.** Промежуточный DTD в HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

И наконец, в примере 7.21 обозначен документ HTML 4.01, содержащий фреймы.

**Пример 7.21.** Фреймовый DTD в HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
```

## Типы документов HTML5

Использование типов документов в HTML5 существенно упростилось, поскольку теперь остался только один из них:

```
<!DOCTYPE html>
```

Одного простого слова html достаточно, чтобы сказать браузеру о том, что ваша веб-страница разработана для HTML5. Более того, поскольку все самые последние версии наиболее популярных браузеров начиная примерно с 2011 года поддерживают большинство спецификаций HTML5, растет вероятность того, что этот тип документа будет единственным, который вам понадобится, если только вы не выберете в качестве своей задачи обслуживание устаревших браузеров.

## Типы документов XHTML 1.0

Возможно, прежде вам уже встречались HTML-документы одного или нескольких типов. Но с появлением XHTML 1.0 синтаксис претерпел небольшие изменения, показанные в следующих примерах.

Строгий DTD в примере 7.22 исключает применение нерекомендуемых атрибутов и требует использования только правильного кода.

**Пример 7.22.** Строгий DTD в XHTML 1.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Промежуточный DTD в XHTML 1.0, показанный в примере 7.23, позволяет использовать нерекомендуемые атрибуты, и это самый востребованный DTD.

**Пример 7.23.** Промежуточный DTD в XHTML 1.0

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

В примере 7.24 показан единственный XHTML 1.0 DTD, поддерживающий фреймы.

**Пример 7.24.** Фреймовый DTD в XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

## Проверка правильности XHTML

Чтобы проверить код XHTML, следует посетить сайт проверки правильности W3C по адресу http://validator.w3.org, с помощью которого можно проверить документ по URL-адресу путем загрузки формы или путем его набора или копирования и вставки в веб-форму. Перед разработкой кода PHP, предназначенного для создания веб-страницы, передайте выборку выходной информации, которую хотите создавать, на сайт проверки правильности. Независимо от того, насколько тщательно вы создавали свой XHTML-код, вы будете удивлены обилию пропущенных ошибок.

Если документ не полностью совместим с XHTML, вы получите сообщение, объясняющее, как это можно исправить. На рис. 7.3 показано, что документ из примера 7.18 успешно прошел проверку на соответствие строгому XHTML 1.0.

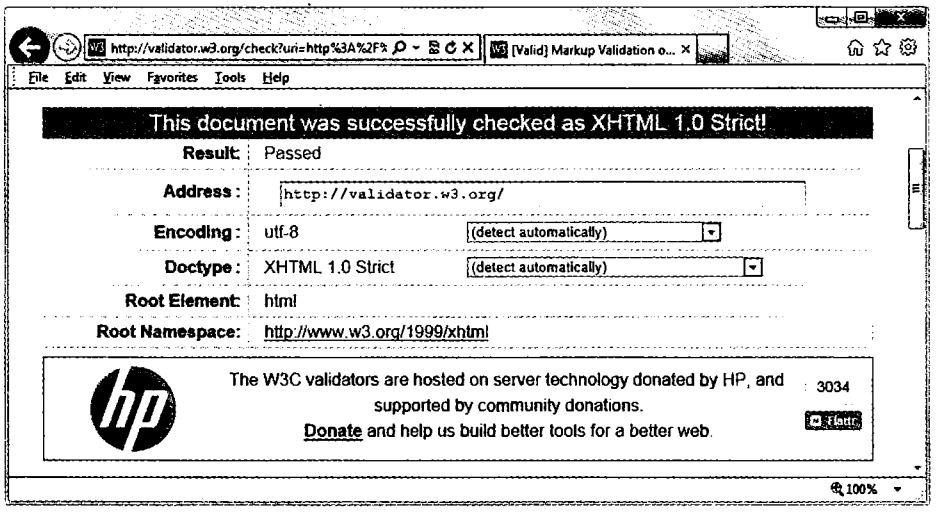



**Рис. 7.3.** Документ из примера 7.18, прошедший проверку правильности

Оказывается, документы XHTML 1.0 настолько близки к HTML, что даже при вызове в браузере, который не знаком с XHTML, они будут отображаться правильно. Но есть одна потенциальная проблема, связанная с тегом <script>. Для обеспечения совместимости следует избегать использования синтаксиса <script src="script.src" />, заменяя его синтаксисом <script src="script.src"></script>.

В этой главе мы проделали еще одно длинное путешествие на пути овладения языком PHP. Теперь у вас появилось представление о форматировании текста, работе с файлами, XHTML и т. д.

# Проверьте ваши знания

1. Какой спецификатор преобразования следует использовать в функции `printf` для отображения числа с плавающей точкой?
2. Какая инструкция `printf` может быть использована для приема строки `Happy Birthday` и вывода строки `**Happy`?
3. Какой альтернативной функцией следует воспользоваться для выдачи информации из `printf` не в браузер, а в переменную?
4. Как создать отметку времени Unix для времени и даты, представленных в виде «7:11am May 2nd, 2016»?
5. Какой режим доступа к файлу следует использовать в функции `fopen`, чтобы открыть файл в режиме чтения и записи с усечением его размера и установкой указателя на начало файла?
6. Какую PHP-команду следует применить для удаления файла `file.txt`?
7. Какая PHP-функция используется для чтения целиком всего файла и даже для извлечения его из Всемирной паутины?
8. В какой системной переменной PHP содержатся сведения о загруженных на сервер файлах?
9. Какая PHP-функция позволяет запускать системные команды?
10. Что не соответствует правилам в следующем теге XHTML 1.0: `<input type=file name=file size=10>`?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 7».

# 8 Введение в MySQL

Более чем 10 млн установленных на компьютерах копий MySQL позволяют, наверное, считать ее наиболее популярной системой управления базами данных для веб-серверов. Она была разработана в середине 1990-х годов и теперь превратилась в полноценную технологию, входящую в состав многих современных, наиболее посещаемых интернет-ресурсов.

Одна из причин такого успеха — это то, что она, как и PHP, является продуктом свободного пользования. Кроме того, это очень мощная и исключительно быстрая система, способная работать даже на самом скромном оборудовании, не отнимая слишком много системных ресурсов.

Система MySQL обладает также хорошей масштабируемостью, то есть база данных может увеличиваться в объеме вместе с вашим веб-сайтом. Сравнительный анализ нескольких баз данных, проведенный журналом eWEEK, показал, что MySQL и Oracle получили равное количество голосов за лучшую производительность и самый высокий показатель масштабируемости.

## Основные характеристики MySQL

База данных — это структурированная коллекция записей или данных, хранящихся в компьютерной системе и организованных так, что можно осуществлять быстрый поиск и извлечение нужной информации.

В названии MySQL составляющая SQL означает Structured Query Language — «язык структурированных запросов». Если характеризовать его в общих чертах, то это язык, основанный на словах английского языка и используемый также в других системах управления базами данных, например Oracle и Microsoft SQL Server. Он разработан для предоставления возможности создания простых запросов к базе данных посредством команд следующего вида:

```
SELECT title FROM publications WHERE author = 'Charles Dickens';
```

В базе данных MySQL имеются одна или несколько *таблиц*, каждая из которых состоит из *записей*, или *строк*. Внутри строк находятся разные *столбцы*, или *поля*, в которых и содержатся данные. В табл. 8.1 показана учебная база данных, в которой присутствует информация о пяти книгах, структурированная по авторам, названиям, категориям и годам издания.

**Таблица 8.1.** Пример простой базы данных

| Author (автор) | Title (название) | Type (категория) | Year (год) |
|---|---|---|---|
| Mark Twain (Марк Твен) | The Adventures of Tom Sawyer (Приключения Тома Сойера) | Fiction (Художественная) | 1876 |
| Jane Austen (Джейн Остин) | Pride and Prejudice (Гордость и предубеждение) | Fiction (Художественная) | 1811 |
| Charles Darwin (Чарльз Дарвин) | The Origin of Species (Происхождение видов) | Non-Fiction (Научная) | 1856 |
| Charles Dickens (Чарльз Диккенс) | The Old Curiosity Shop (Лавка древностей) | Fiction (Художественная) | 1841 |
| William Shakespeare (Вильям Шекспир) | Romeo and Juliet (Ромео и Джульетта) | Play (Пьеса) | 1594 |

Каждая строка таблицы подобна строке в таблице MySQL, и каждый элемент в этой строке подобен полю MySQL.

Чтобы однозначно идентифицировать эту базу данных, в последующих примерах я буду ссылаться на нее как на базу данных *publications* (издания). Как вы уже заметили, все эти издания относятся к классической литературе, поэтому таблицу в базе данных, содержащую сведения о них, я буду называть *classics*.

# Сводка понятий, используемых в базах данных

Основными понятиями, с которыми следует ознакомиться на данном этапе, являются:

- *база данных* — контейнер для всей коллекции данных MySQL;
- *таблица* — вложенный в базу данных контейнер, в котором хранятся сами данные;
- *строка* — отдельная запись, в которой могут содержаться несколько полей;
- *столбец* — имя поля внутри строки.

Следует заметить, что я не пытаюсь воспроизвести точную терминологию, используемую в учебной литературе по реляционным базам данных, а хочу лишь дать простые, обычные определения, помогающие быстро усвоить основные понятия и приступить к работе с базой данных.

# Доступ к MySQL из командной строки

Работать с MySQL можно тремя основными способами: используя командную строку, применяя веб-интерфейс наподобие phpMyAdmin и задействуя такой язык программирования, как PHP. Третий из перечисленных способов будет рассмотрен в главе 10, а сейчас изучим первые два способа.

# Начало работы с интерфейсом командной строки

В следующих разделах даны соответствующие инструкции для операционных систем Windows, Mac OS X и Linux.

## Для пользователей Windows

Если у вас в соответствии с инструкциями, изложенными в главе 2, установлена программа Zend Server CE WAMP, то доступ к исполняемой программе MySQL можно получить из следующих каталогов (первый из которых относится к 32-разрядным, а второй — к 64-разрядным компьютерам):

```
C:\Program Files\Zend\MySQL51\bin
C:\Program Files (x86)\Zend\MySQL51\bin
```

Если Zend Server CE установлена не в каталоге \Program Files (или \Program Files (x86)), то нужно воспользоваться тем каталогом, в котором она установлена.

По умолчанию начальным для MySQL будет пользователь по имени *root*, у которого не установлен пароль. Учитывая то, что это разработочный сервер, доступ к которому можете получить только вы, мы не станем устанавливать пароль.

Чтобы войти в интерфейс командной строки MySQL, следует выбрать команду **Пуск ▸ Выполнить** и в окне запуска ввести команду `CMD`, после чего нажать клавишу **Enter**. В результате будет вызвано командное окно Windows. Находясь в этом окне, следует ввести следующую команду (внося в нее соответствующие коррективы):

```
"C:\Program Files\Zend\MySQL51\bin\mysql" -u root
"C:\Program Files (x86)\Zend\MySQL51\bin\mysql" -u root
```

Обратите внимание на кавычки, в которые заключены основной путь и имя файла. Их присутствие обусловлено наличием пробелов, которые неправильно интерпретируются командной строкой, а кавычки группируют составные части имени файла в единую строку, понятную программе Command.

Эта команда предписывает MySQL зарегистрировать вас как пользователя *root* без пароля. Теперь вы должны оказаться в среде MySQL и сможете приступить к вводу команд. Чтобы убедиться в том, что все работает должным образом, введите следующую команду, результат выполнения которой должен быть похож на показанный на рис. 8.1:

```
SHOW databases;
```

Если это не приведет к желаемому результату и будет выдано сообщение об ошибке, убедитесь в том, что у вас система MySQL правильно установлена наряду с Zend Server CE (как описано в главе 2). Если же все прошло успешно, переходите к подразделу «Использование интерфейса командной строки».


**Рис. 8.1.** Доступ к MySQL из командной строки Windows

## Для пользователей Mac OS X

Чтобы иметь возможность выполнять то, о чем говорится в этой главе, нужно в соответствии с инструкциями, изложенными в главе 2, установить программу Zend Server CE. Следует также иметь работающий веб-сервер с запущенным сервером MySQL.

Для входа в интерфейс командной строки MySQL необходимо запустить программу `Terminal` (доступную в меню Utilities программы Finder). Затем вызвать программу MySQL, которая должна быть установлена в каталоге /usr/local/zend/mysql/bin.

По умолчанию исходным пользователем для MySQL будет пользователь по имени *root*, и пароль у него также будет *root*. Поэтому для запуска программы наберите следующую команду:

```
/usr/local/zend/mysql/bin/mysql -u root
```

Эта команда предпишет MySQL зарегистрировать вас как пользователя *root* и запросить пароль. Чтобы проверить, что все в порядке, наберите следующую команду (результаты ее выполнения показаны на рис. 8.2):

```
SHOW databases;
```

Если будет получено сообщение об ошибке, не позволяющей подключиться к локальному серверу MySQL (Can't connect to local MySQL server through socket), значит, у вас не запущен сервер MySQL. Убедитесь в том, что вы последовали совету в главе 2, касавшемуся настройки MySQL на запуск вместе с запуском OS X.

Теперь вы готовы перейти к подразделу «Использование интерфейса командной строки».

## Для пользователей Linux

На машинах, работающих под управлением Unix-подобных операционных систем, к которым относится и Linux, PHP и MySQL практически всегда будут

заранее установлены и запущены, позволяя вводить текст примеров следующего раздела. Но сначала нужно войти в систему MySQL, введя следующую команду:

```
mysql -u root -p
```

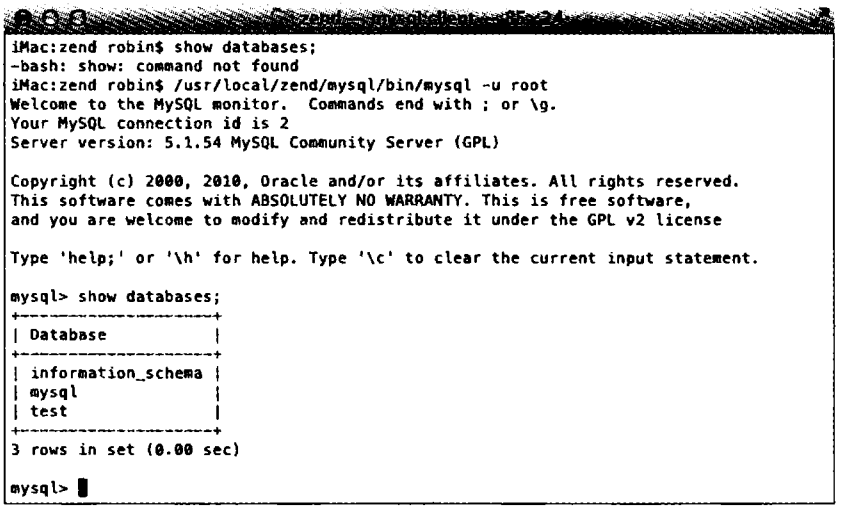



**Рис. 8.2.** Доступ к MySQL из программы Terminal на OS X

Эта команда предписывает MySQL зарегистрировать вас под именем *root* и запросить пароль. Если у вас есть пароль, то нужно его ввести, если пароль отсутствует — просто нажать клавишу Enter.

После входа в систему нужно проверить работоспособность программы, набрав следующую команду, примерный результат выполнения которой показан на рис. 8.3:

```
SHOW databases;
```

Если по каким-то причинам эта команда не сработает, обратитесь к разделу главы 2 «Установка LAMP на Linux», чтобы обеспечить правильную установку MySQL.

Если команда будет выполнена, можете перейти к подразделу «Использование интерфейса командной строки».

## MySQL на удаленном сервере

При получении доступа к MySQL на удаленном сервере нужно выйти на удаленную машину с помощью Telnet (с точки зрения безопасности предпочтительнее для этого воспользоваться SSH), которая, скорее всего, будет работать под управлением операционной системы семейства типа Linux/FreeBSD/Unix. После подключения к удаленной машине можно встретиться с незначительными вариациями в порядке работы, зависящими от настроек сервера, которые выполнены системным администратором, особенно если этот сервер предназначен для коллективного пользования.

Поэтому следует убедиться в доступности MySQL и в том, что у вас есть имя пользователя и пароль. Получив эти сведения, можно набрать следующую команду, в которой вместо username следует вставить предоставленное имя пользователя:

```
mysql -u username -p
```

**Рис. 8.3.** Доступ к MySQL в Linux

После появления приглашения следует ввести пароль. Затем можно попробовать ввести следующую команду, примерный результат выполнения которой показан на рис. 8.3:

```
SHOW databases;
```

В перечне баз данных могут присутствовать и другие ранее созданные базы, среди которых базы test может и не оказаться.

Следует также понимать, что все находится под полным контролем системного администратора, и вы можете столкнуться с некоторыми неожиданными настройками. Например, может оказаться, что вам следует ставить перед именами всех создаваемых вами баз данных уникальную идентификационную строку, обеспечивающую их бесконфликтную работу с базами данных, созданными другими пользователями.

При возникновении любых проблем нужно переговорить с системным администратором, который должен с ними разобраться. У него нужно запросить имя пользователя и пароль и попросить дать вам возможность создавать новые базы данных или как минимум попросить создать для вас хотя бы одну готовую к работе базу данных. Тогда в этой базе можно будет создать все необходимые таблицы.

## Использование интерфейса командной строки

Для всего, что изложено далее в тексте главы, нет никакой разницы, из какой именно системы — Windows, Mac OS X или Linux — вы получаете непосредственный

доступ к MySQL, поскольку все используемые команды (и сообщения об ошибках, которые могут быть получены) абсолютно одинаковы.

## Точка с запятой

Начнем с самого простого. Набирая команду, вы, наверное, заметили точку с запятой (;) в конце `SHOW databases;`? Этот символ используется в MySQL для завершения команд или отделения их друг от друга. Если забыть поставить этот символ, MySQL выдаст приглашение и будет ожидать от вас его ввода. Запрашиваемая точка с запятой стала частью синтаксиса, позволяющего вводить длинные команды, разбивая их на несколько строк. Она также позволяет вводить сразу несколько команд, после каждой из которых стоит точка с запятой. После нажатия клавиши Enter интерпретатор получит все эти команды в едином пакете и выполнит их в порядке следования.

Вместо результата введенной команды довольно часто появляется приглашение MySQL. Это означает, что вы забыли поставить завершающую точку с запятой. В таком случае нужно просто ввести точку с запятой, нажать клавишу Enter, и вы получите желаемый результат.

На экране могут появляться шесть разных приглашений MySQL (табл. 8.2), позволяющих определить, на каком именно этапе многострочного ввода вы находитесь.

**Таблица 8.2.** Шесть приглашений к вводу команды MySQL

| Приглашение MySQL | Значение |
|---|---|
| mysql> | MySQL готова к работе и ждет ввода команды |
| -> | Ожидание следующей строки команды |
| '> | Ожидание следующей строки строкового значения, которое начиналось с одинарной кавычки |
| "> | Ожидание следующей строки строкового значения, которое начиналось с двойной кавычки |
| `> | Ожидание следующей строки строкового значения, которое начиналось с символа засечки (`) |
| /*> | Ожидание следующей строки комментария, который начинался с символов /* |

## Отмена команды

Если, набрав часть команды, вы решили, что ее вообще не следует выполнять, то ни в коем случае не нужно пользоваться сочетанием Ctr+C! Оно закроет программу. Вместо него можно ввести символы \c и нажать клавишу Enter. Порядок использования этой пары символов показан в примере 8.1.

**Пример 8.1.** Отмена ввода строки

```
бессмысленная для mysql строка \c
```

При наборе этой строки MySQL проигнорирует все ранее введенные символы и выдаст новое приглашение. Без \c программа выведет сообщение об ошибке. Но этой парой символов нужно пользоваться с оглядкой: если у вас уже есть откры-

тая строка или комментарий, то прежде, чем применить \c, вам придется их закрыть, иначе MySQL примет \c за часть строки. В примере 8.2 показано, как в таком случае следует задействовать \c.

Пример 8.2. Отмена ввода из строки

```
это "бессмысленная для mysql строка" \c
```

Следует также заметить, что комбинация \c после точки с запятой работать не будет, поскольку это уже будет новая инструкция.

## Команды MySQL

Нам уже приходилось встречаться с командой SHOW, которая выводит список таблиц, баз данных и многих других элементов. В табл. 8.3 приведен перечень наиболее востребованных команд.

**Таблица 8.3.** Подборка наиболее востребованных команд MySQL

| Команда | Параметр(ы) | Назначение |
|---|---|---|
| ALTER | База данных, таблица | Внесение изменений в базу данных или таблицу |
| BACKUP | Таблица | Создание резервной копии таблицы |
| \c | | Отмена ввода |
| CREATE | База данных, таблица | Создание базы данных или таблицы |
| DELETE | Выражение с участием таблицы и строки | Удаление строки из таблицы |
| DESCRIBE | Таблица | Описание столбцов таблиц |
| DROP | База данных, таблица | Удаление базы данных или таблицы |
| EXIT (Ctrl+C) | | Выход |
| GRANT | *Пользователь* подробности | Изменение привилегий *пользователя* |
| HELP (\h, \?) | *Элемент* | Отображение подсказки по *элементу* |
| INSERT | Выражение с *данными* | Вставка *данных* |
| LOCK | Таблица (таблицы) | Блокировка таблицы (таблиц) |
| QUIT (\q) | | То же самое, что и EXIT |
| RENAME | Таблица | Переименование таблицы |
| SHOW | Множество *элементов* для вывода в виде списка | Список сведений об *элементах* |
| SOURCE | *Имя_файла* | Выполнение команд из файла *имя_файла* |
| STATUS (\s) | | Отображение текущего состояния |
| TRUNCATE | Таблица | Опустошение таблицы |
| UNLOCK | Таблица (таблицы) | Снятие блокировки таблицы (таблиц) |
| UPDATE | Выражение с *данными* | Обновление существующей записи |
| USE | *База_данных* | Использование *базы_данных* |

Многие из этих команд будут рассмотрены по мере изучения этой главы, но сначала следует запомнить два важных положения, касающихся команд MySQL.

- Команды и ключевые слова SQL нечувствительны к регистру. Все три команды — `CREATE`, `create` и `CrEaTe` — абсолютно идентичны по смыслу. Но, чтобы было понятнее, для команд рекомендуется использовать буквы верхнего регистра.
- Имена таблиц не чувствительны к регистру в Windows, но чувствительны к регистру в Linux и Mac OS X. Поэтому из соображений переносимости нужно всегда выбирать буквы одного из регистров и пользоваться только ими. Для имен таблиц рекомендуется использовать буквы нижнего регистра или комбинацию из букв верхнего и нижнего регистра.

## Создание базы данных

Если вы работаете на удаленном сервере, у вас только одна учетная запись пользователя и вы имеете допуск только к одной созданной для вас базе данных, то можете перейти к изучению пункта «Создание таблицы» далее. А если это не так, то продолжим, выдав следующую команду для создания новой базы данных по имени *publications*:

```
CREATE DATABASE publications;
```

При успешном выполнении команды будет выведено сообщение, пока не имеющее для нас особого смысла, — `Query OK, 1 row affected (0.38 sec)` (Запрос выполнен, обработана 1 строка за 0,38 с), но вскоре все станет на свои места. После создания базы данных с ней нужно будет работать, поэтому даем следующую команду:

```
USE publications;
```

Теперь должно быть выведено сообщение об изменении текущей базы данных (`Database changed`), и после этого база будет готова к продолжению работы со следующими примерами.

## Организация доступа пользователей

Теперь, когда вы уже убедились в том, насколько просто пользоваться MySQL, и создали свою первую базу данных, настало время посмотреть на то, как происходит организация доступа пользователей, поскольку, вполне вероятно, вам не захочется предоставлять PHP-сценариям привилегированный доступ (root) к MySQL, что грозит большими неприятностями в случае, если кому-то вздумается взломать ваш веб-сайт.

Для создания нового пользователя выдается команда предоставления прав — `GRANT`, которая принимает следующую форму (не вздумайте все это набирать, поскольку это еще не команда):

```
GRANT ПРАВА ON база_данных.объект TO 'имя_пользователя@имя_хоста'
    IDENTIFIED BY 'пароль';
```

Эта форма не должны вызвать каких-либо затруднений, быть может, за исключением фрагмента `база_данных.объект`. Это ссылка на саму базу данных и на содержащиеся в ней объекты, например на таблицы (табл. 8.4).

**Таблица 8.4.** Примерные параметры для команды GRANT

| Параметр | Значение |
|---|---|
| *.* | Все базы данных и все их объекты |
| *база_данных.** | Только база данных с именем *база_данных* и все ее объекты |
| *база_данных.объект* | Только база данных с именем *база_данных* и ее объект с именем *объект* |

Итак, создадим пользователя, который получит доступ только к новой базе данных *publications* и ко всем ее объектам, и введем для этого следующую команду (заменив в ней имя пользователя `jim` и пароль `mypasswd` выбранными вами именем и паролем):

```
GRANT ALL ON publications.* TO 'jim'@'localhost'
 IDENTIFIED BY 'mypasswd';
```

Эта команда предоставляет пользователю `jim@localhost` полный доступ к базе данных `publications` при использовании пароля `mypasswd`. Работоспособность этой установки можно проверить, если ввести команду `QUIT` для выхода из системы, а затем перезапустить MySQL, воспользовавшись прежним способом запуска, но вместо `-u root -p` набрав `-u jim -p` или использовав в этой строке созданное вами имя пользователя. В табл. 8.5 показаны команды, соответствующие используемой вами операционной системе, при этом предполагается, что вы установили Zend Server CE (как указано в главе 2), но если на вашей системе MySQL-клиент установлен в другой каталог, то в команду следует внести соответствующие коррективы.

**Таблица 8.5.** Запуск MySQL и вход в систему под именем jim@localhost

| Операционная система | Пример команды |
|---|---|
| 32-разрядная версия Windows | "C:\Program Files\Zend\MySQL51\bin\mysql" -u jim -p |
| 64-разрядная версия Windows | "C:\Program Files (x86)<br>\Zend\MySQL51\bin\mysql" -u jim -p |
| Mac OS X | /usr/local/zend/mysql/bin/mysql -u jim -p |
| Linux | mysql -u jim –p |

Теперь, как только появится приглашение, нужно лишь ввести свой пароль, и вход в систему будет открыт. Кстати, при желании можете поместить пароль сразу же после ключа `-p` (не используя никаких пробелов). Тем самым вы избежите его ввода после появления приглашения. Но такой подход не приветствуется, поскольку, если в вашей системе зарегистрировались и другие пользователи, они могут подсмотреть вводимую вами команду и получить доступ к вашему паролю.

Вы можете предоставлять только те права, которыми уже обладаете, и должны иметь право на ввод команд GRANT. Этим и ограничивается выбор предоставляемых вам прав, если только вам не предоставлены абсолютно все права. Если вас интересуют подробности, то, пожалуйста, зайдите на сайт, где, кроме всего прочего, описана и команда REVOKE, позволяющая отозвать уже предоставленные права: http://tinyurl.com/mysqlgrant.

Вам также следует знать, что, если при создании нового пользователя не будет указана инструкция IDENTIFIED BY, у пользователя не будет пароля, из-за чего возникнет неблагоприятная с точки зрения безопасности ситуация, которой следует избегать.

## Создание таблицы

В данный момент вы должны находиться в системе MySQL, обладать всеми (ALL) правами, выделенными для базы данных *publications* (или той базы данных, которая была для вас создана), и быть готовыми к созданию своей первой таблицы. Поэтому нужно включить базу данных в работу, набрав следующую команду (и заменив *publications* именем своей базы данных, если оно у нее другое):

```
USE publications;
```

Теперь наберите построчно команды, которые приведены в примере 8.3.

**Пример 8.3.** Создание таблицы по имени classics

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 type VARCHAR(16),
 year CHAR(4)) ENGINE MyISAM;
```

Эту же команду можно ввести одной строкой:

```
CREATE TABLE classics (author VARCHAR(128), title
 VARCHAR(128), type VARCHAR(16), year CHAR(4)) ENGINE MyISAM;
```

но команды MySQL могут быть длинными и сложными, поэтому я рекомендую набирать их построчно до тех пор, пока вы не привыкнете к набору длинных строк.

После ввода команды MySQL должна выдать ответ: `Query OK, 0 rows affected`, а также показать время, затраченное на выполнение команды. Если вместо этого появится сообщение об ошибке, внимательно проверьте синтаксис команды. Должны быть на месте все скобки и запятые, а может быть, допущена какая-нибудь опечатка. Команда `ENGINE MyISAM`, которая своим присутствием в примере могла вызвать у вас недоумение, указывает MySQL тип механизма управления базой данных, применяемого к этой таблице.

Чтобы проверить факт создания новой таблицы, наберите команду:

```
DESCRIBE classics;
```

Если все в порядке, то вы увидите последовательность команд и ответов, показанных в примере 8.4, в которой особое внимание следует обратить на отображение формата таблицы.

**Пример 8.4.** Сеанс работы с MySQL: создание и проверка формата новой таблицы

```
mysql> USE publications;
Database changed
mysql> CREATE TABLE classics (
    ->  author VARCHAR(128),
    ->  title VARCHAR(128),
    ->  type VARCHAR(16),
    ->  year CHAR(4)) ENGINE MyISAM;
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE classics;
```

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| author | varchar(128) | YES  |     | NULL    |       |
| title  | varchar(128) | YES  |     | NULL    |       |
| type   | varchar(16)  | YES  |     | NULL    |       |
| year   | char(4)      | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Команда DESCRIBE является неоценимым средством отладки, когда нужно убедиться в успешном создании таблицы MySQL. Этой командой можно воспользоваться также для того, чтобы просмотреть имена полей или столбцов таблицы и типы данных в каждом из них. Рассмотрим подробнее все заголовки:

- Field — имя каждого из полей или столбцов таблицы;
- Type — тип данных, сохраняемых в поле;
- Null — показывает, может ли поле содержать значение NULL;
- Key — MySQL поддерживает *ключи*, или *индексы*, позволяющие ускорить просмотр и поиск данных. Под заголовком Key показан тип применяемого ключа (если таковой имеется);
- Default — исходное значение, присваиваемое полю, если при создании новой строки не указано никакого значения;
- Extra — дополнительная информация, например о настройке поля на автоматическое приращение его значения.

## Типы данных

В примере 8.3 можно было заметить, что для трех полей таблицы объявлены типы данных VARCHAR, а для одного — тип данных CHAR. Термин VARCHAR означает *VARiable length CHARacter string* — строка символов переменной длины, а команда воспринимает числовое значение, указывающее MySQL максимальную длину, разрешенную для строки, хранящейся в этом поле.

Этот тип данных очень удобен, поскольку позволяет MySQL планировать размер базы данных и эффективнее осуществлять просмотр и поиск данных. Но есть у него и недостаток: если присвоить строковое значение длиннее позволенного, оно будет усечено до максимальной длины, объявленной в определении таблицы.

Но у поля year (год) более предсказуемые значения, поэтому вместо VARCHAR для него используется более подходящий тип данных — CHAR(4). Параметр 4 позволяет выделить под него 4 байта данных, поддерживающих все года от -999 и до 9999. Можно было бы, конечно, сохранять год и в значении, состоящем из двух цифр, но если данные не утратят своей актуальности и в следующем столетии или показатель лет каким-то образом опять вернется к нулевому значению, то эту проблему нужно решать в первоочередном порядке, поскольку она очень похожа на «проблему 2000 года» из-за которой даты начиная с 1 января 2000 года на многих самых крупных в мире компьютерных системах могли быть отнесены к 1900 году.

Тип данных YEAR не выбран для хранения года в таблице classics потому, что он поддерживает только 0000 год и диапазон лет с 1901-го по 2155-й. MySQL из соображений эффективности хранит значение года в 1 байте, а это значит, что храниться могут только 256 значений, в то время как книги в таблице classics изданы задолго до сохраняемого диапазона лет.

Оба типа данных, и CHAR, и VARCHAR, принимают строки текста, ограничивая их длину размером поля. Разница между ними состоит в том, что каждая строка в поле CHAR имеет указанный размер. Если поместить в него строку меньшего размера, она будет дополнена пробелами. В поле VARCHAR дополнения текста не происходит; его размер может изменяться таким образом, чтобы в него помещался вставленный текст. Но при использовании поля VARCHAR требуется идти на небольшие издержки, чтобы отслеживать размер каждого значения. Поэтому CHAR больше подходит для тех случаев, когда данные во всех записях имеют одинаковый размер, а VARCHAR эффективнее применять, когда размеры могут сильно отличаться друг от друга и возрастать. Но за это приходится расплачиваться тем, что доступ к данным типа VARCHAR осуществляется несколько медленнее, чем к данным типа CHAR.

## Тип данных CHAR

В табл. 8.6 перечислены все типы символьных данных CHAR. Все они предлагают указать параметр, устанавливающий максимальную (или точную) длину строки, которая может быть помещена в поле. Из таблицы следует, что у каждого типа есть присущее ему максимальное значение длины. Для данных типа VARCHAR длиной от 0 до 255 байт требуется еще 1 байт в хранилище, а для данных длиной более 256 байт требуется еще 2 байта.

**Таблица 8.6.** Типы данных CHAR, используемые в MySQL

| Тип данных | Количество байтов | Примеры |
|---|---|---|
| CHAR(*n*) | В точности равное *n* (≤ 255) | CHAR(5) Hello использует 5 байт<br>CHAR(57) New York использует 57 байт |
| VARCHAR(*n*) | Вплоть до *n* (≤ 65 535) | VARCHAR(100) Greetings использует 9 байт плюс еще 1 байт<br>VARCHAR(7) Morning использует 7 байт плюс еще 1 байт |

## Тип данных BINARY

Тип данных BINARY используется для хранения строк, заполненных байтами, не имеющими никакой связи с таблицей символов (табл. 8.7). Например, тип данных BINARY можно использовать для хранения изображения в формате GIF.

**Таблица 8.7.** Типы данных BINARY, используемые в MySQL

| Тип данных | Количество байтов | Примеры |
|---|---|---|
| BINARY(*n*) или BYTE(*n*) | В точности равное *n* (≤ 255) | Похож на CHAR, но содержит двоичные данные |
| VARBINARY(*n*) | Вплоть до *n* (≤ 65 535) | Похож на VARCHAR, но содержит двоичные данные |

## Типы данных TEXT и VARCHAR

Типы данных TEXT и VARCHAR имеют незначительные отличия друг от друга.

- До выхода версии 5.0.3 MySQL удаляла из полей VARCHAR все начальные и замыкающие пробелы, и они могли быть длиной только до 256 байт.
- В полях типа TEXT не может быть исходных значений.
- В столбце TEXT MySQL индексирует только первые *n* символов (*n* задается при создании индекса).

Это означает, что VARCHAR является более приемлемым и быстрее обрабатываемым типом данных, если нужно вести поиск по всему содержимому поля. Если поиск никогда не будет выполняться более чем в конкретном количестве начальных символов хранящегося в поле значения, то, наверное, нужно остановить свой выбор на типе данных TEXT (табл. 8.8).

**Таблица 8.8.** Типы данных TEXT, используемые в MySQL

| Тип данных | Количество байтов | Особенности |
|---|---|---|
| TINYTEXT(*n*) | Вплоть до *n* (≤ 255) | Считается строкой с набором символов |
| TEXT(*n*) | Вплоть до *n* (≤ 65 535) | Считается строкой с набором символов |
| MEDIUMTEXT(*n*) | Вплоть до *n* (≤ 16 777 215) | Считается строкой с набором символов |
| LONGTEXT(*n*) | Вплоть до *n* (≤ 4 294 967 295) | Считается строкой с набором символов |

## Тип данных BLOB

Термин BLOB означает Binary Large Object — «большой двоичный объект», и поэтому, как и можно было предположить, тип данных BLOB больше всего подходит для хранения двоичных данных, превышающих по объему 65 536 байт. Другим основным отличием BLOB от типа данных BINARY является то, что для столбцов типа BLOB нельзя задавать исходные значения (табл. 8.9).

**Таблица 8.9.** Типы данных BLOB, используемые в MySQL

| Тип данных | Количество байтов | Особенности |
|---|---|---|
| TINYBLOB(*n*) | Вплоть до *n* (≤ 255) | Считается не набором символов, а двоичными данными |
| BLOB(*n*) | Вплоть до *n* (≤ 65 535) | Считается не набором символов, а двоичными данными |
| MEDIUMBLOB(*n*) | Вплоть до *n* (≤ 16 777 215) | Считается не набором символов, а двоичными данными |
| LONGBLOB(*n*) | Вплоть до *n* (≤ 4 294 967 295) | Считается не набором символов, а двоичными данными |

## Числовые типы данных

В MySQL поддерживаются различные числовые типы данных, от одиночного байта до чисел с плавающей точкой с удвоенной точностью. Хотя для числового поля можно использовать до 8 байт, лучше все же выбрать поле с самым скромным типом данных, в котором способно уместиться наибольшее из ожидаемых вами значений. Тогда ваша база данных будет небольшой по объему и быстрой по доступу.

В табл. 8.10 перечислены числовые типы данных, поддерживаемые MySQL, и диапазоны значений, которые могут содержаться в их полях. Если вы не знакомы с терминологией, поясню, что число *со знаком* имеет диапазон возможных значений от отрицательного до нуля и от нуля до положительного значения, а число *без знака* может быть в диапазоне от нуля до положительного значения. Оба они могут иметь одинаковую величину, нужно лишь представить число со знаком сдвинутым наполовину влево, с одной половиной в отрицательном, а с другой — в положительном диапазоне. Следует заметить, что значения с плавающей точкой (любой точности) могут быть только числами со знаком.

**Таблица 8.10.** Числовые типы данных, используемые в MySQL

| Тип данных | Количество байт | Минимальное значение (со знаком/без знака) | Максимальное значение (со знаком/без знака) |
|---|---|---|---|
| TINYINT | 1 | –128<br>0 | 127<br>255 |
| SMALLINT | 2 | –32 768<br>0 | 32 767<br>65 535 |
| MEDIUMINT | 3 | –8 388 608<br>0 | 8 388 607<br>16 777 215 |
| INT или INTEGER | 4 | –2 147 483 648<br>0 | 2 147 483 647<br>4 294 967 295 |
| BIGINT | 8 | –9 223 372 036 854 775 808<br>0 | 9 223 372 036 854 775 807<br>18 446 744 073 709 551 615 |
| FLOAT | 4 | –3,402823466E+38<br>(без знака не бывает) | 3.402823466E+38<br>(без знака не бывает) |
| DOUBLE или REAL | 8 | –1,7976931348623157E–308<br>(без знака не бывает) | 1.7976931348623157E+308<br>(без знака не бывает) |

Чтобы указать, какой именно тип данных используется, со знаком или без знака, применяется спецификатор UNSIGNED. В следующем примере создается таблица по имени tablename, содержащая поле fieldname с типом данных UNSIGNED INTEGER:

```
CREATE TABLE tablename (fieldname INT UNSIGNED
```

При создании числового поля можно также передать в качестве параметра необязательное число:

```
CREATE TABLE tablename (fieldname INT(4)
```

Но при этом следует помнить, что в отличие от типов данных BINARY и CHAR этот параметр не показывает количество байтов, выделяемых под хранение. Может быть, это противоречит интуитивному восприятию, но на самом деле это число обозначает отображаемую ширину данных в поле при его извлечении. Оно часто используется вместе со спецификатором ZEROFILL:

```
CREATE TABLE tablename (fieldname INT(4) ZEROFILL
```

Этот спецификатор указывает на то, что все числа шириной меньше четырех символов дополняются одним или несколькими нулями, для того чтобы ширина

отображаемого поля составляла четыре символа. Если поле уже занимает четыре и более символа, дополнение не производится.

## Типы данных DATE и TIME

В табл. 8.11 показана еще одна важная категория типов данных, поддерживаемая MySQL, которая относится к дате и времени.

**Таблица 8.11.** Типы данных DATE и TIME, используемые в MySQL

| Тип данных | Формат времени-даты |
|---|---|
| DATETIME | '0000-00-00 00:00:00' |
| DATE | '0000-00-00' |
| TIMESTAMP | '0000-00-00 00:00:00' |
| TIME | '00:00:00' |
| YEAR | 0000 (только годы 0000 и 1901–2155) |

Значения, имеющие типы данных DATETIME и TIMESTAMP, отображаются одинаково. Основное различие в том, что у TIMESTAMP слишком узкий диапазон (от 1970 и до 2037 года), а в DATETIME может храниться практически любая нужная дата, если только вы не интересуетесь античной историей или научной фантастикой.

Но TIMESTAMP также полезен, потому что, используя его, можно позволить MySQL установить для вас нужное значение. Если при добавлении строки не задавать значение для поля с этим типом данных, то в него автоматически будет вставлено текущее время. Можно также заставить MySQL обновлять столбец с типом данных TIMESTAMP при каждом изменении строки.

## Тип данных AUTO_INCREMENT

Иногда нужно обеспечить уникальность каждой строки, имеющейся в базе данных. В вашей программе это можно сделать за счет тщательной проверки вводимых данных и обеспечения их различия хотя бы в одном из значений в любых двух строках. Но такой подход не защищен от ошибок и работает только в конкретных обстоятельствах. Например, в таблице один и тот же автор может появляться несколько раз. Точно так же, скорее всего, будет повторяться год издания и т. д. В таком случае гарантировать отсутствие продублированных строк будет довольно трудно.

В общем виде эта проблема решается за счет специально выделенного для этой цели дополнительного столбца. Вскоре мы рассмотрим использование ISBN (International Standard Book Number — международный стандартный книжный номер) издания для обеспечения уникальности строк в таблице classics, но сначала нужно представить вам тип данных с автоприращением — AUTO_INCREMENT.

В соответствии с названием столбца, которому назначен этот тип данных, его содержимому будет устанавливаться значение на единицу большее, чем значение записи в этом же столбце в предыдущей вставленной строке. В примере 8.5 показано, как нужно добавлять новый столбец по имени id к таблице classics и придавать ему свойства автоприращения.

**Пример 8.5.** Добавление столбца id с автоприращением

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

Здесь представлена команда ALTER, очень похожая на команду CREATE. Команда ALTER работает с уже существующей таблицей и может добавлять, изменять или удалять столбцы. В нашем примере добавляется столбец по имени id, имеющий следующие характеристики:

- INT UNSIGNED — делает столбец способным принять целое число, достаточно большое для того, чтобы в таблице могло храниться более 4 млрд записей;
- NOT NULL — обеспечивает наличие значения в каждой записи столбца. Многие программисты используют его в поле NULL, чтобы показать отсутствие в нем какого-либо значения. Но тогда могут появляться дубликаты, противоречащие самому смыслу существования этого столбца. Поэтому появление в нем значения NULL запрещено;
- AUTO_INCREMENT — заставляет MySQL установить для этого столбца уникальное значение в каждой строке, как было описано ранее. Фактически мы не управляем значением, которое будет появляться в каждой строке этого столбца, но это и не нужно: все, о чем мы беспокоимся, — гарантия уникальности этого значения;
- KEY — столбец с автоприращением полезно использовать в качестве ключа, поскольку вы будете стремиться искать строки на основе значений этого столбца. Пояснения будут даны в разделе «Индексы» далее.

Теперь каждая запись будет иметь уникальное число в столбце id, для первой записи это будет начальное число 1, а счет других записей будет вестись по нарастающей. Как только будет вставлена новая строка, в ее столбец id будет автоматически записано следующее по порядку число.

Этот столбец можно не добавлять после создания таблицы, а сразу включить в нее, слегка изменив формат команды CREATE. В данном случае команда из примера 8.3 должна быть заменена командой из примера 8.6. Обратите особое внимание на ее последнюю строку.

**Пример 8.6.** Добавление столбца id с автоприращением при создании таблицы

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 type VARCHAR(16),
 year CHAR(4),
 id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE MyISAM;
```

Если хочется проверить, был ли добавлен столбец, нужно посмотреть имеющиеся в таблице столбцы и типы данных, воспользовавшись следующей командой:

```
DESCRIBE classics;
```

Теперь, когда мы закончили изучение этого типа данных, столбец id нам больше не нужен, поэтому, если вы его создали, воспользовавшись командой из примера 8.5, его нужно удалить, воспользовавшись командой из примера 8.7.

**Пример 8.7.** Удаление столбца id

```
ALTER TABLE classics DROP id;
```

## Добавление данных к таблице

Для добавления данных к таблице используется команда INSERT. Рассмотрим ее в действии, заполнив таблицу classics данными из таблицы 8.1, многократно используя одну и ту же форму команды INSERT (пример 8.8).

**Пример 8.8.** Заполнение таблицы classics

```
INSERT INTO classics(author, title, type, year)
 VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');
INSERT INTO classics(author, title, type, year)
 VALUES('Jane Austen','Pride and Prejudice','Fiction','1811');
INSERT INTO classics(author, title, type, year)
 VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');
INSERT INTO classics(author, title, type, year)
 VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');
INSERT INTO classics(author, title, type, year)
 VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
```

После каждой второй строки вы должны увидеть сообщение об успешной обработке запроса — Query OK. Как только будут введены все строки, наберите следующую команду, которая отобразит содержимое таблицы:

```
SELECT * FROM classics;
```

Результат должен быть похож на тот, что показан на рис. 8.4.

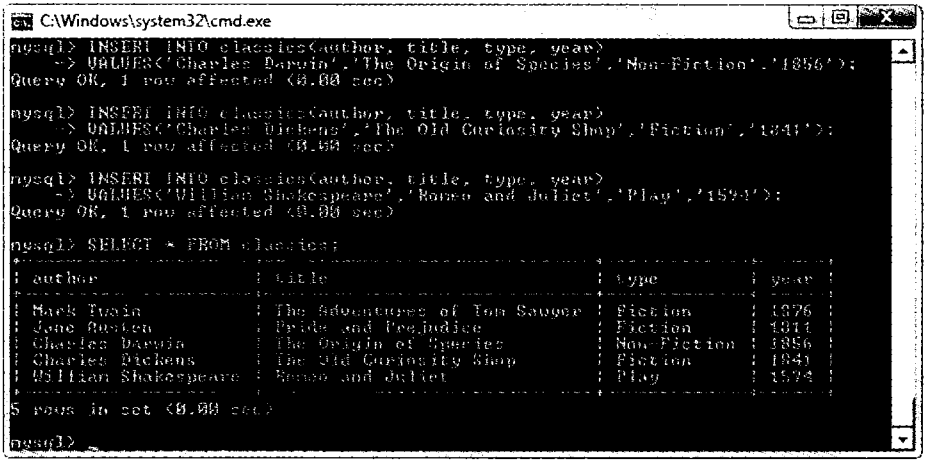



**Рис. 8.4.** Заполнение таблицы classics и просмотр ее содержимого

Сейчас не стоит обращать внимания на команду SELECT, ее очередь наступит в разделе «Запросы к базе данных MySQL с помощью PHP». Достаточно сказать, что в данном виде она отображает все только что введенные данные.

Теперь вернемся назад и посмотрим, как используется команда INSERT. Ее первая часть, INSERT INTO classics, сообщает MySQL, куда нужно вставлять следующие за ней данные. Затем в круглых скобках перечисляются четыре имени столбцов: author, title, type и year, которые отделяются друг от друга запятыми. Таким образом MySQL сообщается, что именно в эти четыре поля будут вставляться данные.

Во второй строке каждой команды INSERT содержится ключевое слово VALUES, за которым следуют четыре строковых значения, взятых в кавычки и отделенных друг от друга запятыми. Они обеспечивают MySQL теми четырьмя значениями, которые будут вставлены в четыре ранее указанных столбца. (Как и во всех остальных примерах, разбиение команды на строки было всего лишь моим собственным решением, придерживаться которого не обязательно.)

Каждый элемент данных будет вставлен по порядку в соответствующие столбцы. Если порядок перечисления столбцов и данных будет случайно перепутан, данные попадут не в те столбцы. А количество указанных столбцов должно соответствовать количеству элементов данных.

## Переименование таблиц

Переименование таблицы, как и любые другие изменения ее структуры или метаданных, осуществляется посредством команды ALTER. Поэтому, чтобы, к примеру, изменить имя таблицы classics на pre1900, воспользуйтесь следующей командой:

```
ALTER TABLE classics RENAME pre1900;
```

Если применить эту команду, то потом, чтобы без изменений работали все последующие примеры этой главы, вам придется вернуть таблице ее прежнее имя, для чего нужно будет ввести такую команду:

```
ALTER TABLE pre1900 RENAME classics;
```

## Изменение типа данных столбца

Для изменения типа данных столбца также используется команда ALTER, но в этом случае вместе с ней применяется ключевое слово MODIFY. Поэтому для изменения типа данных столбца year с CHAR(4) на SMALLINT (для которого потребуется только 2 байта памяти, что способствует экономии дискового пространства), нужно ввести следующую команду:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

После этого, если для MySQL есть смысл конвертировать тип данных, система автоматически изменит данные, сохраняя их значение. В данном случае она заменит каждое строковое значение сопоставимым с ним целым числом, пока строку можно будет распознать как отображение целого числа.

## Добавление нового столбца

Предположим, что таблица создана и заполнена большим объемом данных и тут выяснилось, что нужен еще один столбец. Не стоит расстраиваться. Посмотрите, как можно добавить к таблице новый столбец pages, который будет использоваться для хранения количества страниц, имеющихся в книге:

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Эта команда добавляет новый столбец по имени pages, в котором используется тип данных UNSIGNED SMALLINT, подходящий для хранения значений вплоть до 65 535. Этого наверняка более чем достаточно для любой когда-либо изданной книги!

И если запросить у MySQL описание обновленной таблицы, воспользовавшись показанной далее командой DESCRIBE, то можно будет увидеть внесенные в нее изменения (рис. 8.5):

```
DESCRIBE classics;
```

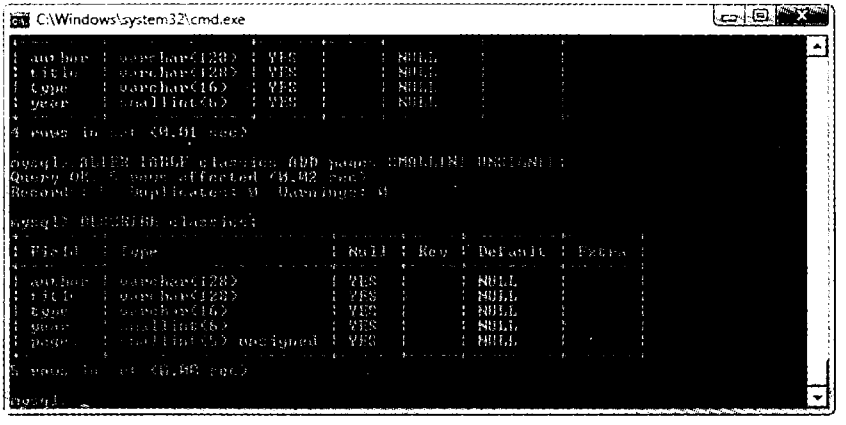



**Рис. 8.5.** Добавление нового столбца pages и просмотр таблицы

## Переименование столбца

Посмотрев еще раз на рис. 8.5, можно заметить, что наличие в ней столбца type может привести к путанице, поскольку такое же имя используется MySQL для идентификации типа данных. Но это не проблема — давайте изменим имя этого столбца на category:

```
ALTER TABLE classics CHANGE type category VARCHAR(16);
```

Обратите внимание на добавление VARCHAR(16) в конце этой команды. Это связано с тем, что ключевое слово CHANGE требует указания типа данных даже в том случае, если вы не собираетесь его изменять, и VARCHAR(16) — это тот самый тип данных, который был указан при создании столбца type.

## Удаление столбца

Поразмыслив, можно прийти к выводу, что столбец pages, в котором хранится количество страниц, не представляет для данной базы данных особой ценности, поэтому его можно удалить, используя ключевое слово DROP:

```
ALTER TABLE classics DROP pages;
```

Учтите, что ключевое слово DROP нужно применять с особой осторожностью, поскольку его действие носит необратимый характер и по недоразумению можно удалить целые таблицы (и даже базы данных)!

## Удаление таблицы

Удалить таблицу очень просто. Но поскольку я не хочу заставлять вас заново вводить все данные в таблицу `classics`, мы ее удалять не станем. Давайте вместо этого просто создадим новую таблицу, проверим факт ее существования, а затем удалим ее, набрав команду, приведенную в примере 8.9. Результат выполнения всех четырех команд показан на рис. 8.6.

**Пример 8.9.** Создание, просмотр и удаление таблицы

```
CREATE TABLE disposable(trash INT);
DESCRIBE disposable;
DROP TABLE disposable;
SHOW tables;
```


**Рис. 8.6.** Создание, просмотр и удаление таблицы

# Индексы

В данный момент у нас есть действующая таблица `classics`, в которой можно будет без труда, пользуясь средствами MySQL, отыскать нужную информацию. Но все так просто лишь до тех пор, пока она не разрастется до пары сотен строк. Тогда с каждой добавленной строкой доступ к базе данных будет становиться все медленнее и медленнее, поскольку MySQL при обработке запроса придется вести поиск в каждой строке. Это похоже на поиск нужной информации в каждой книге, имеющейся в библиотеке.

Разумеется, вам не придется вести поиск в библиотеках подобным образом, поскольку в них есть либо обычная картотека, либо, что более вероятно, собственная база данных. То же самое относится и к MySQL, поскольку ценой небольших затрат оперативной памяти и дискового пространства можно создать «картотеку» для таблицы, которая будет использоваться MySQL для выполнения мгновенного поиска.

## Создание индекса

Возможности быстрого поиска можно добиться путем добавления *индекса* либо при создании таблицы, либо в любое время впоследствии. Но сделать это не так-то просто. Существуют, к примеру, различные типы индексов, такие как INDEX, PRIMARY KEY и FULLTEXT. Кроме того, нужно решить, каким столбцам нужен индекс, а для этого нужно спрогнозировать, по каким данным этих столбцов будет осуществляться поиск. Индексы можно усложнять, комбинируя в одном индексе данные из нескольких столбцов. И даже когда вы все это поймете, у вас будет возможность сократить размер индекса за счет ограничения объема данных каждого индексируемого столбца.

Если представить себе поисковые операции применительно к таблице classics, становится ясно, что поиск может осуществляться во всех столбцах. Но если бы не был удален столбец pages, созданный в пункте «Добавление нового столбца» выше, то он, наверное, не понадобился бы для индекса, поскольку большинство людей вряд ли стали бы искать книги по количеству страниц. Давайте все же продолжим и добавим индекс к каждому столбцу, воспользовавшись командами, приведенными в примере 8.10.

**Пример 8.10.** Добавление индексов к таблице classics

```
ALTER TABLE classics ADD INDEX(author(20));
ALTER TABLE classics ADD INDEX(title(20));
ALTER TABLE classics ADD INDEX(category(4));
ALTER TABLE classics ADD INDEX(year);
DESCRIBE classics;
```

Первые две команды создают индексы для столбцов авторов и названий — author и title, ограничивая каждый индекс только первыми двадцатью символами. Например, когда MySQL индексирует название:

```
The Adventures of Tom Sawyer
```

то на самом деле в индексе будут сохранены только первые 20 символов:

```
The Adventures of To
```

Это делается для сокращения размера индекса и для оптимизации скорости доступа к базе данных. Я выбрал 20 символов, поскольку их должно быть достаточно для обеспечения уникальности большинства строк, встречающихся в данных столбцах. Если MySQL обнаружит два индекса с одинаковым содержимым, ей нужно будет понапрасну потратить время на обращение к самой таблице и на проверку проиндексированного столбца, для того чтобы определить, какая именно строка действительно соответствует условиям поиска.

Что касается столбца категории — category, то на данный момент, чтобы идентифицировать уникальность строки, достаточно только первого символа (F для Fiction, N для Non-Fiction и P для Play), но я выбрал индекс из четырех символов, чтобы дать возможность в будущем вводить такие категории, уникальность которых можно будет определить только по четырем символам. (Если позже набор категорий усложнится еще больше, этот столбец можно будет переиндексировать.)

И наконец, я не стал задавать ограничения на индекс столбца года издания — year, поскольку в нем хранятся не строки, а целые числа.

Результат ввода этих команд (и команды DESCRIBE, позволяющей убедиться в том, что они работают) можно увидеть на рис. 8.7, который показывает наличие ключа MUL для каждого столбца. Это означает, что в этом столбце может многократно присутствовать одно и то же значение, что, собственно, нам и нужно, поскольку имена авторов могут встречаться многократно, одни и те же названия книг могут использоваться множеством авторов и т. д.

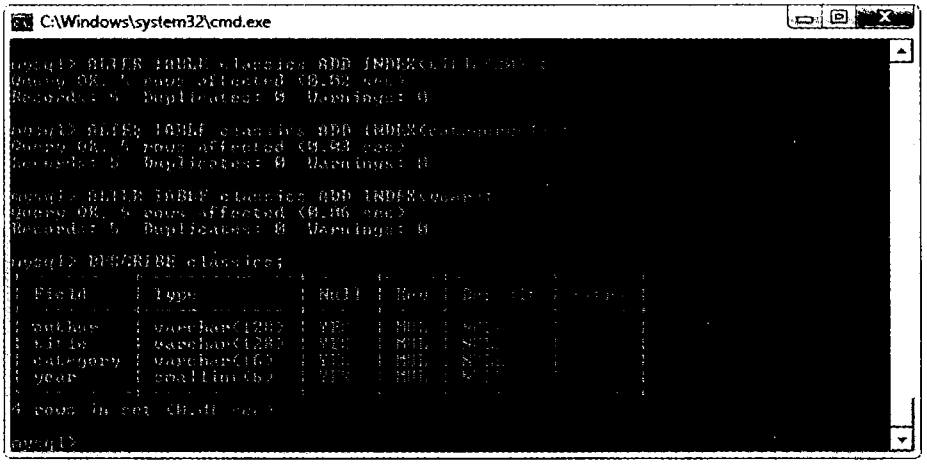



**Рис. 8.7.** Добавление индексов к таблице classics

## Использование команды CREATE INDEX

Индекс можно добавить не только командой ALTER TABLE, но и командой CREATE INDEX. Эти две команды являются равнозначными, за исключением того, что CREATE INDEX не может использоваться для создания индекса типа первичного ключа — PRIMARY KEY (см. пункт «Первичные ключи» далее). Формат этой команды показан во второй строке примера 8.11.

**Пример 8.11.** Эти две команды являются эквивалентными

```
ALTER TABLE classics ADD INDEX(author(20));
CREATE INDEX author ON classics (author(20));
```

## Добавление индексов при создании таблиц

Чтобы добавить индекс, не нужно выжидать какое-то время после создания таблицы. Это может отнять много времени, поскольку добавление индекса к большой таблице — длительный процесс. Поэтому давайте рассмотрим команду, создающую таблицу classics с уже имеющимися индексами.

Пример 8.12 является переработкой примера 8.3, в котором одновременно с таблицами создаются и индексы. Учтите, что для включения всех изменений, выполненных в данной главе, в этой версии используется новое имя столбца category вместо прежнего имени type, а для столбца year указан тип данных SMALLINT, а не

CHAR(4). При желании попробовать эту команду в работе без предварительного удаления текущей таблицы classics замените слово classics в первой строке каким-нибудь другим словом, например classics1, а после завершения работы удалите таблицу classics1.

**Пример 8.12.** Создание таблицы classics с индексами

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 category VARCHAR(16),
 year SMALLINT,
 INDEX(author(20)),
 INDEX(title(20)),
 INDEX(category(4)),
 INDEX(year)) ENGINE MyISAM;
```

## Первичные ключи

В данный момент у нас создана таблица classics и за счет добавления индексов обеспечен быстрый поиск, но кое-что все же упущено. Можно вести поиск по всем имеющимся в таблице изданиям, но нет единого уникального ключа для каждого издания, обеспечивающего мгновенный доступ к строке. Важность наличия ключа с уникальным значением для каждой строки (известного как *первичный ключ*) проявится, когда мы станем комбинировать данные из разных таблиц (см. подраздел «Первичные ключи: ключи к реляционным базам данных» раздела «Проектирование базы данных» главы 9).

В представленном ранее пункте «Тип данных AUTO_INCREMENT» подраздела «Типы данных» предыдущего раздела, в котором рассматривался создаваемый столбец id с автоприращением, было сказано, что он может быть использован в качестве первичного ключа для этой таблицы. Но я захотел возложить эту задачу на более подходящий столбец: признанный во всем мире номер ISBN.

Поэтому продолжим работу с таблицей и создадим новый столбец для этого ключа. Теперь, помня о том, что номер ISBN состоит из 13 символов, можно решить, что с этой задачей справится следующая команда:

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Но это не так. Если запустить эту команду на выполнение, будет получено сообщение об ошибке, связанной с дубликатом записи для ключа 1: Duplicate entry. Причина в том, что таблица уже заполнена данными, а эта команда пытается добавить столбец со значением NULL к каждой строке, что запрещено, поскольку все столбцы, использующие первичный ключ, должны иметь уникальное значение. Но если бы таблица была пуста, то эта команда была бы выполнена без проблем, как и при добавлении первичного ключа сразу же после создания таблицы.

В сложившейся ситуации нужно немного схитрить: создать новый столбец без индекса, заполнить его данными, а затем воспользоваться командой из примера 8.13. К счастью, в этом наборе данных каждый год имеет уникальное значение, поэтому для идентификации каждой обновляемой строки можно воспользоваться столбцом

year. Учтите, что в этом примере применяются ключевые слова UPDATE и WHERE, которые более подробно будут рассмотрены в подразделе «Создание запросов к базе данных MySQL» далее.

**Пример 8.13.** Заполнение столбца isbn данными и использование первичного ключа

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
ALTER TABLE classics ADD PRIMARY KEY(isbn);
DESCRIBE classics;
```

После ввода этих команд будет получен результат, похожий на копию экрана, показанную на рис. 8.8. Обратите внимание на то, что в синтаксисе команды ALTER TABLE ключевое слово INDEX заменено ключевыми словами PRIMARY KEY (сравните примеры 8.10 и 8.13).


**Рис. 8.8.** Добавление первичного ключа к таблице classics

Чтобы создать первичный ключ при создании таблицы classics, можно воспользоваться командой, показанной в примере 8.14. И в этом случае, если вы хотите испробовать эту команду в работе, нужно заменить имя classics в строке 1 каким-нибудь другим, а затем удалить проверочную таблицу.

**Пример 8.14.** Создание таблицы classics вместе с индексами

```
CREATE TABLE classics (
 author VARCHAR(128),
 title VARCHAR(128),
 category VARCHAR(16),
 year SMALLINT,
 isbn CHAR(13),
```

```
INDEX(author(20)),
INDEX(title(20)),
INDEX(category(4)),
INDEX(year),
PRIMARY KEY (isbn)) ENGINE MyISAM;
```

## Создание индекса FULLTEXT

В отличие от обычного индекса, имеющийся в MySQL индекс FULLTEXT позволяет осуществлять сверхбыстрый поиск целых столбцов текста. Он сохраняет каждое слово каждой строки данных в специальном индексе, в котором можно вести поиск, используя «естественный язык» наподобие того, что используется в поисковом механизме.

Вообще-то утверждение о том, что система MySQL хранит все слова в индексе FULLTEXT, не вполне соответствует действительности, поскольку в ней имеется встроенный список более чем из 500 слов, которые она предпочитает игнорировать в силу их широкой распространенности и практической бесполезности при любом поиске. Это список, называемый стоповыми словами — stopwords, включает слова the, as, is, of и т. д. Список помогает MySQL работать при FULLTEXT-поиске намного быстрее и не раздувать размеры базы данных. Полный список стоповых слов приведен в приложении В.

Рассмотрим некоторые особенности индексов FULLTEXT, о которых нужно знать.

- Индексы FULLTEXT могут использоваться только с таблицами типа MyISAM, использующими исходное ядро (механизм хранения) MySQL (MySQL поддерживает как минимум десять различных ядер). Если нужно привести таблицу к типу MyISAM, можно применить команду MySQL:

  ```
  ALTER TABLE tablename ENGINE = MyISAM;
  ```

- Индексы FULLTEXT могут создаваться только для столбцов с типами данных CHAR, VARCHAR и TEXT.
- Определение индекса FULLTEXT может быть дано в инструкции CREATE TABLE при создании таблицы или добавлено позже с использованием инструкции ALTER TABLE (или CREATE INDEX).
- *Намного* быстрее будет загрузить большие наборы данных в таблицу, не имеющую индекса FULLTEXT, а затем создать индекс, чем загружать их в таблицу, у которой уже имеется индекс FULLTEXT.

Чтобы создать индекс FULLTEXT, примените его к одной или нескольким записям, как в примере 8.15, в котором индекс FULLTEXT добавляется к двум столбцам — author и title, принадлежащим таблице classics (этот индекс является дополнением к тем, что уже были созданы, и не влияет на их работу).

**Пример 8.15.** Добавление индекса FULLTEXT к таблице classics

```
ALTER TABLE classics ADD FULLTEXT(author,title);
```

Теперь в этой паре столбцов можно вести поиск с использованием индекса FULLTEXT. Эта возможность могла бы проявиться в полную силу, если бы вы могли

теперь ввести весь текст этих книг в базу данных (учитывая, что они не защищены авторскими правами), тогда они были бы полностью доступны для поиска. Поисковые операции с использованием индекса FULLTEXT рассмотрены далее в пункте «MATCH...AGAINST» подраздела «Создание запросов к базе данных MySQL».

Если система MySQL станет при доступе к вашей базе данных работать медленнее, чем вы от нее ожидали, то проблема чаще всего заключается в ваших индексах. Либо у вас нет индекса там, где он нужен, либо индексы составлены неоптимальным образом. Зачастую данная проблема решается за счет тонкой настройки индексов таблиц. Производительность не входит в тематику данной книги, но в главе 9 будет дан ряд подсказок, чтобы вы знали, что именно нужно искать.

# Создание запросов к базе данных MySQL

Итак, мы создали базу данных MySQL и таблицы, заполнили их данными и добавили к ним индексы, чтобы ускорить поиск. Теперь настало время посмотреть, как именно ведется этот поиск и какие для этого имеются команды и спецификаторы.

## SELECT

На рис. 8.4 уже было показано, что команда SELECT используется для извлечения данных из таблицы. В том разделе я воспользовался ее наипростейшей формой для выбора всех данных и их отображения, что вам вряд ли когда-нибудь пригодится, разве что для просмотра самых маленьких таблиц, поскольку все данные будут прокручиваться на экране и скрываться в нечитаемой области. А теперь рассмотрим команду SELECT более подробно.

Ее основной синтаксис имеет следующий вид:

```
SELECT что-нибудь FROM имя_таблицы;
```

Этим что-нибудь, как вы уже видели, может быть символ звездочки (*), означающий «каждый столбец», вместо него можно указать какие-нибудь конкретные столбцы. В примере 8.16 показано, как выбрать только автора и название (author и title) и только название и ISBN. Результат выполнения этих команд показан на рис. 8.9.

**Пример 8.16.** Две разных инструкции SELECT

```
SELECT author,title FROM classics;
SELECT title,isbn FROM classics;
```

## SELECT COUNT

Другим вариантом параметра что-нибудь является функция COUNT, которая может быть использована множеством способов. В примере 8.17 она отображает количество строк в таблице за счет передачи ей в качестве параметра символа звездочки (*), означающего «все строки». В соответствии с вашими ожиданиями будет возвращено число 5, поскольку в таблицу внесены сведения о пяти книгах.

**Пример 8.17.** Подсчет количества строк

```
SELECT COUNT(*) FROM classics;
```

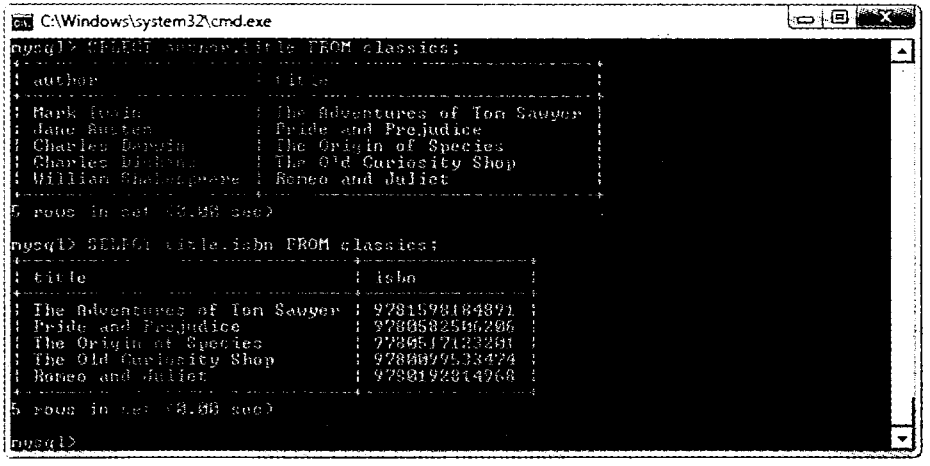



**Рис. 8.9.** Вывод, полученный в результате выполнения двух разных инструкций SELECT

## SELECT DISTINCT

Этот спецификатор (и его синоним DISTINCTROW) позволяет исключать множество записей, имеющих одинаковые данные. Предположим, к примеру, что вам нужно получить список всех авторов, фигурирующих в таблице. Если просто выбрать столбец author из таблицы, содержащей несколько книг одного и того же автора, то будет отображен длинный список с одинаковыми именами авторов, повторяющимися снова и снова. Но за счет добавления ключевого слова DISTINCT можно показать каждого автора всего лишь один раз. Давайте проверим этот спецификатор, добавив еще одну строку, в которой повторяется один из уже имеющихся авторов (пример 8.18).

**Пример 8.18.** Дублирование данных

```
INSERT INTO classics(author, title, category, year, isbn)
 VALUES('Charles Dickens','Little Dorrit','Fiction','1857','9780141439969');
```

Теперь, когда Чарльз Диккенс появляется в таблице дважды, мы можем сравнить результаты использования команды SELECT со спецификатором DISTINCT и без него. В примере 8.19 и на рис. 8.10 показано, что при вводе простой команды SELECT Диккенс будет показан дважды, а команда со спецификатором DISTINCT выводит его только один раз.

**Пример 8.19.** Команда SELECT со спецификатором DISTINCT и без него

```
SELECT author FROM classics;
SELECT DISTINCT author FROM classics;
```

## DELETE

Когда нужно удалить строку из таблицы, используется команда DELETE. Ее синтаксис похож на синтаксис команды SELECT, он позволяет сузить диапазон удаляемой информации до конкретной строки или строк путем использования таких спецификаторов, как WHERE и LIMIT.


**Рис. 8.10.** Выборка данных с использованием спецификатора DISTINCT и без него

Теперь, если вы вводили команду, показанную в примере 8.18, и изучали работу спецификатора DISTINCT, нужно удалить Little Dorrit путем ввода команды, показанной в примере 8.20.

**Пример 8.20.** Удаление новой записи

```
DELETE FROM classics WHERE title='Little Dorrit';
```

В этом примере команда DELETE выдается для всех строк, в столбце title которых содержится строковое значение Little Dorrit.

Ключевое слово WHERE обладает большими возможностями, и очень важно, чтобы оно было набрано правильно. Ошибка может навести команду на не те строки (или вообще ни к чему не привести в том случае, если условию WHERE не будет найдено ни одного соответствия). Поэтому теперь нужно уделить немного внимания этому условию, играющему очень важную роль в языке SQL.

## WHERE

Ключевое слово WHERE позволяет сузить диапазон действия запроса, возвращая только те данные, в отношении которых конкретное выражение возвращает истинное значение. За счет использования оператора равенства = код в примере 8.20 возвращает только те строки, в которых значение столбца title в точности соответствует строке Little Dorrit. В примере 8.21 показаны еще два фрагмента, в которых WHERE используется с оператором =.

**Пример 8.21.** Использование ключевого слова WHERE

```
SELECT author,title FROM classics WHERE author="Mark Twain";
SELECT author,title FROM classics WHERE isbn="9781598184891 ";
```

Применительно к нашей таблице эти две команды отобразят один и тот же результат. Но мы можем без особого труда добавить еще несколько книг Марка Твена, и тогда команда в первой строке отобразит все названия книг, принадлежа-

щих его перу, а команда во второй строке — прежний результат (потому что, как мы знаем, ISBN имеет уникальное значение) — The Adventures of Tom Sawyer. Иными словами, поисковые операции, использующие уникальный ключ, более предсказуемы, и новые доказательства этого вы увидите позже, при рассмотрении роли уникальных и первичных ключей.

При проведении поисковых операций можно также осуществлять проверку на соответствие шаблону, для чего используется спецификатор LIKE, позволяющий вести поиск в разных частях строк. Этот спецификатор должен использоваться с символом % до или после некоторого текста. Если его поместить до текста, это будет означать «что-нибудь до», а если после текста — «что-нибудь после». В примере 8.22 показаны три разных запроса, один из которых предназначен для начала строки, другой — для конца, а третий — для любого места в строке. Результат выполнения этих команд приведен на рис. 8.11.

**Пример 8.22.** Использование спецификатора LIKE

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";
SELECT author,title FROM classics WHERE title LIKE "%Species";
SELECT author,title FROM classics WHERE title LIKE "%and%";
```

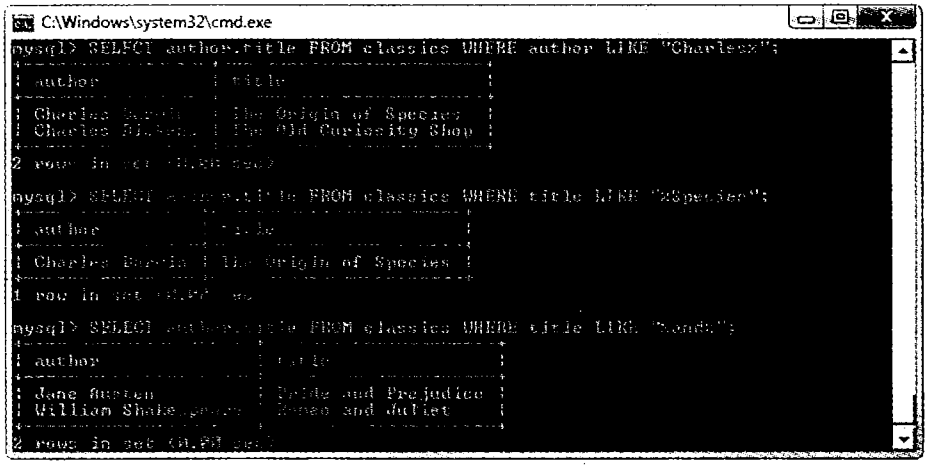



**Рис. 8.11.** Использование ключевого слова WHERE со спецификатором LIKE

Первая команда выведет книги, принадлежащие перу как Чарльза Дарвина, так и Чарльза Диккенса, потому что спецификатор LIKE был настроен на возвращение всего соответствующего строке Charles, за которой следует любой другой текст. Затем будет возвращена информация о книге The Origin of Species, потому что есть только одна строка, столбец которой заканчивается строковым значением Species. И на последний запрос будет возвращена информация о книгах Pride and Prejudice и Romeo and Juliet, потому что обе записи соответствуют запросу строкового значения and в любом месте столбца.

Символ % будет также соответствовать пустому месту в той позиции, которую он занимает. Иными словами, он может соответствовать пустой строке.

## LIMIT

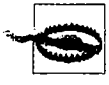
Спецификатор LIMIT позволяет выбрать количество выводимых в запросе строк и место, с которого таблица начнет их возвращать. Когда передается один параметр, он указывает MySQL начать действие спецификатора с верхней части результатов и вернуть только то количество строк, которое задано этим параметром. Если передать спецификатору два параметра, то первый укажет смещение относительно начала результатов, которое MySQL должна учесть при их отображении, а второй укажет, сколько строк нужно вывести. Можно представить, что первый параметр сообщает: «Нужно пропустить это количество результатов, ведя счет сверху».

В пример 8.23 включены три команды. Первая возвращает первые три строки из таблицы. Вторая возвращает две строки, начиная с позиции 1 (пропуская первую строку). А последняя возвращает одну строку, начинающуюся с позиции 3 (пропуская первые три строки). Результаты выполнения всех трех команд показаны на рис. 8.12.

**Пример 8.23.** Ограничение количества возвращаемых результатов

```
SELECT author,title FROM classics LIMIT 3;
SELECT author,title FROM classics LIMIT 1,2;
SELECT author,title FROM classics LIMIT 3,1;
```


**Рис. 8.12.** Ограничение диапазона выводимых строк с помощью спецификатора LIMIT



Ключевое слово LIMIT требует особого внимания, поскольку смещение начинается с нулевой позиции, а количество возвращаемых строк — с единицы. Поэтому спецификатор LIMIT 1,3 означает возвращение трех строк, начиная со *второй* строки.

## MATCH...AGAINST

Конструкция MATCH...AGAINST может быть применена к столбцу, для которого был создан индекс FULLTEXT (см. выше пункт «Создание индекса FULLTEXT»). Используя эту конструкцию, можно вести поиск, применяя в качестве критерия элементы

обычного языка, как при работе с поисковыми механизмами Интернета. В отличие от использования конструкций WHERE...= или WHERE...LIKE, конструкция MATCH... AGAINST позволяет вводить в поисковый запрос несколько слов и проверять на их наличие все слова в столбцах, имеющих индекс FULLTEXT. Индексы FULLTEXT нечувствительны к регистру букв, поэтому неважно, какой именно регистр используется в ваших запросах.

Предположим, что вы добавили индекс FULLTEXT к столбцам author и title и ввели три запроса, показанные в примере 8.24. Первый из них требует вернуть любой из этих столбцов, в котором содержится слово and. Поскольку and является стоповым словом, MySQL его проигнорирует, и запрос всегда будет возвращать пустой набор независимо от того, что хранится в столбцах. Второй запрос требует вернуть любые строки, содержащие в любом месте и в любом порядке оба слова: old и shop. Третий запрос применяет тот же вид поиска для слов tom и sawyer. Результаты выполнения этих запросов показаны на рис. 8.13.

**Пример 8.24.** Использование конструкции MATCH... AGAINST с индексами FULLTEXT

```
SELECT author,title FROM classics
 WHERE MATCH(author,title) AGAINST('and');
SELECT author,title FROM classics
 WHERE MATCH(author,title) AGAINST('old shop');
SELECT author,title FROM classics
 WHERE MATCH(author,title) AGAINST('tom sawyer');
```

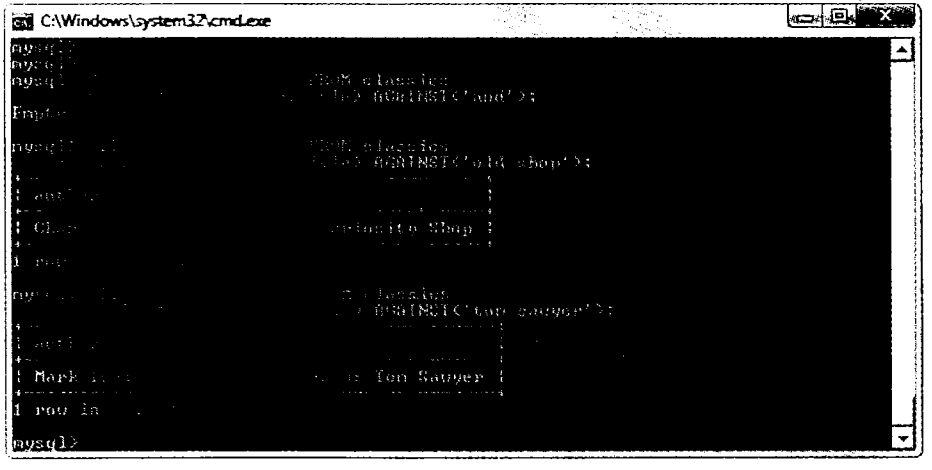



**Рис. 8.13.** Использование конструкции MATCH...AGAINST в индексе FULLTEXT

## MATCH...AGAINST...IN BOOLEAN MODE

При желании придать своим запросам, использующим конструкцию MATCH...AGAINST, более широкие возможности, нужно воспользоваться булевым режимом. Это изменение выражается в том, что стандартный запрос по индексу FULLTEXT ведет поиск любой комбинации искомых слов, не требуя присутствия всех этих слов в тексте. Присутствие отдельного слова в столбце приводит к тому, что поисковая операция возвращает строку.

Булев режим позволяет также ставить впереди искомых слов знак + или -, чтобы показать, что они должны быть включены или исключены. Если обычный булев режим требует «искать присутствие любого из этих слов», то знак «плюс» означает, что «это слово обязательно должно присутствовать, иначе строку возвращать не нужно». Знак «минус» означает, что «этого слова быть не должно, если оно присутствует, то строку возвращать не нужно».

В примере 8.25 показаны два запроса, использующие булев режим. Первый запрос требует вернуть все строки, в которых содержится слово `charles` и нет слова `species`. Во втором запросе используются двойные кавычки, чтобы потребовать вернуть все строки, включающие в себя фразу `origin of`. На рис. 8.14 показаны результаты выполнения этих запросов.

**Пример 8.25.** Использование MATCH...AGAINST...IN BOOLEAN MODE

```
SELECT author,title FROM classics
 WHERE MATCH(author,title)
 AGAINST('+charles -species' IN BOOLEAN MODE);
SELECT author,title FROM classics
 WHERE MATCH(author,title)
 AGAINST('"origin of"' IN BOOLEAN MODE);
```

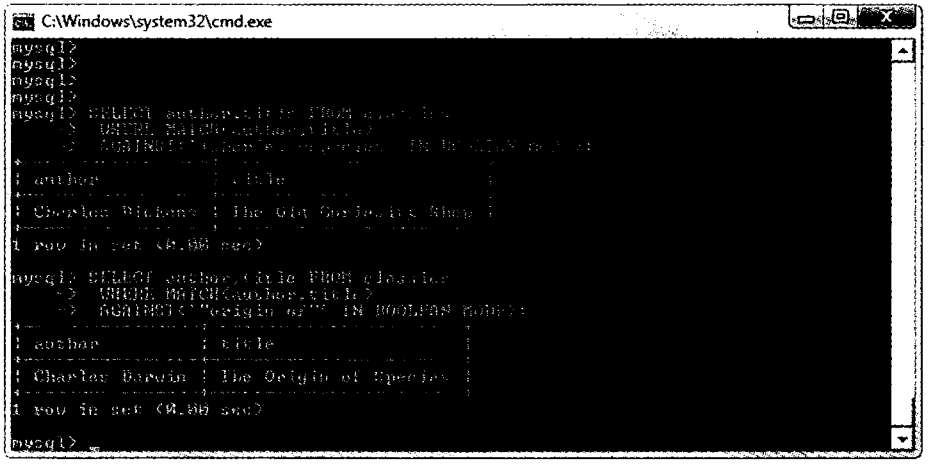



**Рис. 8.14.** Использование конструкции MATCH...AGAINST... IN BOOLEAN MODE

Как, наверное, и ожидалось, первый запрос вернет только запись о книге `The Old Curiosity Shop` Чарльза Диккенса. Запись о книге Чарльза Дарвина игнорируется, поскольку из результата должна быть исключена любая строка, содержащая слово `species`.

Во втором запросе есть кое-что интересное: частью искомой строки является стоповое слово of, но оно все же используется в поиске, поскольку двойные кавычки отменяют учет стоповых слов.

## UPDATE...SET

Эта конструкция позволяет обновлять содержимое поля. Если нужно изменить содержимое одного или нескольких полей, сначала следует сузить область действия запроса до того поля или полей, которые будут подвергаться изменениям, практически тем же способом, который применялся в команде SELECT. В примере 8.26 показаны два разных способа использования UPDATE...SET. Копия экрана с результатами работы этих команд приведена на рис. 8.15.

**Пример 8.26.** Использование UPDATE...SET

```
UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'
 WHERE author='Mark Twain';
UPDATE classics SET category='Classic Fiction'
 WHERE category='Fiction';
```

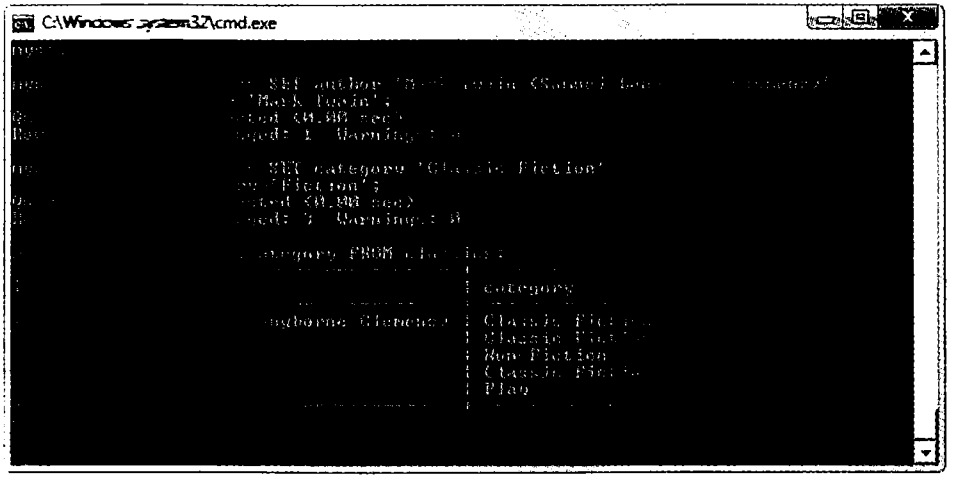



**Рис. 8.15.** Обновление столбцов в таблице classics

В первом запросе действие которого затрагивает только одну строку, к литературному псевдониму Mark Twain добавляется настоящее имя писателя — Samuel Langhorne Clemens заключенное в скобки. А вот второй запрос воздействует на три столбца, поскольку он заменяет все появления слова Fiction в столбце category термином Classic Fiction.

При выполнении обновления можно также воспользоваться такими уже приведенными здесь спецификаторами, как LIMIT, а также рассматриваемыми далее ключевыми словами ORDER BY и GROUP BY.

## ORDER BY

Спецификатор ORDER BY позволяет отсортировать возвращаемые результаты по одному или нескольким столбцам в возрастающем или в убывающем порядке. В примере 8.27 показаны два таких запроса, результаты работы которых можно увидеть на рис. 8.16.

**Пример 8.27.** Использование ORDER BY

```
SELECT author,title FROM classics ORDER BY author;
SELECT author,title FROM classics ORDER BY title DESC;
```

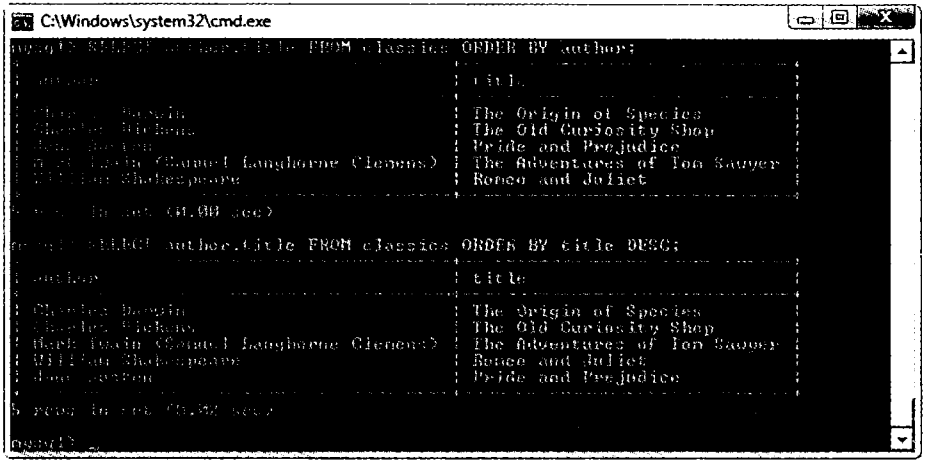



**Рис. 8.16.** Сортировка результатов запроса

Первый запрос возвращает издания, отсортированные по авторам в возрастающем алфавитном порядке (этот режим используется по умолчанию), а второй возвращает их отсортированными по названию в убывающем порядке.

Если нужно отсортировать все столбцы по авторам, а затем в убывающем порядке по году издания (чтобы сначала стояли самые последние), нужно ввести следующий запрос:

```
SELECT author,title,year FROM classics ORDER BY author,year DESC;
```

Здесь показано, что каждый спецификатор сортировки по возрастанию и по убыванию применяется к отдельному столбцу. Ключевое слово DESC применяется только к столбцу, который указан перед ним, — year. Поскольку для столбца author разрешено использовать порядок сортировки, применяемый по умолчанию, этот столбец сортируется в возрастающем порядке. Можно также указать порядок сортировки этого столбца по возрастанию и в явном виде, в итоге будут получены аналогичные результаты:

```
SELECT author,title,year FROM classics ORDER BY author ASC,year DESC;
```

## GROUP BY

Точно так же, как и при использовании ORDER BY, можно сгруппировать результаты, возвращаемые запросом, с помощью спецификатора GROUP BY, который больше всего подходит для извлечения информации о группе данных. Например, если нужно узнать, сколько изданий каждой категории присутствует в таблице classics, можно ввести запрос

```
SELECT category,COUNT(author) FROM classics GROUP BY category;
```

который вернет следующую информацию:

```
+----------------+---------------+
| category       | COUNT(author) |
+----------------+---------------+
| Classic Fiction |             3 |
| Non-Fiction    |             1 |
| Play           |             1 |
+----------------+---------------+
3 rows in set (0.00 sec)
```

# Объединение таблиц

Управление несколькими таблицами, содержащими различные виды информации в одной базе данных, считается вполне обычным делом. Рассмотрим, к примеру, таблицу клиентов — customers, для которой нужно обеспечить возможность использования перекрестных ссылок с приобретенными ими книгами из таблицы classics. Чтобы создать эту новую таблицу и поместить в нее информацию о трех клиентах и их покупках, введите команды из примера 8.28. Результаты показаны на рис. 8.17.

**Пример 8.28.** Создание и заполнение таблицы customers

```
CREATE TABLE customers (
 name VARCHAR(128),
 isbn VARCHAR(128),
 PRIMARY KEY (isbn)) ENGINE MyISAM;
INSERT INTO customers(name,isbn)
 VALUES('Joe Bloggs','9780099533474');
INSERT INTO customers(name,isbn)
 VALUES('Mary Smith','9780582506206');
INSERT INTO customers(name,isbn)
 VALUES('Jack Wilson','9780517123201');
SELECT * FROM customers;
```

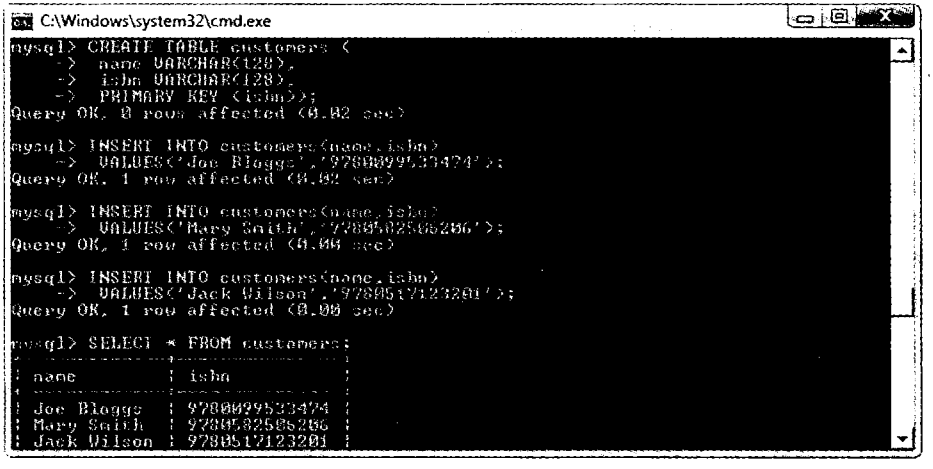



**Рис. 8.17.** Создание таблицы customers

Существует также быстрый способ для вставки сразу нескольких строк данных, как в примере 8.28, позволяющий заменить три отдельных запроса INSERT INTO одним, в котором перечисляются вставляемые данные, отделенные друг от друга запятыми:

```
INSERT INTO customers(name,isbn) VALUES
('Joe Bloggs','9780099533474'),
('Mary Smith','9780582506206'),
('Jack Wilson','9780517123201');
```

Разумеется, в настоящей таблице, содержащей сведения о покупателях, будут присутствовать также адреса, номера телефонов, адреса электронной почты и т. д., но на данном этапе изучения они для нас не представляют интереса.

При создании новой таблицы следует обратить внимание на то, что у нее есть кое-что общее с таблицей `classics`: столбец под названием `isbn`. Поскольку его предназначение в обеих таблицах совпадает (ISBN всегда является ссылкой на одну и ту же книгу), этот столбец можно использовать для связывания двух таблиц вместе в едином запросе, как в примере 8.29.

**Пример 8.29.** Объединение двух таблиц в одном запросе SELECT

```
SELECT name,author,title from customers,classics
 WHERE customers.isbn=classics.isbn;
```

В результате будет выведена следующая информация:

```
+-------------+-----------------+-----------------------+
| name        | author          | title                 |
+-------------+-----------------+-----------------------+
| Joe Bloggs  | Charles Dickens | The Old Curiosity Shop |
| Mary Smith  | Jane Austen     | Pride and Prejudice    |
| Jack Wilson | Charles Darwin  | The Origin of Species  |
+-------------+-----------------+-----------------------+
3 rows in set (0.00 sec)
```

Видите, как этот запрос искусно связал вместе обе таблицы, чтобы продемонстрировать книги из таблицы `classics`, приобретенные покупателями из таблицы `customers`?

## NATURAL JOIN

Используя `NATURAL JOIN`, можно сократить количество вводимого текста и сделать запросы немного более понятными. В этом виде объединения участвуют две таблицы, в которых автоматически объединяются столбцы с одинаковыми именами. Для получения тех же результатов, что и в примере 8.29, можно ввести следующий запрос:

```
SELECT name,author,title FROM customers NATURAL JOIN classics;
```

## JOIN...ON

Если нужно указать столбец, по которому следует объединить две таблицы, используется конструкция `JOIN...ON`, благодаря которой можно получить те же результаты, что и в примере 8.29:

```
SELECT name,author,title FROM customers
 JOIN classics ON customers.isbn=classics.isbn;
```

## Использование ключевого слова AS

Можно сократить количество вводимого текста и улучшить читаемость запроса за счет создания псевдонимов с помощью ключевого слова AS. После имени таблицы нужно поставить AS, а затем используемый псевдоним. Следующий код идентичен по своей работе коду, приведенному в примере 8.29:

```
SELECT name,author,title from
  customers AS cust, classics AS class WHERE cust.isbn=class.isbn;
```

Результат выполнения этой операции имеет следующий вид:

```
+-------------+-----------------+-----------------------+
| name        | author          | title                 |
+-------------+-----------------+-----------------------+
| Joe Bloggs  | Charles Dickens | The Old Curiosity Shop |
| Mary Smith  | Jane Austen     | Pride and Prejudice    |
| Jack Wilson | Charles Darwin  | The Origin of Species  |
+-------------+-----------------+-----------------------+
3 rows in set (0.00 sec)
```

Псевдонимы особенно полезны в длинных запросах, содержащих множественные ссылки на одни и те же имена таблиц.

# Использование логических операторов

Для дальнейшего сужения пространства выбора в запросах MySQL, использующих ключевое слово WHERE, можно также использовать логические операторы AND, OR и NOT. В примере 8.30 показаны варианты применения каждого из них, но их можно использовать в любых сочетаниях.

**Пример 8.30.** Использование логических операторов

```
SELECT author,title FROM classics WHERE
 author LIKE "Charles%" AND author LIKE "%Darwin";
SELECT author,title FROM classics WHERE
 author LIKE "%Mark Twain%" OR author LIKE "%Samuel Langhorne Clemens%";
SELECT author,title FROM classics WHERE
 author LIKE "Charles%" AND author NOT LIKE "%Darwin";
```

Первый запрос выбран потому, что Чарльз Дарвин может фигурировать в некоторых строках под своим полным именем — Чарльз Роберт Дарвин. А запрос возвращает сведения о книгах, для которых значение столбца author начинается с Charles и заканчивается Darwin. Второй запрос ищет книги, принадлежащие перу Марка Твена, используя для этого либо литературный псевдоним — Mark Twain, либо настоящее имя писателя — Samuel Langhorne Clemens. Третий запрос возвращает книги с авторами, чье имя Charles, а фамилия не Darwin.

# Функции MySQL

Стремление применять функции MySQL при таком обилии достаточно мощных функций PHP может вызвать недоумение. А ответ предельно прост: функции MySQL работают с данными непосредственно в самой базе. А при использовании PHP приходится сначала извлекать строку данных из MySQL, выполнять обработку, а затем выдавать первоначально задуманный запрос к базе данных.

Применение встроенных функций MySQL не только существенно сокращает время обработки сложных запросов, но и упрощает сами запросы. При желании подробные сведения обо всех доступных функциях можно найти по следующим URL-адресам:

- строковые функции: http://tinyurl.com/mysqlstrfuncs;
- функции даты и времени: http://tinyurl.com/mysqldatefuncs.

Первоначальные сведения о наиболее востребованном наборе функций изложены в приложении Г.

# Работа с MySQL через phpMyAdmin

Для работы с MySQL, безусловно, важно изучить все представленные здесь основные команды и особенности их работы, но после того, как они уже изучены, для управления базами данных и таблицами будет намного проще и быстрее использовать программу phpMyAdmin.

Весь дальнейший материал основан на предположении о том, что вы уже проработали предыдущие примеры этой главы, создали в базе данных `publications` таблицы `classics` и `customers` и теперь остается только выбрать раздел, относящийся к вашей операционной системе.

## Для пользователей Windows

Перед тем как вводить в адресную строку браузера следующую строку, нужно убедиться в том, что программа Zend Server CE уже запущена и, значит, база данных MySQL готова к работе:

```
http://localhost/phpMyAdmin
```

Теперь экран вашего браузера должен приобрести вид, показанный на рис. 8.18, где нужно будет набрать имя пользователя `zend` (предлагается по умолчанию) и не вводить никакого пароля. После этого вы должны увидеть экран, похожий на рис. 8.19. Теперь вы можете приступать к изучению подраздела «Использование phpMyAdmin».

## Для пользователей Mac OS X

Убедитесь в том, что у вас запущена программа Zend Server CE и в ней запущены серверы Apache и MySQL, а затем наберите в браузере следующую строку:

```
http://localhost:10081/phpmyadmin/
```

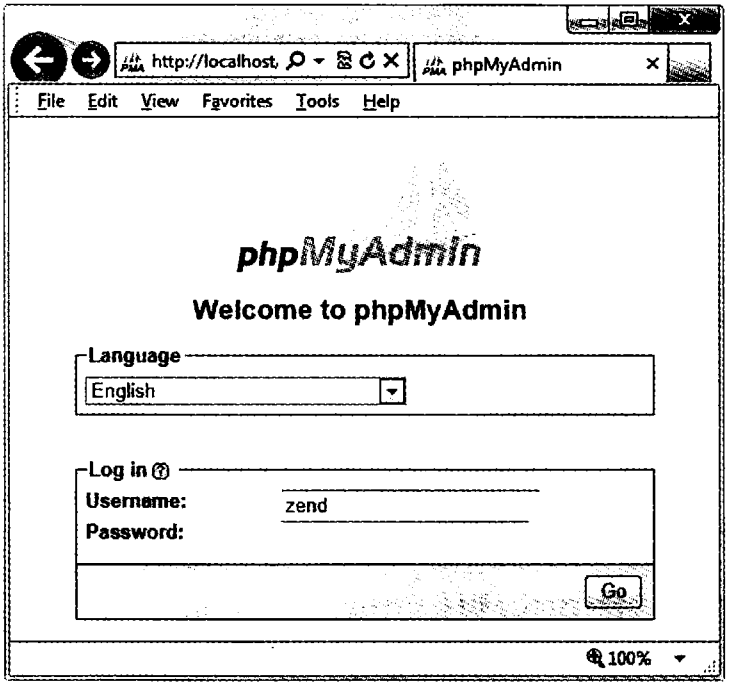



**Рис. 8.18.** Регистрация в phpMyAdmin

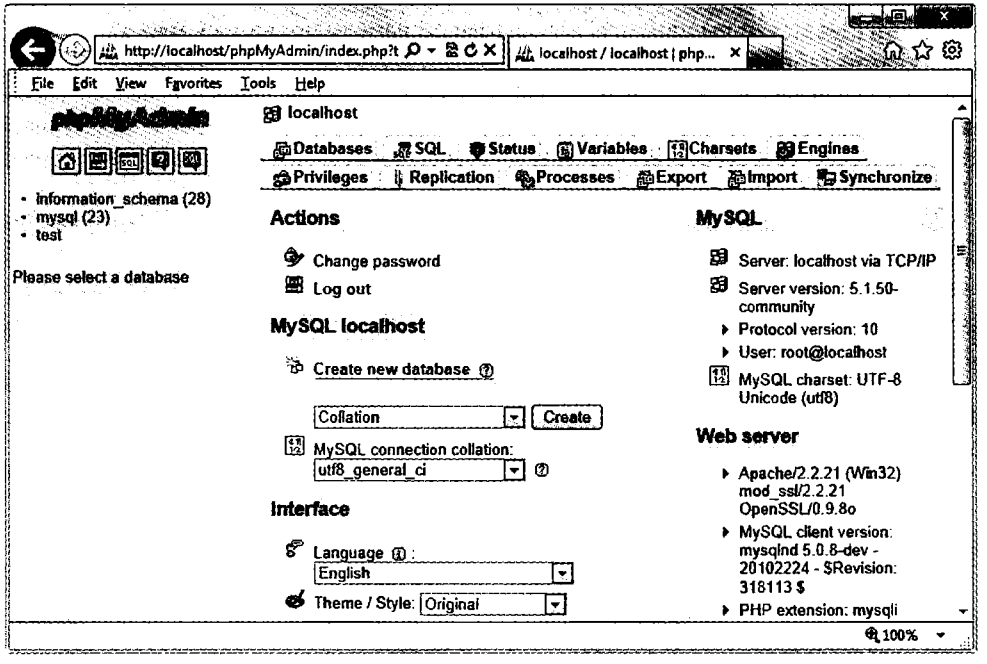



**Рис. 8.19.** Главная страница phpMyAdmin

Число 10081 обозначает интерфейсный порт сервера Zend и должно всегда вводиться для вызова интерфейса или любых подразделов, таких как phpMyAdmin.

Вы должны увидеть экран, похожий на рис. 8.18, где нужно будет набрать имя пользователя zend (предлагается по умолчанию) и не вводить никакого пароля. Ваш браузер должен приобрести вид, показанный на рис. 8.19, и теперь вы можете перейти к изучению подраздела «Использование phpMyAdmin».

## Для пользователей Linux

Если у вас установлен Zend Server CE с MySQL, для запуска phpMyAdmin можно набрать в браузере следующий адрес:

```
https://localhost:10082/phpMyAdmin
```

Теперь ваш браузер должен приобрести вид, похожий на рис. 8.18. После ввода имени пользователя zend (предлагается по умолчанию) вы должны увидеть экран, похожий на рис. 8.19. Теперь можно переходить к изучению следующего подраздела.

## Использование phpMyAdmin

На левой панели главной страницы phpMyAdmin, которая теперь должна появиться в окне вашего браузера, щелкните на раскрывающемся меню Databases (Базы данных) и выберите базу данных publications, после чего должна открыться база данных, а ниже ее названия должны появиться имена ее двух таблиц. Щелкните на имени classics, и вы увидите, что на правой панели появилась масса информации об этой таблице (рис. 8.20).

Отсюда можно осуществлять все основные операции над базами данных, включая их создание, добавление таблиц, создание индексов и многое другое. Документация по работе с программой phpMyAdmin выложена по адресу http://www.phpmyadmin.net/documentation/.

Если вы прорабатывали вместе со мной все примеры этой главы, то я вас поздравляю с окончанием этого длинного путешествия. Мы прошли большой путь создания базы данных MySQL до отправки сложных запросов, сочетающих несколько таблиц, использующих булевы операторы и усиленных различными спецификаторами MySQL.

В следующей главе мы приступим к изучению подходов к проектированию эффективных баз данных, освоим современные SQL-технологии, а также функции и транзакции MySQL.

# Проверьте ваши знания

1. Для чего нужна точка с запятой в запросах MySQL?
2. Какие команды используются для просмотра доступных баз данных или таблиц?

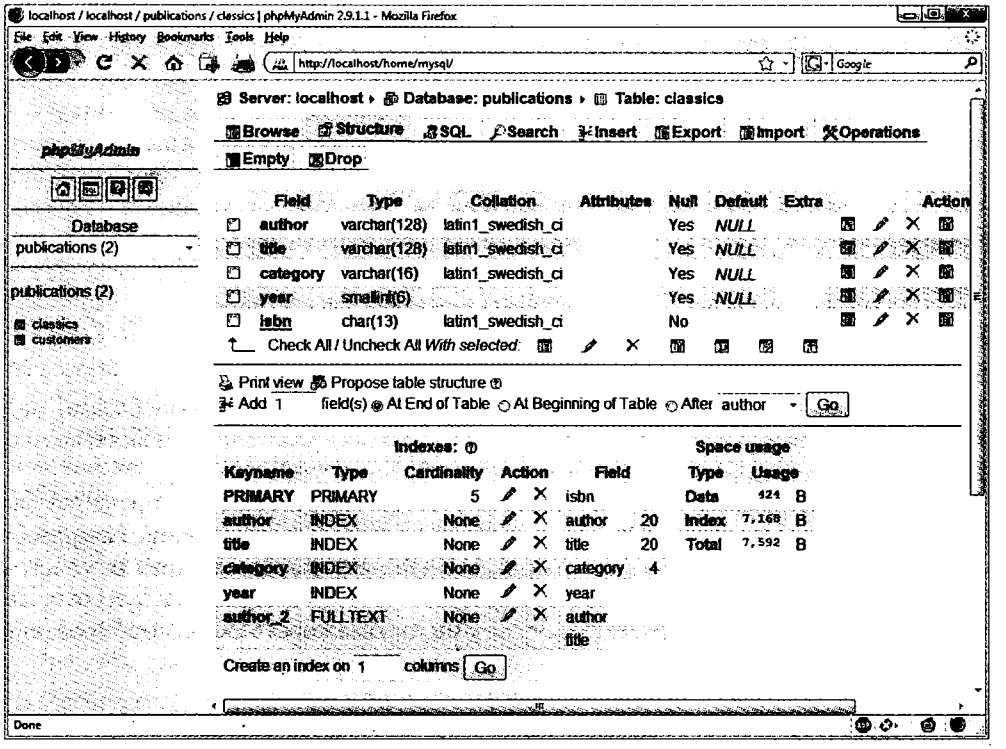



**Рис. 8.20.** Таблица classics, отображаемая в phpMyAdmin

3. Как на локальном хосте создается новый пользователь MySQL с именем `newuser` и паролем `newpass`, которому открыт доступ ко всему содержимому базы данных `newdatabase`?
4. Как просмотреть структуру таблицы?
5. Для чего нужен индекс в MySQL?
6. Какие преимущества дает индекс `FULLTEXT`?
7. Что такое стоповое слово?
8. Оба спецификатора, и `SELECT DISTINCT` и `GROUP BY`, приводят к отображению только одной строки для каждого значения в столбце, даже если такое значение имеют несколько строк. Какое основное различие между `SELECT DISTINCT` и `GROUP BY`?
9. Как можно с помощью инструкции `SELECT...WHERE` вернуть только те строки, в которых в каком-нибудь месте столбца `author` таблицы `classics`, используемой в этой главе, содержится слово `Langhorne`?
10. Что должно быть определено в двух таблицах, чтобы появилась возможность их объединения?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 8».

# 9 Освоение MySQL

В главе 8 была заложена хорошая основа для работы с реляционными базами данных с использованием SQL. Были рассмотрены создание баз данных и включаемых в них таблиц, а также вставка, поиск, изменение и удаление данных.

Теперь, вооружившись этими знаниями, нужно изучить проектирование баз данных, работать с которыми можно максимально быстро и эффективно. Например, научиться принимать решения о том, какие данные в какие таблицы помещать. За годы существования баз данных были разработаны руководства, следуя которым можно обеспечить эффективную работу с ними и возможность их масштабирования по мере наполнения все новыми и новыми данными.

## Проектирование базы данных

Перед тем как создавать базу данных, очень важно ее удачно спроектировать, в противном случае, скорее всего, придется возвращаться назад и изменять ее структуру, разбивая одни и объединяя другие таблицы и перемещая различные графы из таблицы в таблицу с целью достижения рациональных связей, которыми MySQL будет легче воспользоваться.

Для начала было бы неплохо сесть за стол с листом бумаги и карандашом и набросать подборку тех запросов, которые, на ваш взгляд, чаще всего будут нужны вам и вашим пользователям.

Для базы данных книжного интернет-магазина могут быть записаны следующие вопросы.

- Сколько авторов, книг и покупателей имеется в базе данных?
- Каким автором написана та или иная книга?
- Какие книги написаны тем или иным автором?
- Какая книга продается по самой высокой цене?
- Какая книга является лидером продаж?
- Какие книги не покупались в этом году?
- Какие книги приобретены тем или иным покупателем?
- Какие книги были приобретены вместе с какими-нибудь другими книгами?

Разумеется, к такой базе данных может быть сделано и множество других запросов, но даже эта подборка даст вам представление о том, как следует спланировать структуру таблиц.

Например, книги и номера ISBN должны быть, наверное, скомбинированы в одной таблице, поскольку они тесно взаимосвязаны (некоторые тонкости этого вопроса будут исследованы чуть позже). В отличие от этого книги и покупатели должны находиться в разных таблицах, поскольку они весьма слабо взаимосвязаны. Покупатель может купить любую книгу и даже несколько экземпляров одной и той же книги, а книга может быть приобретена многими покупателями, и может не привлечь внимания еще большего количества потенциальных покупателей.

Когда планируется множество поисковых операций по каким-нибудь столбцам, зачастую их лучше всего поместить в общую таблицу. А когда какие-то элементы слабо связаны друг с другом, их лучше поместить в отдельные таблицы.

Если принять во внимание эти элементарные правила, то можно предположить, что для удовлетворения всех этих запросов нам понадобятся как минимум три таблицы.

- `authors` (авторы). Предполагается большое количество поисков по авторам, многие из которых сотрудничали при написании книг, а значит, будут показаны вместе. Оптимальных результатов поиска можно добиться, если о каждом авторе будет дана вся относящаяся к нему информация, следовательно, нам нужна таблица авторов — `authors`.
- `books` (книги). Многие книги появляются в различных изданиях. Иногда у них разные издатели, а иногда разные книги имеют одно и то же название. Связи между книгами и авторами настолько сложны, что для книг нужна отдельная таблица.
- `customers` (покупатели). Причина, по которой покупатели должны находиться в собственной таблице, еще более прозрачна — покупатели могут приобрести любую книгу любого автора.

## Первичные ключи: ключи к реляционным базам данных

Используя возможности реляционных баз данных, мы можем задавать всю информацию для каждых автора, книги и покупателя в одном и том же месте. Очевидно, что нас интересуют связи между ними, например, кто написал каждую книгу и кто ее приобрел, и мы можем сохранить эту информацию лишь за счет создания связей между тремя таблицами. Я покажу вам основные принципы, которые нетрудно будет усвоить на практике.

Секрет заключается в присвоении каждому автору уникального идентификатора. То же самое делается для каждой книги и каждого покупателя. Смысл всего этого был объяснен в предыдущей главе: нам нужен первичный ключ. Для книги имеет смысл использовать в этом качестве номер ISBN, хотя вам, может быть, придется столкнуться с несколькими одинаковыми книгами, имеющими разные номера ISBN. Авторам и покупателям можно просто назначить произвольные

ключи, имеющие свойство автоприращения — AUTO_INCREMENT, что, судя по предыдущей главе, делается весьма просто.

Короче говоря, каждая таблица будет спроектирована вокруг какого-нибудь объекта, в котором, скорее всего, будет вестись интенсивный поиск, — в данном случае вокруг автора, книги или покупателя, и этот объект должен иметь первичный ключ. В качестве ключа не следует выбирать ничего, что могло бы иметь одинаковое значение для разных объектов. Ситуация с номером ISBN является тем самым редким случаем, когда сама издательская индустрия предоставила нам первичный ключ, который можно считать уникальным для каждого продукта. В большинстве случаев для этих целей следует создавать произвольный ключ, использующий свойство AUTO_INCREMENT.

# Нормализация

Процесс распределения данных по таблицам и создания первичных ключей называется *нормализацией*. Основная цель нормализации — обеспечить, чтобы каждая порция информации появлялась в базе данных только один раз. Дублирование данных приводит к крайне неэффективной работе, поскольку неоправданно увеличивает объем базы данных и замедляет тем самым доступ к информации. Еще важнее то, что дубликаты создают большой риск обновления только одной строки продублированных данных, приводят к несогласованности в базе данных, являющейся потенциальным источником серьезных ошибок.

Если, к примеру, названия книг перечисляются и в таблице авторов, и в таблице книг и возникает необходимость исправить опечатку в названии, нужно будет вести поиск в обеих таблицах и вносить одинаковые изменения везде, где встречаются названия книг. Лучше хранить названия в одном месте, а в других местах использовать номер ISBN.

В процессе разбиения базы данных на несколько таблиц важно не зайти слишком далеко и не создать больше таблиц, чем требуется, что может также привести к неэффективности конструкции и замедлению доступа к данным.

К счастью, изобретатель реляционной модели Эдгар Кодд проанализировал понятие нормализации и разбил его на три отдельные схемы, названные *первой*, *второй* и *третьей нормальными формами*. Если вносить изменения в базу данных, последовательно удовлетворяющие каждой из этих форм, то будет обеспечена оптимальная сбалансированность базы данных, способствующая достижению быстрого доступа и использованию минимального объема оперативной и дисковой памяти.

Чтобы понять, как выполняется нормализация, начнем с весьма несуразной базы данных, представленной в табл. 9.1, в которой имеется одна таблица, содержащая все сведения об авторах, книгах и вымышленных покупателях. Ее можно рассматривать в качестве первой попытки создания таблицы, отслеживающей, кто из покупателей какие книги заказал. Неэффективность такой конструкции не вызывает сомнений, поскольку данные повсеместно дублируются (дубликаты в таблице выделены полужирным шрифтом), но это только лишь наша отправная точка.

**Таблица 9.1.** Крайне неэффективная конструкция таблицы базы данных

| Author 1 (Автор 1) | Author 2 (Автор 2) | Title (Название) | ISBN | Price (Цена) | Cust. name (Имя покупателя) | Cust. address (Адрес покупателя) | Purch. date (Дата покупки) |
|---|---|---|---|---|---|---|---|
| *David Sklar* | *Adam Trachtenberg* | *PHP Cookbook* | *0596101015* | *44,99* | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| Danny Goodman | | Dynamic HTML | 0596527403 | 59,99 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Hugh E. Williams | David Lane | PHP and MySQL | 0596005436 | 44,95 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| *David Sklar* | *Adam Trachtenberg* | *PHP Cookbook* | *0596101015* | *44,99* | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Rasmus Lerdorf | Kevin Tatroe & Peter MacIntyre | Programming PHP | 0596006815 | 39,99 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

В следующих трех разделах мы проанализируем эту конструкцию базы данных, и вы увидите, как она может быть улучшена за счет удаления продублированных записей и разбиения одной таблицы на несколько более практичных таблиц, в каждой из которых будет храниться один тип данных.

# Первая нормальная форма

Чтобы база данных соответствовала первой нормальной форме, она должна выполнять три требования.

1. В ней не должно быть повторяющихся столбцов, содержащих одни и те же типы данных.
2. Все графы должны содержать только одно значение.
3. Для уникальной идентификации каждой строки должен быть первичный ключ.

Рассматривая по порядку эти требования, вы заметите, что в столбцы `Author 1` и `Author 2` заложены повторяющиеся типы данных. Итак, у нас уже появилась та самая графа, которую следует поместить в отдельную таблицу, поскольку повторяющаяся графа `Author` противоречит правилу 1.

Второе несоответствие связано с тем, что для последней книги, Programming PHP, указаны три автора. Я считаю, что использование одного и того же столбца `Author 2` для имен двух авторов — `Kevin Tatroe` и `Peter MacIntyre` — нарушает правило 2. Это еще одна причина перемещения всех сведений об авторах в отдельную таблицу.

А вот правило 3 здесь соблюдается, потому что первичный ключ в столбце ISBN уже создан.

В табл. 9.2 показаны результаты перемещения столбцов авторов из табл. 9.1. Теперь здесь уже меньше беспорядка, хотя все еще остаются дубликаты, выделенные полужирным шрифтом.

**Таблица 9.2.** Результаты удаления столбца Authors из табл. 9.1

| Title (Название) | ISBN | Price (Цена) | Cust. name (Имя покупателя) | Cust. Address (Адрес покупателя) | Purch. Date (Дата покупки) |
|---|---|---|---|---|---|
| *PHP Cookbook* | *0596101015* | *44,99* | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| Dynamic HTML | 0596527403 | 59,99 | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| PHP and MySQL | 0596005436 | 44,95 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| *PHP Cookbook* | *0596101015* | *44,99* | **Darren Ryder** | **4758 Emily Drive, Richmond, VA 23219** | **Dec 19 2008** |
| Programming PHP | 0596006815 | 39,99 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

Новая таблица Authors приведенная в табл. 9.3, проста по структуре и имеет довольно небольшой размер. В ней просто перечисляются номера ISBN, принадлежащие книге с тем или иным названием, рядом с которыми размещается автор. Если у книги более одного автора, соавторы получают собственную строку. Поначалу эта таблица может показаться несуразной, потому что по ней нельзя понять сразу, кто из авторов какую книгу написал. Но не стоит переживать: MySQL может быстро проинформировать вас об этом. Для этого нужно лишь сообщить, для какой именно книги нужна такая информация, и MySQL воспользуется ее ISBN для поиска в таблице авторов, что займет какие-то миллисекунды.

**Таблица 9.3.** Новая таблица Authors

| ISBN | Author (Автор) |
|---|---|
| 0596101015 | David Sklar |
| 0596101015 | Adam Trachtenberg |
| 0596527403 | Danny Goodman |
| 0596005436 | Hugh E Williams |
| 0596005436 | David Lane |
| 0596006815 | Rasmus Lerdorf |
| 0596006815 | Kevin Tatroe |
| 0596006815 | Peter MacIntyre |

Как было отмечено ранее, ISBN будет служить в качестве первичного ключа для таблицы книг — Books, когда дело дойдет до ее создания. Я упомянул об этом, чтобы подчеркнуть, что ISBN тем не менее не являет первичным ключом для таблицы Authors. При практической разработке для таблицы Authors также нужно создать первичный ключ, обеспечивающий его уникальную идентификацию.

Поэтому для таблицы Authors графа ISBN является простой графой, для которой в целях ускорения поиска может быть, наверное, создан ключ, но этот ключ будет уже не *первичным*. Фактически в этой таблице он и *не может* быть первичным, поскольку не обладает уникальностью: один и тот же номер ISBN появляется по нескольку раз в тех случаях, когда над одной книгой работали несколько авторов.

Поскольку мы будем использовать такой ключ для связи авторов с книгами в другой таблице, эта графа называется *внешним* ключом.

Ключи (которые также называются индексами) имеют в MySQL несколько предназначений. Основной целью создания ключа является ускорение поиска. В главе 8 были показаны примеры, в которых ключи использовались в условиях WHERE для осуществления поиска. Но ключ можно применять и для уникальной идентификации элемента. Таким образом, уникальный ключ часто задействуется в качестве первичного ключа в одной таблице и в качестве внешнего ключа для связи строк этой таблицы со строками другой.

## Вторая нормальная форма

Первая нормальная форма позволяет разобраться с продублированными данными (или избыточностью) в нескольких столбцах. Вторая нормальная форма имеет отношение только к решению проблемы избыточности в нескольких строках. Чтобы привести базу данных ко второй нормальной форме, ваши таблицы должны уже иметь первую нормальную форму.

Как только это будет сделано, для определения столбцов, данные в которых повторяются в разных местах, и последующего их перемещения в собственные таблицы применяется вторая нормальная форма.

Давайте еще раз посмотрим на табл. 9.2. Видите, Darren Ryder приобрел две книги, и поэтому его данные продублированы. Это говорит о том, что графы, имеющие отношение к покупателю (Customer name и Customer address), следует переместить в их собственные таблицы. В табл. 9.4 показан результат удаления двух столбцов, касающихся покупателя, из табл. 9.2.

**Таблица 9.4.** Новая таблица Titles

| ISBN | Title (Название) | Price (Цена) |
|---|---|---|
| 0596101015 | PHP Cookbook | 44,99 |
| 0596527403 | Dynamic HTML | 59,99 |
| 0596005436 | PHP and MySQL | 44,95 |
| 0596006815 | Programming PHP | 39,99 |

Таким образом, в табл. 9.4 остались только графы номера ISBN, названия (Title) и цены (Price) для четырех уникальных книг, поэтому теперь это эффективная в использовании и независимая таблица, удовлетворяющая требованиям как первой, так и второй нормальной формы. Попутно мы справились с сокращением информации до уровня тех данных, которые имеют непосредственное отношение к книгам с определенными названиями. Эта таблица может также включать год издания, количество страниц, количество переизданий и т. д., поскольку все эти данные имеют тесную связь друг с другом. Единственное правило, которого следует придерживаться, гласит: сюда нельзя помещать графы, которые могут содержать несколько значений для одной книги, поскольку тогда нам придется указывать одну и ту же книгу в нескольких строках, нарушая таким образом правила второй нормальной формы. К примеру, к нарушениям на этом этапе нормализации может привести восстановление столбца авторов.

Но, изучая извлеченные графы, относящиеся к покупателям, которые теперь показаны в табл. 9.5, можно заметить, что эта таблица все же требует дополнительной нормализации, поскольку сведения о покупателе Darren Ryder по-прежнему продублированы. Следует также признать, что правило 2 первой нормальной формы (все графы должны содержать только одно значение) здесь не соблюдается, поскольку адресные данные нужно разбить на отдельные графы для адреса — Address, города — City, штата — State и почтового индекса — Zip.

**Таблица 9.5.** Сведения о покупателях из табл. 9.2

| ISBN | Cust. Name (Имя покупателя) | Cust. Address (Адрес покупателя) | Purch. Date (Дата покупки) |
|---|---|---|---|
| 0596101015 | Emma Brown | 1565 Rainbow Road, Los Angeles, CA 90014 | Mar 03 2009 |
| 0596527403 | Darren Ryder | 4758 Emily Drive, Richmond, VA 23219 | Dec 19 2008 |
| 0596005436 | Earl B. Thurston | 862 Gregory Lane, Frankfort, KY 40601 | Jun 22 2009 |
| 0596101015 | Darren Ryder | 4758 Emily Drive, Richmond, VA 23219 | Dec 19 2008 |
| 0596006815 | David Miller | 3647 Cedar Lane, Waltham, MA 02154 | Jan 16 2009 |

Нужно продолжить разбиение этой таблицы, чтобы обеспечить однократный ввод каждого из сведений, касающихся покупателя. Поскольку ISBN не относится к таким сведениям и не может использоваться в качестве первичного ключа для идентификации покупателей (или авторов), должен быть создан новый ключ.

В табл. 9.6 показан результат нормализации таблицы Customers в соответствии с правилами первой и второй нормальных форм. Теперь у каждого покупателя есть уникальный номер покупателя, который называется CustNo, используется в качестве первичного ключа и который, скорее всего, был создан с использованием свойства автоприращения — AUTO_INCREMENT. Все составляющие адресов были также распределены по разным столбцам, для того чтобы упростить их поиск и обновление.

**Таблица 9.6.** Новая таблица Customers

| CustNo (Номер покупателя) | Name (Имя) | Address (Адрес) | City (Город) | State (Штат) | Zip (Почтовый индекс) |
|---|---|---|---|---|---|
| 1 | Emma Brown | 1565 Rainbow Road | Los Angeles | CA | 90014 |
| 2 | Darren Ryder | 4758 Emily Drive | Richmond | VA | 23219 |
| 3 | Earl B. Thurston | 862 Gregory Lane | Frankfort | KY | 40601 |
| 4 | David Miller | 3647 Cedar Lane | Waltham | MA | 02154 |

В то же время для нормализации табл. 9.6 необходимо было удалить информацию о покупках, поскольку в противном случае в ней встречались бы одни и те же сведения о покупателе для каждой купленной им книги. Вместо этого данные о покупках теперь помещены в новую таблицу Purchases (табл. 9.7).

**Таблица 9.7.** Новая таблица Purchases

| CustNo (Номер покупателя) | ISBN | Date (Дата) |
|---|---|---|
| 1 | 0596101015 | Mar 03 2009 |
| 2 | 0596527403 | Dec 19 2008 |
| 2 | 0596101015 | Dec 19 2008 |
| 3 | 0596005436 | Jun 22 2009 |
| 4 | 0596006815 | Jan 16 2009 |

Здесь в качестве ключа, связывающего вместе таблицы Customers и Purchases, опять используется графа CustNo из табл. 9.6. Поскольку здесь повторно появляется графа ISBN, эта таблица может быть связана также либо с таблицей Authors, либо с таблицей Titles.

Графа CustNo может быть полезна в качестве ключа (но только не первичного) в таблице Purchases: один и тот же покупатель может приобрести несколько книг (и даже несколько экземпляров одной и той же книги), поэтому графа CustNo не может служить первичным ключом. Фактически у таблицы Purchases вообще нет первичного ключа. И это вполне нормально, поскольку потребностей в отслеживании уникальных покупок не предвидится. Если один покупатель приобретет два экземпляра одной и той же книги, то придется смириться с двумя строками, содержащими одну и ту же информацию. Для упрощения поиска можно определить в качестве ключей, только не первичных, обе графы, CustNo и ISBN.

Теперь у нас четыре таблицы, на одну больше, чем те три, которые потребовались бы согласно нашим первоначальным прикидкам. Мы пришли к этому решению в процессе нормализации, методически следуя правилам первой и второй нормальных форм, которые однозначно позволили выявить необходимость существования дополнительной четвертой таблицы под названием Purchases (Покупки).

У нас есть следующие таблицы: Authors (см. табл. 9.3), Titles (см. табл. 9.4), Customers (см. табл. 9.6) и Purchases (см. табл. 9.7). Каждая из них может быть связана с любой другой с помощью либо ключа CustNo, либо ключа ISBN.

Например, для того, чтобы посмотреть, какие книги приобрел Darren Ryder, их можно поискать через табл. 9.6, Customers, где мы увидим, что CustNo этого покупателя — 2.

Теперь, имея этот номер, можно перейти к табл. 9.7, Purchases, найти там графу ISBN и увидеть, что он приобрел книги с номерами 0596527403 и 0596101015 19 декабря 2008 года. Подобные поиски кажутся утомительными для человека, но не составляют ни малейшего труда для MySQL.

Определить названия этих книг можно, обратившись затем к табл. 9.4, Titles, и увидев, что это книги Dynamic HTML и PHP Cookbook. Если нужно узнать авторов этих книг, следует воспользоваться номерами ISBN, которые теперь нужно найти в табл. 9.3, Authors. Станет понятно, что книгу с номером ISBN 0596527403, Dynamic HTML, написал Danny Goodman, а авторы книги с номером ISBN 0596101015, PHP Cookbook — David Sklar и Adam Trachtenberg.

## Третья нормальная форма

После приведения в соответствие первой и второй нормальным формам база данных приобрела вполне подходящий вид, и в дальнейшем вам, возможно, уже не придется что-либо в ней изменять. Но если применить к базе данных более суровые требования, то можно довести ее до соответствия правилам третьей нормальной формы, которые требуют, чтобы данные, не имеющие непосредственной зависимости от первичного ключа, но имеющие зависимость от другого значения в таблице, были также перемещены в отдельные таблицы в соответствии с тем, к чему они имеют отношение.

Например, касательно табл. 9.6, Customers, можно утверждать, что ключи State, City и Zip code не имеют прямого отношения к каждому покупателю, поскольку эти же составляющие будут присутствовать в адресах многих других людей. Но они напрямую связаны друг с другом тем, что улица в адресе — Address относится к городу — City, а город относится к штату — State.

Поэтому, чтобы соблюсти правила третьей нормальной формы для табл. 9.6, ее нужно разбить на табл. 9.8–9.11.

**Таблица 9.8.** Таблица Customers, соответствующая правилам третьей нормальной формы

| CustNo (Номер покупателя) | Name (Имя) | Address (Адрес) | Zip (Почтовый индекс) |
|---|---|---|---|
| 1 | Emma Brown | 1565 Rainbow Road | 90014 |
| 2 | Darren Ryder | 4758 Emily Drive | 23219 |
| 3 | Earl B. Thurston | 862 Gregory Lane | 40601 |
| 4 | David Miller | 3647 Cedar Lane | 02154 |

**Таблица 9.9.** Таблица Zip codes, соответствующая правилам третьей нормальной формы

| Zip (Почтовый индекс) | CityID (Идентификатор города) |
|---|---|
| 90014 | 1234 |
| 23219 | 5678 |
| 40601 | 4321 |
| 02154 | 8765 |

**Таблица 9.10.** Таблица Cities, соответствующая правилам третьей нормальной формы

| CityID (Идентификатор города) | Name (Название) | StateID (Идентификатор штата) |
|---|---|---|
| 1234 | Los Angeles | 5 |
| 5678 | Richmond | 46 |
| 4321 | Frankfort | 17 |
| 8765 | Waltham | 21 |

**Таблица 9.11.** Таблица States, соответствующая правилам третьей нормальной формы

| StateID (Идентификатор штата) | Name (Название) | Abbreviation (Аббревиатура) |
|---|---|---|
| 5 | California | CA |
| 46 | Virginia | VA |
| 17 | Kentucky | KY |
| 21 | Massachusetts | MA |

Ну и как пользоваться этим набором из четырех таблиц вместо одной таблицы 9.6? В табл. 9.8 нужно найти `Zip-код`, затем в табл. 9.9 — соответствующий ему город. Располагая этой информацией, в табл. 9.10 можно найти название города, а затем — идентификатор штата — `StateID`, который можно использовать в табл. 9.11 для поиска его названия.

Хотя подобное подстраивание под третью нормальную форму может показаться излишним, у него могут быть и свои преимущества. Например, взгляните на табл. 9.11, в которую удалось включить как название, так и двухбуквенную аббревиатуру штата. Сюда же при желании можно также включить данные о количестве жителей и другие демографические сведения.

Таблица 9.10 может также содержать более локализованную демографическую информацию, которая может оказаться полезной вам и (или) вашим покупателям. Разбивая эти данные на части, вы можете упростить обслуживание своей базы данных в будущем, когда потребуется добавить к таблицам дополнительные графы.

Решение о том, к чему именно следует применить правило третьей нормальной формы, может оказаться непростым. Оценка должна основываться на том, какие дополнительные данные могут понадобиться со временем. Если вы абсолютно уверены в том, что ничего, кроме имени и адреса покупателя, не понадобится, то, наверное, без этой заключительной стадии нормализации можно будет обойтись.

С другой стороны, представьте, что вы создаете базу данных для такой крупной организации, как Почтовая служба США. Что вы будете делать, если город будет переименован? С такой таблицей, как табл. 9.6, вам придется проводить глобальный поиск и менять название города везде, где оно упоминается. Но если ваша база данных нормализована по правилам третьей нормальной формы, нужно будет изменить всего лишь одну запись в табл. 9.10 для того, чтобы это изменение отразилось на всей базе данных.

Поэтому я советую ответить себе на два вопроса, которые помогут принять решение, нужно ли применять нормализацию по правилам третьей нормальной формы к той или иной таблице.

1. Существует ли вероятность того, что к таблице нужно будет добавить много новых граф?
2. Может ли когда-нибудь для любого из полей этих таблиц потребоваться глобальное обновление?

Если оба ответа на эти вопросы положительные, значит, наверное, вам все же следует провести заключительную стадию нормализации.

## Когда не следует проводить нормализацию

Теперь, когда вы ознакомились со всеми тонкостями нормализации, я хочу рассказать о том, почему нужно отбросить все эти правила при работе с сайтами, имеющими высокий уровень обращений. Теперь я не говорю, что вы зря потратили время на чтение последних нескольких страниц (скорее всего, это не так), но, *не следует* проводить полную нормализацию таблиц, используемых сайтом, если это приведет к излишней загруженности MySQL.

Видите ли, нормализация требует распространения данных по нескольким таблицам, а это означает, что при каждом запросе будет осуществляться несколько вызовов MySQL. Если на сайте, пользующемся высокой популярностью, будут нормализованные таблицы и счет одновременно обслуживаемых пользователей пойдет на десятки, скорость доступа к базе данных существенно снизится, потому что для их обслуживания потребуются сотни обращений к этой базе. Если серьезно, то я хочу пойти еще дальше и сказать, что вы должны провести максимально возможную *денормализацию* любых часто востребуемых данных.

Причина в том, что дублирование данных в таблицах позволяет существенно сократить количество необходимых дополнительных запросов, потому что основная масса востребованных данных доступна в каждой таблице. Это означает, что можно будет просто добавить к запросу еще одну графу, и это поле станет доступно для всех соответствующих результатов, хотя (разумеется), вам придется смириться со всеми упомянутыми ранее издержками, включая использование большого объема дискового пространства и обеспечение обновления каждой отдельной копии продублированных данных, когда одна из них требует модификации.

Конечно, многократные обновления можно компьютеризировать. Система MySQL предоставляет свойство под названием *триггеры* (triggers), которые осуществляют автоматические изменения базы данных в соответствии с произведенными вами изменениями. (Триггеры в данной книге не рассматриваются.) Другой способ копиро-

вания в среде избыточных данных состоит в настройке PHP-программы на регулярный запуск и поддержание всех копий в синхронизированном состоянии. Программа считывает изменения с «ведущей» таблицы и обновляет все остальные. (Способы доступа к MySQL из PHP будут показаны в следующей главе.)

Но пока вы не приобретете опыт работы с MySQL, я рекомендую проводить полную нормализацию всех ваших таблиц, чтобы это вошло в привычку и принесло пользу в дальнейшем. Только после этого можно приступать к выявлению «заторов» в работе MySQL и присматриваться к денормализации.

# Отношения

MySQL называют системой управления реляционными базами данных, потому что в ее таблицах содержатся не только данные, но и *отношения* между ними. Существует три категории отношений.

## «Один к одному»

Отношение «один к одному» между двумя типами данных похоже на традиционные брачные отношения: каждый элемент данных соотносится только с одним элементом другого типа. Это на удивление редкий тип отношений. Например, автор может написать несколько книг, у книги может быть несколько авторов, и даже адрес может быть связан с несколькими покупателями. Возможно, наилучшим примером, встречавшимся в этой главе, может послужить отношение «один к одному» между названием штата и его двухбуквенной аббревиатурой.

Чтобы легче было объяснить, что это такое, давайте предположим, что по какому-нибудь конкретному адресу может проживать только один покупатель. В таком случае отношение `Customers–Addresses` на рис. 9.1 будет отношением «один к одному»: только один покупатель живет по каждому адресу, и по каждому адресу может жить только один покупатель.

Обычно, когда у двух элементов имеется отношение «один к одному», их включают в качестве граф в одну и ту же таблицу. Для отнесения их к двум отдельным таблицам могут быть две причины:

- вы хотите быть готовыми к тому, что позже это отношение изменится;
- в таблице слишком много граф, и вы полагаете, что производительность работы системы или возможности ее обслуживания улучшатся за счет ее разбиения.

Разумеется, когда дело дойдет до создания вашей собственной, настоящей базы данных, между покупателями и адресами нужно будет создать отношения «один ко многим» (*один* адрес, *много* покупателей).

## «Один ко многим»

Отношения «один ко многим» (или «многие к одному») возникают в том случае, когда одна строка в одной таблице связана со многими строками в другой таблице. Вы уже поняли, что в табл. 9.8 возникли бы отношения «один ко многим», если бы

несколько покупателей проживали по одному и тому же адресу. В таком случае ее нужно разбить.

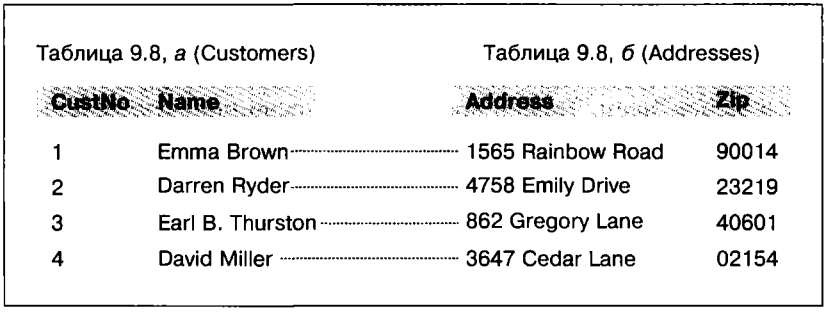


Таблица 9.8, *а* (Customers)

| CustNo | Name |
|---|---|
| 1 | Emma Brown |
| 2 | Darren Ryder |
| 3 | Earl B. Thurston |
| 4 | David Miller |

Таблица 9.8, *б* (Addresses)

| Address | Zip |
|---|---|
| 1565 Rainbow Road | 90014 |
| 4758 Emily Drive | 23219 |
| 862 Gregory Lane | 40601 |
| 3647 Cedar Lane | 02154 |

**Рис. 9.1.** Таблица покупателей, Customers (см. табл. 9.8), разбитая на две таблицы

Если посмотреть на табл. 9.8, *а*, показанную на рис. 9.1, можно увидеть, что у нее имеется отношение «один ко многим» с таблицей покупок, табл. 9.7, поскольку каждый покупатель представлен только одним конкретным человеком из табл. 9.8, *а*.

Но табл. 9.7, `Purchases`, может содержать (и содержит) более одной покупки, сделанной одним и тем же покупателем. Поэтому *один* покупатель имеет отношение *ко многим* покупкам.

На рис. 9.2 эти две таблицы показаны рядом друг с другом, а линии соединяют строки в каждой таблице и, начинаясь в одной строке левой таблицы, могут соединять с ней более одной строки правой таблицы. Схема отношения «один ко многим» также хорошо подходит и для описания отношения «многие к одному», в этом случае нужно левую и правую таблицы поменять местами и рассматривать их как отношение «один ко многим».

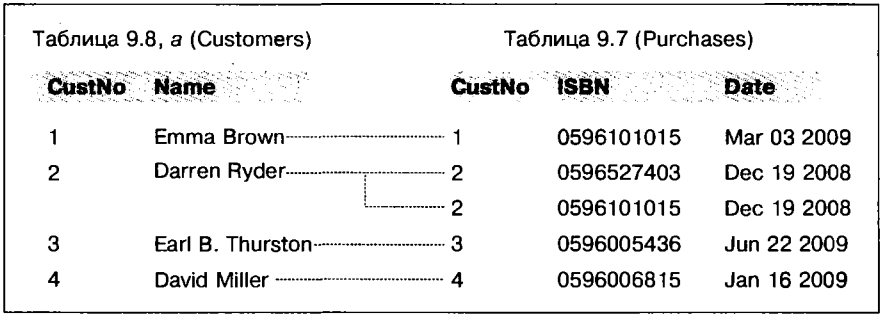


Таблица 9.8, *а* (Customers)

| CustNo | Name |
|---|---|
| 1 | Emma Brown |
| 2 | Darren Ryder |
| 3 | Earl B. Thurston |
| 4 | David Miller |

Таблица 9.7 (Purchases)

| CustNo | ISBN | Date |
|---|---|---|
| 1 | 0596101015 | Mar 03 2009 |
| 2 | 0596527403 | Dec 19 2008 |
| 2 | 0596101015 | Dec 19 2008 |
| 3 | 0596005436 | Jun 22 2009 |
| 4 | 0596006815 | Jan 16 2009 |

**Рис. 9.2.** Иллюстрация отношения между двумя таблицами

## «Многие ко многим»

В отношении «многие ко многим» многие строки в одной таблице связаны со многими строками в другой таблице. Чтобы создать это отношение, нужно добавить третью таблицу, содержащую по столбцу из каждой из этих двух таблиц, с помощью

которых они могут быть связаны. В третьей таблице больше ничего не содержится, она предназначена только для связи других таблиц.

Именно такой промежуточной таблицей и является табл. 9.12. Она была извлечена из табл. 9.7, Purchases (Покупки), но в ней отсутствует информация о дате покупки. Теперь она содержит копию номера ISBN каждой проданной книги, а также номер покупателя.

**Таблица 9.12.** Промежуточная таблица

| Customer (Покупатель) | ISBN |
|---|---|
| 1 | 0596101015 |
| 2 | 0596527403 |
| 2 | 0596101015 |
| 3 | 0596005436 |
| 4 | 0596006815 |

С помощью этой промежуточной таблицы можно пройти по всем хранящимся в базе данным, пользуясь схемой их отношений. За отправную точку можно взять адрес и найти авторов любых книг, приобретенных покупателем, проживающим по этому адресу.

Предположим, к примеру, что нужно найти покупки, связанные с почтовым индексом 23219. Если поискать этот почтовый индекс (zip code) в табл. 9.8, *б*, то можно обнаружить, что покупатель с номером 2 приобрел как минимум одну книгу, имеющуюся в базе данных. Теперь можно воспользоваться табл. 9.8, *а*, и найти имя этого покупателя или воспользоваться новой промежуточной табл. 9.12, для того чтобы найти приобретенную им книгу или книги.

По этой таблице можно определить, что были приобретены две книги, и, отследив их номера в табл. 9.4, найти названия и цены этих книг или обратиться к табл. 9.3 и увидеть в ней их авторов.

Если вам показалось, что все это, по сути, не что иное, как сочетание нескольких отношений «один ко многим», то так оно и есть. Чтобы проиллюстрировать это, на рис. 9.3 все три таблицы представлены вместе.

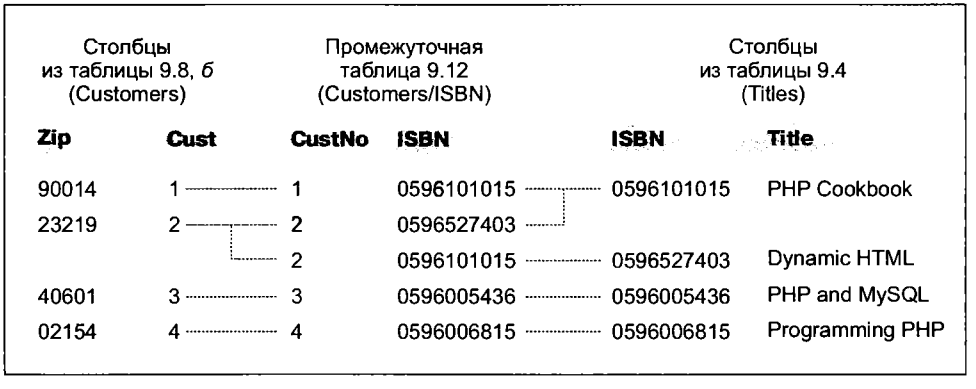



**Рис. 9.3.** Создание отношения «многие ко многим» с помощью третьей таблицы

Проследите по любому почтовому индексу (zip-коду) в левой таблице связанные с ним идентификаторы покупателей. Далее можно проследить их связь с промежуточной таблицей, которая объединяет левую и правую таблицы путем связывания покупательских идентификаторов и номеров ISBN. Теперь остается только проследовать по ISBN к правой таблице, чтобы увидеть, к какой книге он относится.

Промежуточную таблицу можно использовать также для следования в обратном направлении — от названий книг до zip-кода. Из таблицы `Titles` можно взять ISBN, которым воспользоваться для поиска в промежуточной таблице идентификационных номеров покупателей этих книг, и, наконец, в таблице `Customers` идентификационные номера будут сопоставлены с zip-кодами мест проживания покупателей.

## Базы данных и анонимность

Интересный аспект использования отношений заключается в том, что о каком-нибудь элементе, например покупателе, можно собрать массу сведений, не зная ничего о его личности. Обратите внимание на то, что в предыдущем примере мы прошли от покупательских zip-кодов к их покупкам и вернулись назад, не определяя имен покупателей. Базы данных могут использоваться для отслеживания сведений о людях, но они также могут использоваться и для защиты относящихся к ним конфиденциальных данных, при этом сохраняется возможность поиска полезной информации.

# Транзакции

В некоторых приложениях жизненно необходимо, чтобы последовательность запросов шла в нужном порядке и при этом каждый отдельный запрос успешно завершался. Представим, например, что создается последовательность запросов для перевода средств с одного банковского счета на другой. Вам бы не хотелось, чтобы при этом происходило что-либо подобное.

- Вы зачислили средства на второй счет, а когда попытались снять их с первого счета, при обновлении данных произошел сбой, и теперь эти средства числятся на обоих счетах.
- Вы сняли средства с первого банковского счета, но при запросе на обновление с целью их зачисления на второй счет произошел сбой и теперь эти средства бесследно исчезли.

Как видите, для этого типа транзакций важен не только порядок выполнения запросов, необходимо также, чтобы все части транзакции завершились успешно. Но как все это обеспечить? Ведь после осуществления запроса аннулировать его уже невозможно. Необходимо ли отслеживать все части транзакции, а затем проводить полный откат, если одна из ее частей даст сбой? Ничего этого делать не нужно, поскольку MySQL поставляется с мощным средством обработки транзакций, которое защищает именно от таких непредвиденных обстоятельств.

Кроме того, транзакции предоставляют одновременный доступ к базе данных множеству пользователей или программ за счет обеспечения очередности проведения всех транзакций, и каждый пользователь или программа соблюдают очередность, не наступая друг другу на пятки, — MySQL со всем этим прекрасно справляется.

## Ядра (механизмы хранения) транзакций

Чтобы использовать имеющееся в MySQL средство обработки транзакций, нужно воспользоваться MySQL-ядром InnoDB. Это делается довольно просто, потому что нужно всего лишь использовать другой параметр при создании таблицы. Создадим таблицу банковских счетов, введя команды, показанные в примере 9.1. (Напомню, что для этого нужно получить доступ к командной строке MySQL и воспользоваться подходящей для этой таблицы базой данных.)

**Пример 9.1.** Создание таблицы, готовой к обработке транзакций

```
CREATE TABLE accounts (
number INT, balance FLOAT, PRIMARY KEY(number)
) ENGINE InnoDB;
DESCRIBE accounts;
```

Команда, которая находится в последней строке этого примера, отобразит содержимое новой таблицы, позволяя убедиться в ее успешном создании. Будет выведена следующая информация:

```
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| number  | int(11) | NO   | PRI | 0       |       |
| balance | float   | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Теперь создадим в этой таблице две строки, которые можно будет задействовать в транзакциях. Для этого введем команды, показанные в примере 9.2.

**Пример 9.2.** Заполнение таблицы accounts

```
INSERT INTO accounts(number, balance) VALUES(12345, 1025.50);
INSERT INTO accounts(number, balance) VALUES(67890, 140.00);
SELECT * FROM accounts;
```

Команда в третьей строке отобразит содержимое таблицы, подтверждая успешное создание строк. Будет выведена следующая информация:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 |  1025.5 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

После создания и предварительного заполнения этой таблицы можно приступить к использованию транзакций.

## Использование команды BEGIN

Транзакции в MySQL начинаются либо с команды BEGIN, либо с команды START TRANSACTION. Чтобы отправить транзакцию системе MySQL, введите команды, показанные в примере 9.3.

**Пример 9.3.** Транзакция MySQL

```
BEGIN;
UPDATE accounts SET balance=balance+25.11 WHERE number=12345;
COMMIT;
SELECT * FROM accounts;
```

Результаты этой транзакции выводятся командой, содержащейся в последней строке, и должны иметь следующий вид:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 | 1050.61 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

Как видите, баланс счета 12345 увеличился на 25,11 и теперь составляет 1050,61. В примере 9.3 можно было также заметить команду COMMIT, которая рассматривается в следующем разделе.

## Использование команды COMMIT

Когда вы убедитесь в том, что ряд запросов, входящих в транзакцию, успешно выполнен, введите команду COMMIT, чтобы передать все изменения базе данных. До тех пор пока не будет получена команда COMMIT, все внесенные изменения рассматриваются MySQL как временные. Эта особенность позволяет отменить транзакцию, отправляя вместо команды передачи COMMIT команду отката ROLLBACK.

## Использование команды ROLLBACK

Используя команду ROLLBACK, можно заставить MySQL забыть обо всех запросах, выданных с начала до конца транзакции. Можете проверить эту команду в действии путем ввода транзакции по переводу средств, показанной в примере 9.4.

**Пример 9.4.** Транзакция по переводу средств

```
BEGIN;
UPDATE accounts SET balance=balance-250 WHERE number=12345;
UPDATE accounts SET balance=balance+250 WHERE number=67890;
SELECT * FROM accounts;
```

Как только будут введены эти строки, вы увидите следующий результат:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 |  800.61 |
|  67890 |     390 |
+--------+---------+
2 rows in set (0.00 sec)
```

Теперь у первого банковского счета значение на 250 единиц меньше, чем раньше, а значение второго увеличилось на 250 единиц — вы осуществили между ними перевод на 250 единиц. А теперь предположим, что что-то пошло не так и эту транзакцию нужно отменить. Для этого нужно лишь ввести команду, показанную в примере 9.5.

**Пример 9.5.** Отмена транзакции с помощью команды ROLLBACK

```
ROLLBACK;
SELECT * FROM accounts;
```

Теперь вы должны увидеть следующую выходную информацию, показывающую восстановление прежнего баланса на обоих счетах, благодаря тому что транзакция была отменена командой ROLLBACK:

```
+--------+---------+
| number | balance |
+--------+---------+
|  12345 | 1050.61 |
|  67890 |     140 |
+--------+---------+
2 rows in set (0.00 sec)
```

# Использование команды EXPLAIN

Система MySQL поставляется с мощным инструментарием, который позволяет исследовать, как она интерпретировала выданные ей запросы. Используя команду EXPLAIN, можно получить отображение состояния любого запроса, чтобы понять, можно ли его выдать более удобным или эффективным способом. Применение этой команды с созданной ранее таблицей accounts показано в примере 9.6.

**Пример 9.6.** Использование команды EXPLAIN

```
EXPLAIN SELECT * FROM accounts WHERE number='12345';
```

Результаты выполнения команды EXPLAIN будут выглядеть следующим образом:

```
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
|id|select_type|table   |type |possible_keys|key    |key_len|ref  |rows|Extra|
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
| 1|SIMPLE     |accounts|const|PRIMARY      |PRIMARY|4      |const|   1|     |
+--+-----------+--------+-----+-------------+-------+-------+-----+----+-----+
1 row in set (0.00 sec)
```

Здесь MySQL предоставляет вам следующую информацию:

- `select_type`. Тип выборки простой — `SIMPLE`. При объединении таблиц будет показан объединенный (join) тип;
- `table`. Текущей запрашиваемой таблицей была `accounts`;
- `type`. Тип запроса — `const`. Если идти от худшего к лучшему, то возможные значения выстраиваются в следующий ряд: `ALL`, `index`, `range`, `ref`, `eq_ref`, `const`, `system` и `NULL`;
- `possible_keys`. Возможно, это первичный ключ, `PRIMARY`, а это значит, что доступ должен быть быстрым;
- `key`. В данном случае используется ключ `PRIMARY`, что является хорошим показателем;
- `key_len`. Длина ключа равна 4. Это количество байтов индекса, которое будет использовано MySQL;
- `ref`. Столбец `ref` отображает, какие графы или константы используются с ключом. В данном случае используется константный ключ;
- `rows`. Количество строк, которые должны быть просмотрены этим запросом, равно 1, что также является хорошим показателем.

Когда появится запрос, который подозревается в лишней трате времени на свое выполнение, попробуйте воспользоваться командой `EXPLAIN`, чтобы посмотреть, как его можно оптимизировать. Вы сможете обнаружить, какие ключи (если таковые имеются) были задействованы, какова их длины и т. д., и тогда можно будет соответствующим образом подкорректировать запрос или конструкцию таблицы (или таблиц).

После того как эксперименты с временной таблицей `accounts` будут завершены, может появиться желание удалить эту таблицу с помощью следующей команды:

```
DROP TABLE accounts;
```

# Резервное копирование и восстановление данных

Независимо от того, какого рода данные хранятся в вашей базе данных, они все равно должны представлять для вас определенную ценность, даже если она измеряется временем, необходимым для их повторного ввода в случае повреждения жесткого диска. Поэтому для защиты вложенного вами труда важно сохранять резервные копии. Может также возникнуть потребность в перемещении вашей базы данных на новый сервер, и наилучшим способом является предварительное снятие с нее резервной копии. Важно также время от времени проверять резервные копии, для того чтобы убедиться в их целостности и работоспособности.

Создание резервных копий и восстановление данных MySQL существенно облегчается при использовании команды `mysqldump`.

# Использование команды mysqldump

Команда mysqldump позволяет выгрузить базу данных или коллекцию баз данных в один или несколько файлов, содержащих все инструкции, необходимые для воссоздания всех ваших таблиц и повторного заполнения их данными. Эта команда также может создавать файлы в формате с разделением значений запятыми — CSV (Comma-Separated Values) и в других текстовых форматах, использующих разделители, или даже в XML-формате. Главный недостаток этой команды заключается в том, что в процессе резервного копирования таблицы нужно обеспечить, чтобы никто не вел в нее запись. Эта задача решается разными способами, но самый простой заключается в остановке MySQL-сервера перед запуском mysqldump и его повторном запуске после окончания ее работы.

Можно также перед запуском команды mysqldump заблокировать все копируемые таблицы. Для блокировки чтения таблиц (поскольку нам нужно считать данные) в командную строку MySQL нужно ввести следующую команду:

```
LOCK TABLES имя_таблицы1 READ, имя_таблицы2 READ ...
```

А для снятия блокировки нужно ввести следующую команду:

```
UNLOCK TABLES;
```

По умолчанию вся выходная информация выводится командой mysqldump на стандартное устройство, но ее можно перенаправить в файл, воспользовавшись символом >.

Стандартный формат mysqldump имеет следующий вид:

```
mysqldump -u пользователь -pпароль база_данных
```

Но перед тем, как выгружать содержимое базы данных, следует убедиться в том, что путь к программе mysqldump может быть найден по умолчанию или что ее размещение указано в самой команде. В табл. 9.13 показаны наиболее вероятные места нахождения этой программы для различных установок и операционных систем, рассмотренных в главе 2. Если у вас какой-нибудь другой вариант установки, ее местонахождение может быть несколько иным.

**Таблица 9.13.** Наиболее вероятные места нахождения программы mysqldump для различных установок

| Операционная система и программа | Наиболее вероятная папка местонахождения |
|---|---|
| ZendServerCE для 32-разрядной версии Windows | C:\Program Files\zend\MySQL51\bin |
| ZendServerCE для 64-разрядной версии Windows | C:\Program Files(x86)\zend\MySQL51\bin |
| OS X Zend Server CE | /usr/local/zend/mysql/bin |
| Linux Zend Server CE | /usr/local/zend/mysql/bin |

Для вывода содержимого базы данных publications, созданной в главе 8, на экран наберите mysqldump (или при необходимости укажите полный путь) и команду, показанную в примере 9.7.

**Пример 9.7.** Вывод базы данных publications на экран

```
mysqldump -u пользователь -pпароль publications
```

Вместо слов пользователь и пароль подставьте имя пользователя и пароль, которые используются в вашей установке MySQL. Если пароль для пользователя не установлен, эту часть команды можно опустить, но часть команды -u пользователь является обязательной, если только у вас не установлен привилегированный доступ (root) без пароля и вы не работаете в этом режиме (что делать не рекомендуется). Результат ввода этой команды будет похож на тот, что изображен на рис. 9.4.

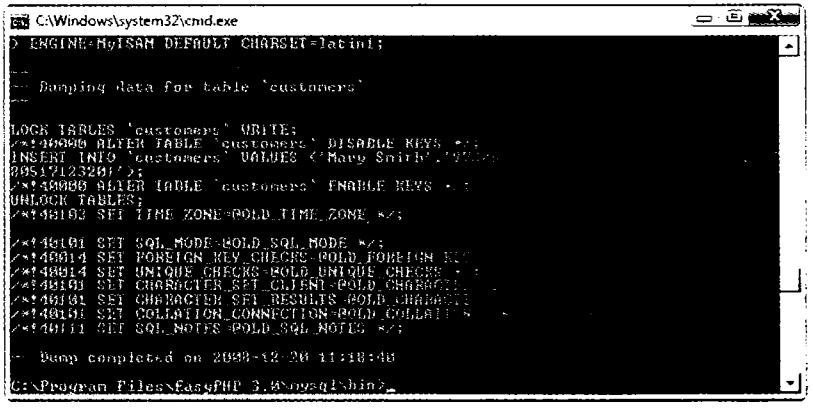



**Рис. 9.4.** Выгрузка базы данных publications на экран

## Создание файла резервной копии

Запустив команду mysqldump и убедившись в том, что она выводит на экран нужные данные, можно перенаправить данные резервной копии непосредственно в файл, используя символ >.

Если предположить, что вам захотелось назвать файл резервной копии publications.sql, нужно ввести команду, показанную в примере 9.8 (не забудьте подставить вместо слов пользователь и пароль настоящее имя пользователя и пароль).

**Пример 9.8.** Выгрузка базы данных publications в файл

```
mysqldump -u пользователь -pпароль publications > publications.sql
```

Команда в примере 9.8 сохраняет файл резервной копии в текущем каталоге. Если нужно сохранить его в каком-нибудь другом месте, то перед именем файла следует указать соответствующий путь. Кроме того, необходимо убедиться в том, что каталог, куда будет сохраняться файл резервной копии, имеет соответствующие установки доступности, позволяющие записывать в него этот файл.

При выводе файла резервной копии на экран или загрузке его в текстовый редактор вы увидите, что он состоит из последовательности SQL-команд:

```
DROP TABLE IF EXISTS 'classics';
CREATE TABLE 'classics' (
  'author' varchar(128) default NULL,
  'title' varchar(128) default NULL,
  'category' varchar(16) default NULL,
  'year' smallint(6) default NULL,
  'isbn' char(13) NOT NULL default '',
  PRIMARY KEY  ('isbn'),
  KEY 'author' ('author' (20)),
  KEY 'title' ('title' (20)),
  KEY 'category' ('category' (4)),
  KEY 'year' ('year'),
  FULLTEXT KEY 'author_2' ('author','title')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Это весьма продуманный код, который может быть использован для восстановления базы данных из резервной копии, даже если она уже существует, поскольку сначала он удалит все таблицы, которые должны быть воссозданы, избавляясь таким образом от потенциальных ошибок MySQL.

## Создание резервной копии отдельной таблицы

Чтобы создать резервную копию отдельной таблицы базы данных (такой как таблица `classics` базы данных `publications`), сначала нужно из командной строки MySQL заблокировать таблицу, набрав следующую команду:

```
LOCK TABLES publications.classics READ;
```

Это обеспечит работу MySQL в режиме чтения, но сделает невозможной запись. Затем, не закрывая командную строку MySQL, используйте другое окно терминала, чтобы ввести из командной строки операционной системы следующую команду:

```
mysqldump -u пользователь -pпароль publications classics > classics.sql
```

Теперь можно снять блокировку таблицы, для чего в первом окне терминала в командной строке MySQL нужно ввести следующую команду, которая разблокирует все таблицы, заблокированные в текущем сеансе:

```
UNLOCK TABLES;
```

## Создание резервной копии всех таблиц

Если понадобится создать резервную копию сразу всех ваших баз данных MySQL (включая и такие системные базы данных, как `mysql`), можно воспользоваться командой, показанной в примере 9.9, которая позволит восстановить всю установку базы данных MySQL, но при этом следует не забыть про блокировку там, где она потребуется.

**Пример 9.9.** Выгрузка всех баз данных MySQL в файл

```
mysqldump -u пользователь -pпароль --all-databases > all_databases.sql
```

Разумеется, в файлах резервных копий баз данных MySQL содержится очень много строк SQL-кода. Я советую потратить несколько минут на изучение ряда этих строк для ознакомления с типами команд, которые встречаются в файлах резервных копий, и с порядком их работы.

## Восстановление данных из файла резервной копии

Чтобы восстановить данные из файла, нужно вызвать исполняемую программу mysql и передать ей файл, из которого восстанавливаются данные, для чего следует воспользоваться символом <. Для восстановления всей базы данных, выгруженной с помощью ключа --all-databases, используется команда, показанная в примере 9.10.

**Пример 9.10.** Восстановление полного набора баз данных

```
mysql -u пользователь -рпароль < all_databases.sql
```

Для восстановления одной базы данных применяется ключ -D, за которым следует имя базы данных. В примере 9.11 показано, как восстановить базу данных publications из резервной копии, созданной кодом, который показан в примере 9.8.

**Пример 9.11.** Восстановление базы данных publications

```
mysql -u пользователь -рпароль -D publications < publications.sql
```

Для восстановления отдельной таблицы базы данных используется команда, показанная в примере 9.12, где в базе данных publications восстанавливается только таблица classics.

**Пример 9.12.** Восстановление таблицы classics в базе данных publications

```
mysql -u пользователь -рпароль -D publications < classics.sql
```

## Выгрузка данных в файлы формата CSV

Как уже отмечалось, программа mysqldump обладает завидной гибкостью и поддерживает различные типы выходных данных, в том числе формат CSV. В примере 9.13 показано, как можно выгрузить данные из таблиц classics и customers базы данных publications в файлы classics.txt и customers.txt, находящиеся в папке c:/temp. По умолчанию при работе в среде Zend Server CE пользователь должен быть под именем root и не применять пароль. Если работа идет в операционной системе OS X или Linux, следует изменить путь назначения на существующую папку.

**Пример 9.13.** Выгрузка данных в файлы формата CSV

```
mysqldump -u пользователь -рпароль --no-create-info --tab=c:/temp
  --fields-terminated-by=',' publications
```

Команда слишком длинная, и в этом примере она занимает две строки, но вводить ее нужно в одной строке. В результате работы команды будет выведен следующий текст:

```
Mark Twain (Samuel Langhorne Clemens)','The Adventures
 of Tom Sawyer','Classic Fiction','1876','9781598184891
Jane Austen','Pride and Prejudice','Classic Fiction','1811','9780582506206
Charles Darwin','The Origin of Species','Non-Fiction','1856','9780517123201
Charles Dickens','The Old Curiosity Shop','Classic Fiction','1841','9780099533474
```

```
William Shakespeare','Romeo and Juliet','Play','1594','9780192814968

Mary Smith','9780582506206
Jack Wilson','9780517123201
```

## Планирование резервного копирования

Золотое правило резервного копирования гласит, что его следует проводить с той периодичностью, которая имеет практический смысл. Чем ценнее данные, тем чаще следует создавать их резервные копии и тем больше резервных копий нужно делать. Если ваша база данных обновляется хотя бы раз в сутки, то резервное копирование нужно проводить ежедневно. Если же она не подвергается частым обновлениям, то резервные копии можно создавать значительно реже.

Нужно также подумать о создании нескольких резервных копий и о хранении их в разных местах. Если у вас используются несколько серверов, то можно просто растиражировать резервные копии по этим серверам. Можно также прислушаться к хорошему совету, заключающемуся в том, что следует создавать физические резервные копии съемных жестких дисков, миниатюрных носителей, компакт-дисков или DVD и т. д. и хранить их в разных местах, предпочтительно в чем-то вроде сейфов.

После изучения материалов этой главы вы должны стать специалистом по работе как с PHP, так и с MySQL. В следующей главе будет показано, как можно объединить эти две технологии.

# Проверьте ваши знания

1. Что означает слово *отношение* (relationship) применительно к реляционным базам данных?
2. Какое понятие применяется к процессу удаления повторяющихся данных и оптимизации таблиц?
3. Как формулируются три правила первой нормальной формы?
4. Как привести таблицу в соответствие с правилом второй нормальной формы?
5. Что нужно поместить в графу, для того чтобы связать две таблицы, содержащие элементы, имеющие отношение «один ко многим»?
6. Как создать базу данных с отношением «многие ко многим»?
7. Какие команды инициируют и завершают транзакцию MySQL?
8. Какие возможности предоставляет MySQL для изучения подробностей работы запроса?
9. Какую команду нужно использовать для создания резервной копии базы данных `publications` в файле `publications.sql`?

Ответы на эти вопросы можно найти в приложении A, в разделе «Ответы на вопросы главы 9».

# 10 Доступ к MySQL с использованием PHP

При полноценном изучении предыдущих глав вы должны были приобрести навыки работы как с MySQL, так и с PHP. В этой главе вы узнаете, как объединить эти два компонента путем использования встроенных в PHP функций доступа к MySQL.

## Запросы к базе данных MySQL с помощью PHP

Смысл использования PHP в качестве интерфейса к MySQL заключается в форматировании результатов SQL-запросов и придании им внешнего вида, предназначенного для вывода на веб-страницу. Обладая возможностью входа в установленную систему MySQL с помощью своего имени пользователя и пароля, вы можете сделать то же самое и из PHP. Но вместо использования командной строки MySQL для ввода команд и просмотра выходной информации нужно будет создать строки запроса, а затем передать их MySQL. Ответ MySQL поступит в виде структуры данных, которую PHP сможет распознать, а не в виде того отформатированного экранного вывода, который вы видели в процессе работы с командной строкой. Затем с помощью команд PHP можно будет извлекать данные и приводить их к формату веб-страницы.

## Процесс

Процесс использования MySQL с помощью PHP заключается в следующем.

1. Подключение к MySQL.
2. Выбор базы данных, которая будет использоваться.
3. Создание строки запроса.
4. Выполнение запроса.
5. Извлечение результатов и вывод их на веб-страницу.
6. Повторение шагов с 3-го по 5-й до тех пор, пока не будут извлечены все необходимые данные.
7. Отключение от MySQL.

Далее процесс будет рассмотрен поэтапно, но сначала важно настроить все элементы входа в систему на безопасную работу, для того чтобы шпионы, заинтересовавшиеся вашей системой, натыкались на заслон при попытке получения доступа к вашей базе данных.

## Создание файла регистрации

Большинство веб-сайтов, разработанных на PHP, содержат множество программных файлов, которым понадобится доступ к MySQL, и им нужны будут сведения, касающиеся входа в систему и пароля. Поэтому имеет смысл создать отдельный файл для их хранения, а затем включать его туда, где он необходим. Такой файл, который я назвал `login.php`, показан в примере 10.1. Наберите текст этого примера, заменяя значения `имя_пользователя` и `пароль` теми, которыми вы пользуетесь для входа в свою базу данных MySQL, и сохраните текст в файле, поместив его в разработочный каталог, созданный согласно рекомендациям, сделанным в главе 2. Через некоторое время этот файл нам пригодится. Имя хоста `localhost` будет работать до тех пор, пока вы используете базу данных MySQL на своей локальной системе, точно так же будет работать и база данных `publications`, если вы вводили в компьютер код всех встречавшихся ранее примеров.

**Пример 10.1.** Файл `login.php`

```
<?php // login.php
$db_hostname = 'localhost';
$db_database = 'publications';
$db_username = 'имя_пользователя';
$db_password = 'пароль';
?>
```

Для файла `login.php`, показанного в примере 10.1, особую роль играют охватывающие теги `<?php` и `?>`, поскольку они дают понять, что все строки, находящиеся между ними, должны интерпретироваться только как код PHP. Если их не поставить, то при вызове файла непосредственно с вашего веб-сайта он будет отображен в виде текста, раскрывая все ваши секреты. А когда теги на месте, на сайте будет видна пустая страница. Этот файл будет без каких-либо проблем включаться в другие ваши PHP-файлы.

Переменная `$db_hostname` сообщит PHP, какой компьютер следует использовать при подключении к базе данных. Ее присутствие обусловлено тем, что вы можете получить доступ к любой базе данных MySQL на любом компьютере, подключенном к той машине, на которой вы установили PHP, и она может потенциально включать в себя любой хост на просторах Всемирной паутины. Но примеры, приводимые в данной главе, будут работать только на локальном сервере. Поэтому здесь будет указываться не домен вроде **mysql.myserver.com**, а может просто использоваться слово `localhost` (или IP-адрес 127.0.0.1).

В роли рабочей базы данных, `$db_database`, будет выступать база данных `publications`, которую вы, наверное, уже создали, изучая главу 8, или одна из тех баз данных, которую вам предоставил администратор вашего сервера (в таком случае нужно будет также внести соответствующие изменения в файл `login.php`).

Переменным $db_username и $db_password нужно присвоить значения пользовательского имени и пароля, которые используются при работе с MySQL.

Другим преимуществом хранения всех сведений, необходимых для входа в систему, в одном месте станет возможность изменения пароля с нужной вам периодичностью, для чего придется обновлять только один файл, независимо от количества PHP-файлов, получающих доступ к MySQL.

## Подключение к MySQL

После сохранения файла login.php можно будет с помощью инструкции require_once включать его в любые PHP-файлы, которым нужен доступ к базе данных. Выбор пал именно на эту инструкцию, а не на инструкцию include, поскольку, если файл не будет найден, он сгенерирует фатальную ошибку. И уж поверьте мне, если не будет найден файл, содержащий сведения для подключения к вашей базе данных, это действительно будет фатальной ошибкой.

А использование require_once, а не require означает, что файл будет считан только в том случае, если он не был включен до этого в какой-нибудь другой файл, что исключит совершенно бесполезные повторные обращения к диску. Код, используемый для подключения, показан в примере 10.2.

**Пример 10.2.** Подключение к базе данных MySQL

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);

if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
?>
```

В этом примере запускается PHP-функция mysql_connect, которой нужны три параметра: *имя хоста* (hostname) MySQL-сервера, *имя пользователя* (username) и *пароль* (password). В случае успешного подключения эта функция возвращает *идентификатор* сервера, а в случае неудачи — значение FALSE. Обратите внимание на то, что инструкция if во второй строке используется вместе с функцией die, работа которой созвучна ее имени (прекратить) и заключается в выходе из PHP с сообщением об ошибке, если переменная $db_server не имеет истинного значения.

В сообщении, передаваемом функции die, объясняется, что подключиться к базе данных MySQL не представилось возможным, и чтобы помочь выяснить причины произошедшего, здесь содержится вызов функции mysql_error. Эта функция выводит текст объяснения ошибки, возникшей при вызове последней функции MySQL.

Сервер базы данных, указанный в переменной $db_server, будет задействован в ряде следующих примеров для идентификации запрашиваемого сервера MySQL. При таком использовании идентификаторов появляется возможность подключения и доступа к нескольким серверам MySQL из одной PHP-программы.

Функция die хорошо подходит для разработки кода PHP, но на рабочем сервере, вам, конечно же, захочется использовать более вразумительные сообщения об ошибках. В этом случае нужно будет не выходить из программы PHP в аварийном режиме, а составить сообщение, которое будет отображено при нормальном выходе из программы, например:

```
function mysql_fatal_error($msg)
{
    $msg2 - mysql_error();
    echo <<< _END
К сожалению, завершить запрашиваемую задачу не представилось возможным.
Было получено следующее сообщение об ошибке:

    <p>$msg: $msg2</p>

Пожалуйста, щелкните на кнопке возврата вашего браузера
и повторите попытку. Если проблемы не прекратятся,
пожалуйста, <a href="mailto:admin@server.com">сообщите о них
нашему администратору </a>. Спасибо.
_END;
}
```

## Выбор базы данных

После успешного подключения к MySQL появляется возможность выбрать базу данных, с которой нужно будет работать. Как это сделать, показано в примере 10.3.

**Пример 10.3.** Выбор базы данных

```
<?php
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());
?>
```

Для выбора базы данных используется команда `mysql_select_db`. Ей нужно передать имя требуемой базы данных и сервер, к которому вы подключены. Как и в предыдущем примере для вывода сообщения об ошибке и объяснения ее причины в случае сбоя при выборе базы данных в код включена функция `die`. Единственное отличие состоит в том, что в этом примере не нужно запоминать значение, возвращаемое функцией `mysql_select_db`, поскольку она возвращает либо `TRUE`, либо `FALSE`. Поэтому был использован PHP-оператор `or`, означающий следующее: «Если при выполнении предыдущей команды произойдет сбой, нужно выполнить тот код, который следует за этим оператором». Учтите, что для срабатывания оператора `or` в конце первой строки кода не должно быть точки с запятой.

## Создание и выполнение запроса

Оправка запроса к MySQL из PHP сводится к простому вызову функции `mysql_query`. Порядок ее использования показан в примере 10.4.

**Пример 10.4.** Отправка запроса к базе данных

```
<?php
$query = "SELECT * FROM classics";
```

```
$result = mysql_query($query);

if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Сначала переменной `$query` присваивается значение, содержащее код предстоящего запроса. В данном случае запрашивается просмотр всех строк таблицы `classics`. Учтите, что, в отличие от командной строки MySQL, здесь в конце запроса точка с запятой не нужна, поскольку для завершения запроса используется функция `mysql_query`, которая не может применяться для запроса, состоящего из последовательно отправляемых частей. Поэтому MySQL знает, что запрос завершен, и не ищет точки с запятой.

Функция возвращает результат, помещаемый в переменную `$result`. После использования MySQL в режиме командной строки можно подумать, что содержимое `$result` будет таким же, как тот результат, который возвращался запросом, вводимым в командной строке, с горизонтальными и вертикальными линиями и т. д. Но в PHP возвращается не такой результат. В случае успешного выполнения запроса переменная `$result` будет содержать *ресурс*, позволяющий извлечь результаты этого запроса. Извлечение данных будет показано в следующем разделе. При сбое переменная `$result` содержит значение `FALSE`. Поэтому пример завершается проверкой значения этой переменной. Если оно равно `FALSE`, значит, произошла ошибка и должна быть выполнена функция `die`.

## Извлечение результата

Ресурс, возвращенный функцией `mysql_query`, можно использовать для извлечения требуемых данных. Наиболее простой способ заключается в последовательном извлечении нужных ячеек с помощью функции `mysql_result`. Пример 10.5 объединяет и расширяет предыдущие примеры до программы, которую можно набрать и самостоятельно запустить для получения возвращенных результатов. Я советую сохранить это программу в той же папке, в которой был сохранен файл `login.php`, и присвоить файлу этой программы имя `query.php`.

**Пример 10.5.** Поячеечное извлечение результатов

```
<?php // query.php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);

if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());

mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "SELECT * FROM classics";
$result = mysql_query($query);

if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());

$rows = mysql_num_rows($result);

for ($j = 0 ; $j < $rows ; ++$j)
```

```
{
    echo 'Author: '    . mysql_result($result,$j,'author')     '<br />';
    echo 'Title: '     . mysql_result($result,$j,'title')      '<br />';
    echo 'Category: ' . mysql_result($result,$j,'category') . '<br />';
    echo 'Year: '      . mysql_result($result,$j,'year')       '<br />';
    echo 'ISBN: '      . mysql_result($result,$j,'isbn')       '<br /><br />';
}
?>
```

Здесь новыми для нас являются десять последних строк кода, поэтому их и рассмотрим. Они начинаются с присвоения переменной `$rows` значения, возвращенного в результате вызова функции `mysql_num_rows`. Эта функция сообщает о количестве строк, возвращенных запросом.

Получив счетчик строк, мы входим в цикл `for`, который извлекает каждую ячейку данных из каждой строки с помощью функции `mysql_result`. В качестве параметров этой функции используются ресурс `$result`, возвращенный функцией `mysql_query`, номер строки `$j` и имя графы, из которой следует извлечь данные.

Затем результаты, получаемые при каждом вызове функции `mysql_result`, включаются в инструкции `echo` для отображения по одному полю на каждой строке с дополнительным символом перевода строки между строками. Результат запуска этой программы показан на рис. 10.1.

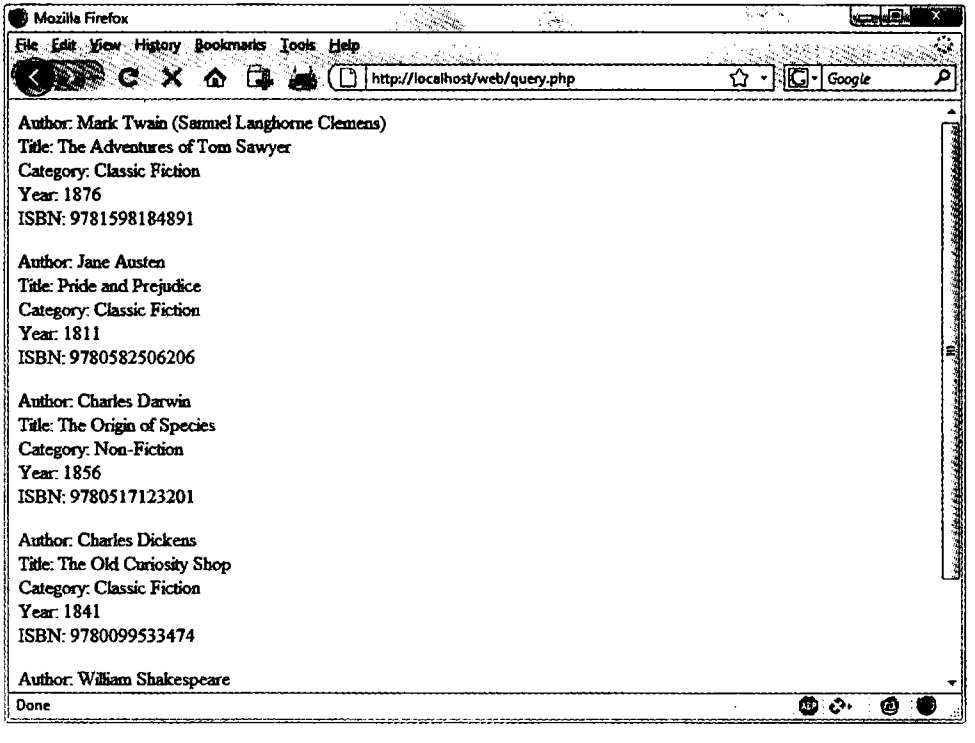



**Рис. 10.1.** Данные, выводимые программой query.php, представленной в примере 10.5

Вы, наверное, помните, что мы в главе 8 заполнили таблицу `classics` пятью строками, и, конечно же, программа `query.php` возвращает пять строк данных. Но в данном виде этот код выполняется крайне неэффективно и медленно, потому что для поячеечного извлечения всех данных в общей сложности осуществляется 25 вызовов функции `mysql_result`. Существует гораздо более удачный способ извлечения данных, позволяющий при каждом вызове функции `mysql_fetch_row` получать отдельную строку таблицы.

В главе 9 шла речь о первой, второй и третьей нормальных формах, а теперь можно заметить, что таблица classics не удовлетворяет правилам этих форм, потому что сведения как об авторах, так и о книгах включены в одну и ту же таблицу. Причина в том, что эта таблица была создана еще до того, как мы приступили к изучению нормализации. Но ее повторное использование в целях иллюстрации доступа к MySQL из PHP избавляет нас от необходимости ввода нового набора тестовых данных, поэтому в данном случае мы продолжим работу с этой таблицей.

## Извлечение строки

Мне было важно показать способ извлечения из базы данных MySQL отдельно взятой ячейки, но гораздо эффективнее извлекать всю строку. Поэтому замените цикл `for` в программе `query.php` (см. пример 10.5) новым циклом, показанным в примере 10.6, и увидите, что будет получен абсолютно такой же результат, как показанный на рис. 10.1.

**Пример 10.6.** Замена цикла for для построчного извлечения результатов

```
<?php
for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = mysql_fetch_row($result);
    echo 'Author: '      $row[0] . '<br />';
    echo 'Title: ' .     $row[1] . '<br />';
    echo 'Category: '    $row[2] . '<br />';
    echo 'Year: '        $row[3] . '<br />';
    echo 'ISBN: '        $row[4] . '<br /><br />';
}
?>
```

В этом измененном коде функция, обращающаяся к MySQL, вызывается в пять раз (на 80 %) реже, поскольку функция `mysql_fetch_row` позволяет извлекать каждую строку таблицы целиком. Эта функция возвращает одну строку данных в массиве, который затем присваивается переменой `$row`.

После извлечения данных останется только обратиться по очереди к каждому элементу массива `$row` (начиная с нулевого). Получается, что `$row[0]` содержит данные об авторе, `$row[1]` — о названии книги и т. д., потому что каждая графа помещена в массив в том порядке, в котором она появляется в таблице MySQL. Также использование вместо `mysql_result` функции `mysql_fetch_row` значительно сокращает объем кода PHP и время его выполнения, поскольку здесь применяется простая ссылка на каждый элемент данных по смещению в массиве, а не имя этого элемента.

## Отключение

По окончании работы с базой данных от нее следует отключиться. Это делается с помощью команды, показанной в примере 10.7.

**Пример 10.7.** Отключение от базы данных MySQL

```
<?php
mysql_close($db_server);
?>
```

Здесь функции нужно передать тот идентификатор, который был возвращен функцией `mysql_connect` из примера 10.2 и сохранен в переменной `$db_server`.

Все подключения к базам данных автоматически закрываются по выходе из PHP, поэтому то, что подключение в примере 10.5 не было закрыто, не имеет никакого значения. Но в более длинных программах, где могут последовательно осуществляться подключения к базам данных и отключения от них, настоятельно рекомендуется по завершении доступа закрывать каждое подключение к базе данных.

# Практический пример

Теперь настало время создать наш первый пример использования PHP для вставки данных в таблицу MySQL и удаления их оттуда. Я рекомендую набрать пример 10.8 и сохранить его в вашем разработочном каталоге в файле под именем `sqltest.php`. В результате работы кода из этого примера экран приобретает вид, показанный на рис. 10.2.

В примере 10.8 создается стандартная HTML-форма. Более подробно такие формы рассматриваются в следующей главе, а здесь обработка формы используется только для демонстрации взаимодействия с базой данных.

**Пример 10.8.** Вставка и удаление данных с помощью программы sqltest.php

```
<?php // sqltest.php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);

if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());

mysql_select_db($db_database, $db_server)
    or die("Невозможно выбрать базу данных: " . mysql_error());

if (isset($_POST['delete']) && isset($_POST['isbn']))
{
    $isbn  = get_post('isbn');
    $query = "DELETE FROM classics WHERE isbn='$isbn'";

    if (!mysql_query($query, $db_server))
```

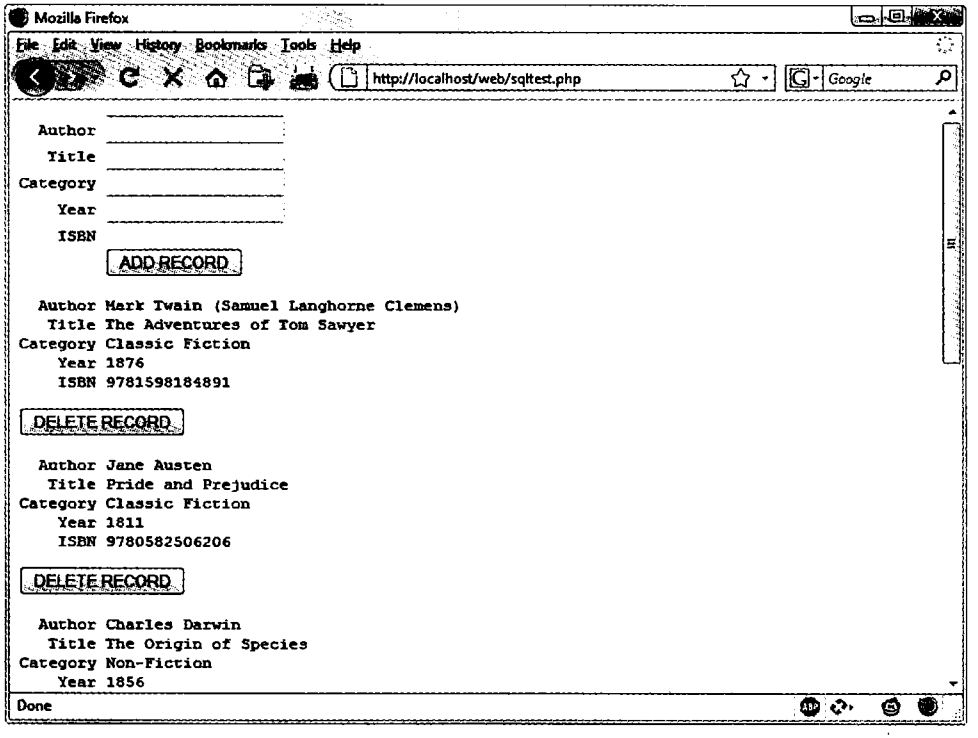



**Рис. 10.2.** Вид экрана, получаемый в результате работы кода из примера 10.8, сохраненного в файле sqltest.php

```
            echo "Сбой при удалении данных: $query<br />"
            mysql_error() . "<br /><br />";
}

if (isset($_POST['author']) &&
    isset($_POST['title']) &&
    isset($_POST['category']) &&
    isset($_POST['year']) &&
    isset($_POST['isbn']))
{
    $author   = get_post('author');
    $title    = get_post('title');
    $category = get_post('category');
    $year     = get_post('year');
    $isbn     = get_post('isbn');

    $query = "INSERT INTO classics VALUES" .
    "('$author', '$title', '$category', '$year', '$isbn')";

    if (!mysql_query($query, $db_server))
        echo "Сбой при вставке данных: $query<br />" .
        mysql_error() . "<br /><br />";
```

```
}

echo <<<_END
<form action="sqltest.php" method="post"><pre>
  Author <input type="text" name="author" />
   Title <input type="text" name="title" />
Category <input type="text" name="category" />
    Year <input type="text" name="year" />
    ISBN <input type="text" name="isbn" />
         <input type="submit" value="ADD RECORD" /> // кнопка
                                                    // ДОБАВИТЬ ЗАПИСЬ
</pre></form>
_END;

$query = "SELECT * FROM classics";
$result = mysql_query($query);

if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
$rows = mysql_num_rows($result);

for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = mysql_fetch_row($result);
    echo <<<_END
<pre>
  Author $row[0]
   Title $row[1]
Category $row[2]
    Year $row[3]
    ISBN $row[4]
</pre>
<form action="sqltest.php" method="post">
<input type="hidden" name="delete" value="yes" />
<input type="hidden" name="isbn" value="$row[4]" />
<input type="submit" value="DELETE RECORD" /></form>  // кнопка
                                                      // УДАЛИТЬ ЗАПИСЬ
_END;
}

mysql_close($db_server);

function get_post($var)
{
    return mysql_real_escape_string($_POST[$var]);
}
?>
```

Эта программа со своими более чем 80 строками кода может показаться пугающей, но не стоит переживать — многие из этих строк уже были изучены в примере 10.5, а в работе этого кода нет ничего сложного.

Сначала производится проверка всех введенных данных, а затем в соответствии с предоставленным вводом осуществляется либо вставка новых данных в таблицу classics базы данных publications, либо удаление строки из этой таблицы. Независимо от того, что именно было введено, программа вслед за этим выводит все строки таблицы в браузер. Давайте посмотрим, как все это работает.

Первая часть нового кода начинается с использования функции isset, чтобы проверить, были ли отправлены программе значения для всех полей. На основании такого подтверждения каждая из первых шести строк, находящихся внутри инструкции if, вызывает функцию get_post, которая появляется в самом конце программы. Эта функция делает небольшую, но очень важную работу: извлекает введенные данные из браузера.

## Массив $_POST

В главе 3 я уже упоминал о том, что браузер отправляет пользовательский ввод в виде GET- или POST-запроса. Обычно предпочтение отдается POST-запросу, поэтому именно он здесь и используется. Веб-сервер объединяет все введенное пользователем (даже если в форме заполнено под сотню полей) и помещает его в массив $_POST.

$_POST относится к ассоциативным массивам, рассмотренным в главе 6. Данные формы будут помещаться в ассоциативный массив по имени $_POST или $_GET в зависимости от того, какой метод используется для отправки формы, POST или GET. Оба этих массива могут быть прочитаны абсолютно одинаковым способом.

У каждого поля имеется элемент в массиве, имеющий точно такое же имя. Поэтому, если в форме есть поле по имени isbn, в массиве $_POST будет элемент с ключом isbn. PHP-программа может прочитать это поле, ссылаясь на него либо в виде $_POST['isbn'], либо в виде $_POST["isbn"] (в данном случае одинарные и двойные кавычки обладают одинаковым действием).

Если синтаксис работы с массивом $_POST кажется вам слишком сложным, можно спокойно воспользоваться приемом, показанным в примере 10.8: скопировать введенные пользователем данные в другие переменные, после чего о массиве $_POST можно будет забыть. Для PHP-программ это вполне нормальный подход: они получают все поля из массива $_POST в самом начале программы и больше к нему не обращаются.

Записывать элемент в массив $_POST нет никакого смысла. Он предназначен только для передачи информации из браузера в программу, и перед его изменением лучше скопировать данные в свои собственные переменные.

Возвращаясь к функции get_post, следует отметить, что она пропускает каждый получаемый ею элемент через функцию mysql_real_escape_string, чтобы удалить любые символы, которые злоумышленник может вставить в целях взлома или изменения вашей базы данных.

## Удаление записи

После загрузки разных переменных, которые могли быть отправлены с любыми переданными им значениями, программа проверяет, есть ли значение у переменной

$_POST['delete']. Если у нее есть значение, значит, пользователь щелкнул на кнопке DELETE RECORD (Удалить запись). В этом случае должно быть отправлено и значение $isbn.

Как уже говорилось, номер ISBN является уникальным идентификатором каждой записи. HTML-форма добавляет ISBN к строке запроса DELETE FROM, создаваемой в переменной $query, которая затем передается функции mysql_query, чтобы этот запрос попал к MySQL. Функция mysql_query возвращает либо TRUE, либо FALSE, и значение FALSE становится поводом для вывода сообщения об ошибке, объясняющего причины неудачи.

Если переменная $delete не содержит слова yes (да), выполняется следующая инструкция else. Переменной $query присваивается текст команды INSERT INTO, за которым следует перечисление пяти вставляемых значений. Затем эта переменная передается функции mysql_query, которая по завершении своей работы возвращает либо TRUE, либо FALSE. Если возвращено значение FALSE, выводится сообщение об ошибке.

## Отображение формы

Теперь мы добрались до той части кода, которая отображает небольшую форму, показанную в верхней части рис. 10.2. Структура echo <<<_END, которая выводит на экран все, что находится между тегами _END, должна быть вам знакома по предыдущим главам.

Вместо команды echo программа может завершить работу с интерпретатором PHP, используя тег ?>, выдать код HTML, а затем опять вернуться к работе с интерпретатором PHP, используя тег <?php. Какой из стилей применять — дело вкуса программиста, но я всегда рекомендую оставаться в рамках PHP-кода в силу следующих причин:

- при отладке (а также при разборе кода другими пользователями) это дает абсолютную уверенность в том, что все содержимое файла .php является кодом PHP. Поэтому не возникает нужды отлавливать временные выходы в код HTML;
- когда значение переменной PHP нужно вставить непосредственно в код HTML, можно просто набрать ее имя внутри этого кода. А при выходе в HTML нужно будет временно вернуться к обработке PHP, вывести переменную, а затем снова вернуться в HTML.

Раздел HTML-формы направляет все, что сделано в форме, в адрес файла sqltest.php. Это означает, что при отправке формы содержимое ее полей будет передано файлу sqltest.php, в котором и содержится сама программа. Форма настроена также на отправку полей в POST-, а не в GET-запросе. Причина в том, что GET-запросы являются дополнением к отправляемому URL-адресу и могут иметь в вашем браузере неприглядный вид. Эти запросы также позволяют пользователям без особого труда вносить изменения в отправляемую информацию и предпринимать попытки взлома вашего сервера. Поэтому при малейшей возможности нужно использовать для отправки данных POST-запросы, которые к тому же имеют преимущество, позволяющее скрыть отправляемые данные от просмотра.

При выводе полей формы HTML отображает кнопку отправки — Submit с именем ADD RECORD (Добавить запись), и форма закрывается. Обратите внимание

на использование тегов <pre> и </pre>, позволяющих воспользоваться моноширинным шрифтом и выровнять по линейке все элементы ввода данных. Внутри тегов <pre> в выводимые данные попадают также символы возврата каретки, стоящие в конце каждой строки.

## Запросы к базе данных

Далее код программы возвращается в привычное для нас русло примера 10.5, где в следующих четырех строках кода в адрес MySQL отправляется запрос на выдачу всех записей в таблице `classics`. Затем переменной `$rows` присваивается значение, равное количеству строк в таблице, и для отображения содержимого каждой строки вводится цикл `for`.

Следующую часть кода я немного упростил. Вместо использования тегов <br />, предназначенных в примере 10.5 для перевода строки, я решил воспользоваться тегом <pre>, чтобы выровнять на экране каждую запись, придав всему изображению привлекательный вид.

После отображения каждой записи следует вторая форма, которая также отправляет все свои данные файлу `sqltest.php` (то есть самой программе), но теперь в форме есть два скрытых поля: `delete` и `isbn`. Поле `delete` устанавливается в `yes`, а полю `isbn` присваивается значение, сохраненное в элементе массива `$row[4]`, в котором содержится ISBN для этой записи. Далее отображается кнопка `Submit` с надписью DELETE RECORD (Удалить запись), и форма закрывается. Затем фигурная скобка закрывает тело цикла `for`, который продолжает работу до тех пор, пока не будут отображены все записи.

В самом конце программы дано определение функции `get_post`, которую мы уже рассматривали. Вот так выглядит наша первая PHP-программа, предназначенная для управления базой данных MySQL. А теперь проверим, на что она способна.

## Запуск программы

После набора программы (и исправления всех опечаток) попробуйте ввести в поля ввода следующие сведения о книге `Moby Dick`, которые предназначены для добавления новой записи к базе данных:

```
Herman Melville
Moby Dick
Fiction
1851
9780199535729
```

Когда эти данные будут отправлены с помощью кнопки ADD RECORD (Добавить запись), прокрутите веб-страницу до самого конца, чтобы посмотреть только что добавленную информацию. Ее предполагаемый вид показан на рис. 10.3.

Теперь посмотрим, как работает удаление записи, специально создав для этого ненужную запись. Попробуйте ввести во все пять полей только одну цифру 1 и щелкните на кнопке ADD RECORD (Добавить запись). Если теперь прокрутить

страницу вниз, станет видна новая запись, состоящая из одних единиц. Конечно, такая запись в таблице не нужна, поэтому теперь щелкните на кнопке DELETE RECORD (Удалить запись) и снова прокрутите страницу вниз, чтобы убедиться в том, что запись была удалена.

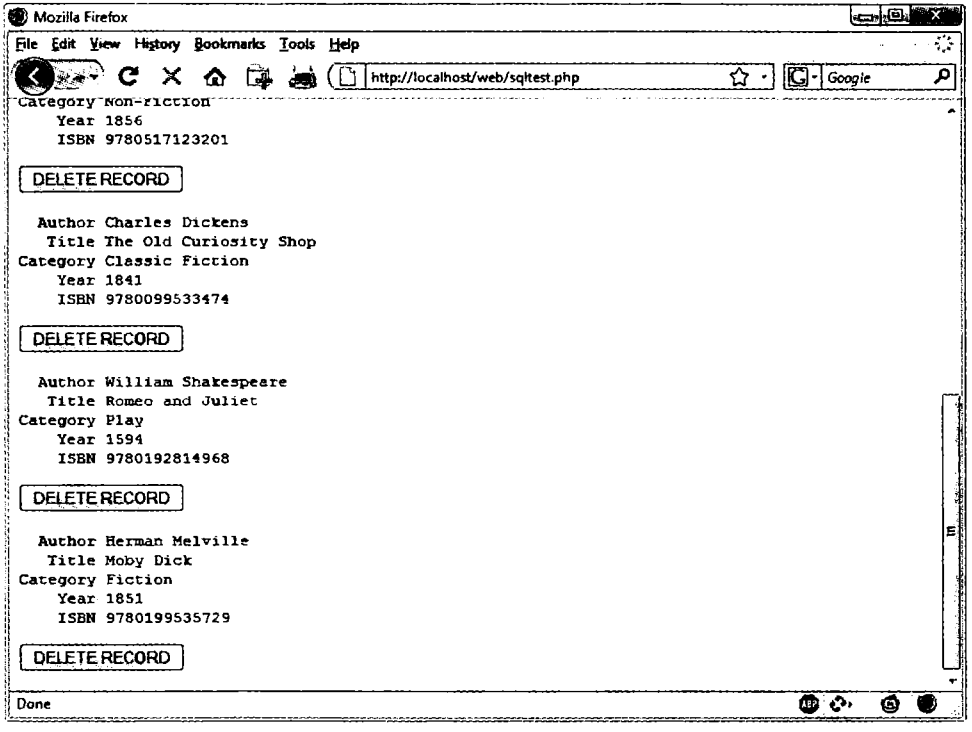



**Рис. 10.3.** Результат добавления сведений о книге Moby Dick в базу данных

Теперь, если все получилось, вы можете добавлять и удалять записи по своему усмотрению. Попробуйте сделать все это несколько раз, но основные записи (включая и новую запись о книге Moby Dick) оставьте нетронутыми, поскольку они нам еще пригодятся. Можно также попробовать добавить запись, состоящую из одних единиц, два раза и посмотреть, как при второй попытке будет выведено сообщение об ошибке, в котором говорится о том, что в таблице уже есть запись со значением ISBN, равным единице.

# Практическая работа с MySQL

Теперь вы готовы к изучению некоторых практических приемов, которые можно использовать в PHP для доступа к базам данных MySQL, куда включены задачи создания и удаления таблиц, вставки, обновления и удаления данных, а также защиты вашей базы данных и веб-сайта от злоумышленников. Учтите, что в следующих примерах предполагается, что вы создали программу `login.php`, рассмотренную ранее в этой главе.

## Создание таблицы

Предположим, что зоопарк поручил вам создать базу данных со сведениями обо всех содержащихся в нем представителях семейства кошачьих. Вам сказали, что в зоопарке девять представителей кошачьих: лев, тигр, ягуар, леопард, пума, гепард, рысь, каракал и домашний кот. Вам необходима отдельная графа для вида кошачьих. У каждого животного есть кличка, и для нее нужна еще одна графа. Необходимо также отслеживать возраст животных, и для этого требуется еще одна графа.

Разумеется, позже могут понадобиться дополнительные графы, например для учета рациона питания, сделанных прививок и других сведений, но пока, чтобы приступить к работе, достаточно и этих граф. Каждому животному нужен также уникальный идентификатор, поэтому решено было создать для него графу `id`.

В примере 10.9 показан код, который можно использовать для создания таблицы MySQL, хранящей все эти сведения. Операция присваивания, относящаяся к главному запросу, выделена полужирным шрифтом.

**Пример 10.9.** Создание таблицы cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "CREATE TABLE cats (
            id SMALLINT NOT NULL AUTO_INCREMENT,
            family VARCHAR(32) NOT NULL,
            name VARCHAR(32) NOT NULL,
            age TINYINT NOT NULL,
            PRIMARY KEY (id)
        )";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Как видите, MySQL-запрос очень похож на тот запрос, который приходилось набирать непосредственно в командной строке, за исключением того, что в нем нет завершающей точки с запятой, поскольку при доступе к MySQL из PHP она не нужна.

## Описание таблицы

Если вы не вошли в командную строку MySQL, то можно воспользоваться весьма полезным фрагментом кода, позволяющим проверить в браузере факт успешного создания таблицы. Этот код просто выдает запрос `DESCRIBE` *имя_таблицы*, а затем выводит HTML-таблицу, имеющую четыре заголовка: `Column` (Графа), `Type` (Тип), `Null` (Нуль) и `Key` (Ключ), ниже которых отображаются все имеющиеся в таблице

графы. Чтобы использовать код примера 10.10 с другими таблицами, нужно просто заменить в запросе имя cats именем новой таблицы.

**Пример 10.10.** Описание таблицы cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "DESCRIBE cats";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
$rows = mysql_num_rows($result);

echo "<table><tr> <th>Column</th> <th>Type</th>
      <th>Null</th> <th>Key</th> </tr>";

for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = mysql_fetch_row($result);
    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
    echo "</tr>";
}

echo "</table>";
?>
```

Информация, выводимая программой, должна иметь следующий вид:

```
Column Type        Null Key
id     smallint(6) NO   PRI
family varchar(32) NO
name   varchar(32) NO
age    tinyint(4)  NO
```

# Удаление таблицы

Удалить таблицу очень легко, и это весьма опасное действие нужно выполнять с большой осторожностью. В примере 10.11 показан необходимый для этого код. Но я не советую его применять до тех пор, пока не будут проработаны все остальные примеры, поскольку в результате его выполнения будет удалена таблица cats и вам придется создавать ее заново с помощью кода, показанного в примере 10.9.

**Пример 10.11.** Удаление таблицы cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
```

```
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "DROP TABLE cats";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

# Добавление данных

Добавим к таблице некоторые данные, воспользовавшись кодом, показанным в примере 10.12.

**Пример 10.12.** Добавление данных к таблице cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4)";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Если изменить значение переменной $query, как показано далее, можно добавить еще два элемента данных и еще раз вызвать программу из вашего браузера:

```
$query = "INSERT INTO cats VALUES(NULL, 'Cougar', 'Growler', 2)";
$query = "INSERT INTO cats VALUES(NULL, 'Cheetah', 'Charly', 3)";
```

Кстати, вы заметили, что в качестве первого параметра было передано значение NULL? Это сделано, потому что столбец id имеет тип AUTO_INCREMENT и MySQL решит, что нужно присвоить следующее доступное в используемой последовательности значение, поэтому мы просто передаем значение NULL, которое будет проигнорировано.

Разумеется, наиболее эффективный способ заполнения MySQL данными заключается в создании массива и вставке данных с использованием только одного запроса.

# Извлечение данных

Теперь, когда в таблицу cats введены некоторые данные, в примере 10.13 показано, как можно убедиться в том, что они были благополучно вставлены в эту таблицу.

**Пример 10.13.** Извлечение строк из таблицы cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "SELECT * FROM cats";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
$rows = mysql_num_rows($result);

echo "<table><tr> <th>Id</th> <th>Family</th>
    <th>Name</th><th>Age</th></tr>";

for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = mysql_fetch_row($result);
    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k) echo "<td>$row[$k]</td>";
    echo "</tr>";
}

echo "</table>";
?>
```

Этот код выдает простой MySQL-запрос `SELECT * FROM cats`, а затем отображает все возвращенные строки. Выходная информация должна иметь следующий вид:

```
Id Family  Name    Age
1  Lion    Leo     4
2  Cougar  Growler 2
3  Cheetah Charly  3
```

Здесь можно убедиться в том, что столбец `id` получает правильное автоприращение.

# Обновление данных

Изменение внесенных в таблицу данных также выполняется очень просто. Вы заметили, что кличка гепарда (cheetah) записана как `Charly`? Исправим ее на `Charlie`, как показано в примере 10.14.

**Пример 10.14.** Переименование гепарда Charly в Charlie

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
```

```
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "UPDATE cats SET name='Charlie' WHERE name='Charly'";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Если теперь еще раз запустить код из примера 10.13, то будет выведена следующая информация:

```
Id Family  Name    Age
1  Lion    Leo     4
2  Cougar  Growler 2
3  Cheetah Charlie 3
```

## Удаление данных

Пума по имени Growler была перевезена в другой зоопарк, поэтому сведения о ней нужно удалить из базы данных. Удаление данных из таблицы показано в примере 10.15.

**Пример 10.15.** Удаление сведений о пуме Growler из таблицы cats

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "DELETE FROM cats WHERE name='Growler'";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Здесь используется стандартный запрос DELETE FROM, и когда будет запущен код из примера 10.13, в отображаемой информации вы сможете увидеть, что строка была удалена:

```
Id Family  Name    Age
1  Lion    Leo     4
3  Cheetah Charlie 3
```

## Использование свойства AUTO_INCREMENT

При использовании свойства AUTO_INCREMENT вы не можете знать, какое значение было дано графе строки, которая предшествовала вставляемой строке. Но если нужно узнать об этом, можно позже обратиться к MySQL с помощью функции mysql_insert_id. Потребность в этом возникает довольно часто, например при обработке покупки можно вставить в таблицу Customers нового покупателя, а затем

сослаться на только что созданный CustId при вставке покупки в таблицу покупок. Пример 10.12 может быть переписан и превращен в пример 10.16, отображающий это значение после каждой вставки.

**Пример 10.16.** Добавление данных к таблице cats и отчет о вставленном id

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";

$result = mysql_query($query);
echo "Значение вставленного ID равно " . mysql_insert_id();
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
?>
```

Теперь содержимое таблицы приобретет следующий вид (обратите внимание на то, что ранее использовавшееся значение id, равное 2, повторно не используется, поскольку в некоторых случаях это может вызвать сложности):

```
Id Family  Name    Age
1  Lion    Leo     4
3  Cheetah Charlie 3
4  Lynx    Stumpy  5
```

## Использование идентификаторов вставленных строк

Зачастую данные вставляются сразу в несколько таблиц: за книгой следует ее автор, за покупателем — его покупка и т. д. При вставке данных в графы с автоприращением нужно будет запомнить вставленный ID, возвращенный для его сохранения в связанной таблице.

Предположим, к примеру, что в целях привлечения дополнительных средств над представителями кошачьих могут брать шефство какие-нибудь организации, и когда животное сохраняется в таблице кошек, нам нужно создать ключ для привязки его к организации, взявшей над ним шефство. Код для этого похож на код примера 10.16, за исключением того, что возвращенный ID, который был вставлен в таблицу, сохраняется в переменной $insertID, а затем используется в качестве составной части следующего запроса:

```
$query    = "INSERT INTO cats VALUES(NULL, 'Lynx', 'Stumpy', 5)";
$result   = mysql_query($query);
$insertID = mysql_insert_id();

$query    = "INSERT INTO owners VALUES($insertID, 'Ann', 'Smith')";
$result   = mysql_query($query);
```

Теперь животное связано со своим шефом посредством уникального кошачьего идентификатора, который был автоматически создан благодаря свойству AUTO_INCREMENT.

### Использование блокировок

Абсолютно безопасная процедура связывания таблиц посредством уже вставленного ID должна использовать блокировки. Если сразу несколько пользователей станут отправлять данные в одну и ту же таблицу, эти блокировки могут немного увеличить время отклика, но такие издержки вполне оправданны. При использовании блокировки наблюдается такая последовательность действий.

1. Блокировка первой таблицы (например, `cats`).
2. Вставка данных в первую таблицу.
3. Извлечение уникального ID из первой таблицы с помощью функции `mysql_insert_id`.
4. Снятие блокировки с первой таблицы.
5. Вставка данных во вторую таблицу.

Перед вставкой данных во вторую таблицу блокировку можно спокойно снять, поскольку вставляемый ID был извлечен и сохранен в переменной программы. Вместо блокировки можно, как показано в главе 9, использовать транзакцию, но это еще больше замедлит работу MySQL-сервера.

## Выполнение дополнительных запросов

Итак, хватит развлекаться с кошками. Чтобы исследовать некоторые более сложные запросы, нужно вернуться к использованию таблиц `customers` и `classics`, которые вы должны были создать в процессе работы с главой 8. В таблице `customers` должны быть сведения о двух покупателях, а в таблице `classics` — сведения о нескольких книгах. В них также совместно используется столбец с номером ISBN, названный `isbn`, который можно применять для выполнения дополнительных запросов.

Например, для отображения каждого из покупателей вместе с названиями и авторами купленных ими книг можно воспользоваться кодом, показанным на рис. 10.17.

**Пример 10.17.** Выполнение вторичного запроса

```
<?php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "SELECT * FROM customers";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
$rows = mysql_num_rows($result);

for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = mysql_fetch_row($result);
```

```
    echo "$row[0] purchased ISBN $row[1]:<br />";

    $subquery = "SELECT * FROM classics WHERE isbn='$row[1]'";

    $subresult = mysql_query($subquery);
    if (!$subresult) die ("Сбой при доступе к базе данных: " . mysql_error());
    $subrow = mysql_fetch_row($subresult);
    echo "  '$subrow[1]' by $subrow[0]<br />";
}
?>
```

В этой программе используется первичный запрос к таблице customers, чтобы найти всех покупателей. Затем на основе номера ISBN книг, приобретенных каждым покупателем, делается новый запрос к таблице classics, чтобы найти название и автора каждой из книг. При выполнении этого кода будет выведена следующая информация:

```
Mary Smith purchased ISBN 9780582506206:
    'Pride and Prejudice' by Jane Austen
Jack Wilson purchased ISBN 9780551123201:
    'The Origin of Species' by Charles Darwin
```

Хотя это и не имеет отношения к иллюстрации использования дополнительных запросов, но в данном случае можно вернуть точно такую же информацию, используя запрос с видом объединения NATURAL JOIN (см. главу 8):

```
SELECT name,isbn,title,author FROM customers
NATURAL JOIN classics;
```

# Предотвращение внедрения SQL-кода

Наверное, опасность передачи MySQL не прошедшей проверку вводимой пользователем информации все же недооценивается. Представьте, к примеру, что для идентификации пользователя используется следующий простой фрагмент кода:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

На первый взгляд этот код может показаться вполне приемлемым. Если пользователь вводит значения fredsmith и mypass, которые присваиваются переменным $user и $pass, то строка запроса, которая передается MySQL, приобретает следующий вид:

```
SELECT * FROM users WHERE user='fredsmith' AND pass='mypass'
```

Все это выглядит вполне пристойно, но что будет, если кто-нибудь введет для переменной $user следующую строку (а для переменной $pass вообще ничего не станет вводить):

```
admin' #
```

Посмотрим на строку, которая будет отправлена MySQL:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Вы поняли, в чем тут проблема (выделено полужирным шрифтом)? В MySQL с символа # начинается комментарий. Поэтому пользователь войдет в систему под именем admin (предположим, что в системе есть пользователь admin), и ему не нужно будет вводить пароль. В следующей строке исполняемая часть запроса выделена полужирным шрифтом, а все остальные символы будут проигнорированы.

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Но вам еще очень посчастливится, если весь вред, нанесенный злоумышленником, этим и ограничится. По крайней мере, вам, может быть, удастся войти в свое приложение и отменить все изменения, внесенные туда пользователем, вошедшим в систему под именем admin. А что, если ваша прикладная программа удаляет пользователя из базы данных? Тогда код может иметь следующий вид:

```
$user  = $_POST['user'];
$pass  = $_POST['pass'];
$query = "DELETE FROM users WHERE user='$user' AND pass='$pass'";
```

На первый взгляд, с ним опять вроде бы все в порядке, но что получится, если кто-нибудь введет для переменной $user следующее значение:

```
anything' OR 1=1 #
```

MySQL интерпретирует это следующим образом (также выделено полужирным шрифтом):

```
DELETE FROM users WHERE user='anything' OR 1=1 #' AND pass=''
```

Условие этого SQL-запроса всегда будет истинным, поэтому вы лишитесь всей базы данных, в которой содержатся сведения о пользователях. Так что же нужно делать в случае подобной атаки?

Не стоит полагаться на встроенное в PHP свойство «волшебных кавычек» (magic quotes), которое автоматически отключает любые символы одинарных и двойных кавычек путем установки перед ними символа обратного слеша (\). Почему? Да потому, что это свойство может быть отключено. Многие программисты именно так и делают, чтобы вставить собственный код, обеспечивающий безопасность. Поэтому нет никаких гарантий того, что на вашем рабочем сервере это свойство не было отключено. Фактически в PHP 5.3.0 его использование не приветствовалось, а из версии 6 оно вообще было удалено.

Вместо этого для всех обращений к MySQL нужно всегда использовать функцию mysql_real_escape_string. В примере 10.18 показана функция, которую можно использовать, чтобы удалить любые «волшебные кавычки», добавленные во введенную пользователем строку, а затем соответствующим образом обезвредить все имеющиеся в ней опасные компоненты.

**Пример 10.18.** Способ обезвреживания данных, введенных пользователем, приемлемый для MySQL

```
<?php
function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return mysql_real_escape_string($string);
}
?>
```

Функция get_magic_quotes_gpc возвращает TRUE, если свойство «волшебных кавычек» находится в активном состоянии. Если это так, любые добавленные к строке слеши подлежат удалению, в противном случае функция mysql_real_eascape_string может отключить некоторые символы дважды, сделав строки непригодными для дальнейшего использования. В примере 10.19 показано, как можно вставить функцию mysql_fix в ваш код.

**Пример 10.19.** Способ безопасного доступа к MySQL при использовании данных, введенных пользователем

```
<?php
$user  = mysql_fix_string($_POST['user']);
$pass  = mysql_fix_string($_POST['pass']);
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return mysql_real_escape_string($string);
}
?>
```

Следует запомнить, что функцию mysql_escape_string можно использовать только при активной базе данных MySQL, в противном случае произойдет ошибка.

## Использование указателей места заполнения

Другой, фактически «пуленепробиваемый» способ предохранения от внедрения SQL-кода заключается в использовании так называемых *указателей места заполнения*. Его смысл — в предварительном определении запроса с использованием символов вопросительного знака (?) в тех местах, где появятся данные. Затем вместо непосредственного вызова MySQL-запроса осуществляется вызов предварительно определенного запроса, которому передаются данные. В результате этого гарантируется, что каждый элемент введенных данных попадает непосредственно в базу данных и не сможет интерпретироваться в качестве SQL-запроса. Иными словами, внедрение SQL-кода становится невозможным.

При использовании командной строки MySQL последовательность выполняемых запросов будет иметь вид, показанный в примере 10.20.

**Пример 10.20.** Использование указателей места заполнения

```
PREPARE statement FROM "INSERT INTO classics VALUES(?,?,?,?,?)";

SET @author   = "Emily Brontл",
    @title    = "Wuthering Heights",
    @category = "Classic Fiction",
    @year     = "1847",
    @isbn     = "9780553212587";

EXECUTE statement USING @author,@title,@category,@year,@isbn;

DEALLOCATE PREPARE statement;
```

Первая команда подготавливает инструкцию statement для вставки данных в таблицу classics. Здесь показано, что вместо значений или переменных для вставляемых в таблицу данных эта инструкция содержит последовательность символов ?. Это и есть указатели места заполнения.

В следующих пяти строках переменным MySQL присваиваются значения, соответствующие вставляемым данным. Затем при выполнении предварительно определенной инструкции эти переменные передаются ей в качестве параметров. И наконец, инструкция удаляется, чтобы освободить задействованные ею ресурсы.

В PHP код для этой процедуры имеет вид, показанный в примере 10.21 (здесь предполагается, что вы уже создали файл login.php с нужными для доступа к базе данных сведениями).

**Пример 10.21.** Использование указателей места заполнения с PHP

```
<?php
require 'login.php';

$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = 'PREPARE statement FROM "INSERT INTO classics
    VALUES(?,?,?,?,?)"';
mysql_query($query);

$query = 'SET @author = "Emily Brontл",'
         '@title = "Wuthering Heights",' .
         '@category = "Classic Fiction",'
         '@year = "1847",' .
         '@isbn = "9780553212587"';
mysql_query($query);

$query = 'EXECUTE statement USING @author,@title,@category,@year,@isbn';
mysql_query($query);

$query = 'DEALLOCATE PREPARE statement';
mysql_query($query);
?>
```

После подготовки инструкции ее можно использовать необходимое количество раз до тех пор, пока она не будет удалена. Такие инструкции часто применяются внутри циклов для быстрой вставки данных в базы путем присваивания значений переменным MySQL с последующим выполнением инструкции. Данный подход более эффективен, чем создание заново всей инструкции при каждом проходе цикла.

## Предотвращение внедрения HTML-кода

Нужно позаботиться о защите еще от одного вида внедрения, который связан не с безопасностью ваших собственных веб-сайтов, а с конфиденциальностью и защитой пользовательских данных. Речь идет о межсайтовом скриптинге (Cross Site Scripting), называемом также XSS.

Эта разновидность внедрения происходит в том случае, когда вы разрешаете пользователю вводить, а затем отображать на вашем веб-сайте HTML- или, что случается чаще, JavaScript-код. Одним из мест, где это часто происходит, является форма для комментариев. Чаще всего злоумышленник пытается написать код, ворующий у пользователей вашего сайта cookie, позволяющие ему узнать пары «имя пользователя — пароль» или другую информацию. Хуже того, злоумышленник может предпринять атаку с целью загрузки на пользовательский компьютер троянского коня.

Чтобы предотвратить это внедрение, нужно лишь вызвать функцию `htmlentities`, выявляющую все коды разметки HTML и заменяющую их формой, которая отображает символы, но не позволяет браузеру действовать в соответствии с их предназначением. Рассмотрим, к примеру, следующий код HTML:

```
<script src='http://x.com/hack.js'> </script><script>hack();</script>
```

Этот код загружает программу на JavaScript, а затем выполняет вредоносные функции. Но если сначала этот код будет пропущен через функцию `htmlentities`, то он превратится в следующую абсолютно безвредную строку:

```
&lt;script src='http://x.com/hack.js'&gt;
 &lt;/script&gt;&lt;script&gt;hack();&lt;/script&gt;
```

Поэтому если вы когда-нибудь соберетесь отобразить какие-нибудь данные, введенные пользователем, то нужно немедленно или сразу же после первого сохранения в базе данных обезвредить их с помощью функции `htmlentities`. Для этого я рекомендую вам создать новую функцию наподобие первой функции, показанной в примере 10.22, которая способна обезвредить попытки как SQL- так и XSS-внедрения.

**Пример 10.22.** Функции для предотвращения атак внедрения SQL и XSS

```
<?php
function mysql_entities_fix_string($string)
{
    return htmlentities(mysql_fix_string($string));
}

function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return mysql_real_escape_string($string);
```

```
}
?>
```

Функция mysql_entities_fix_string сначала вызывает функцию mysql_fix_string, а затем, прежде чем вернуть полностью обезвреженную строку, пропускает результат через функцию htmlentities. В примере 10.23 показана новая, «максимально защищенная» версия примера 10.19.

**Пример 10.23.** Способ безопасного доступа к MySQL и предотвращения XSS-атак

```
<?php
$user  = mysql_entities_fix_string($_POST['user']);
$pass  = mysql_entities_fix_string($_POST['pass']);
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";

function mysql_entities_fix_string($string)
{
    return htmlentities(mysql_fix_string($string));
}

function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return mysql_real_escape_string($string);
}
?>
```

После изучения способов объединения PHP с MySQL и избавления от опасности, которой грозят введенные пользователем данные, в следующей главе будет подробно рассмотрена обработка данных формы, включая вопросы проверки приемлемости данных, выбора нескольких элементов, сопоставления с образцом и обеспечения безопасности.

# Проверьте ваши знания

1. Какая стандартная PHP-функция предназначена для подключения к базе данных MySQL?
2. В каком случае работу функции mysql_result нельзя признать оптимальной?
3. Назовите причину, по которой в большинстве случаев для отправки данных формы лучше воспользоваться методом POST, а не методом GET.
4. Как определить самое последнее значение, введенное в столбец со свойством AUTO_INCREMENT?
5. Какая PHP-функция отключает опасные символы в строке, делая ее пригодной к использованию в MySQL?
6. Какая функция может использоваться для предотвращения XSS-атак внедрения?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 10».

# 11 Обработка форм

Основным способом взаимодействия пользователей с PHP и MySQL является применение HTML-форм. Эти формы появились на заре разработки Всемирной паутины, в 1993 году, даже раньше, чем электронная коммерция, и благодаря простоте и легкости использования не утратили своего значения и по сей день.

Разумеется, с годами HTML-формы совершенствовались, получая дополнительные функциональные возможности обработки информации, поэтому данная глава познакомит вас с современными методами обработки формы и продемонстрирует самые лучшие способы реализации форм для достижения наибольшей степени удобства и безопасности.

## Создание форм

Обработка форм является многоступенчатым процессом. Сначала создается форма, в которую пользователь может вводить необходимые данные. Затем эти данные отправляются веб-серверу, где происходит их разбор, зачастую совмещаемый с проверкой на отсутствие ошибок. Если код PHP найдет одно или несколько полей, требующих повторного ввода, форма может быть заново отображена вместе с сообщением об ошибке. Когда качество введенных данных удовлетворяет программу, она предпринимает некоторые действия, нередко привлекая для этого базы данных, к примеру, для ввода сведений о покупке.

Для создания формы потребуются как минимум следующие элементы:

- открывающий и закрывающий теги — соответственно `<form>` и `</form>`;
- тип передачи данных, задаваемый одним из двух методов `GET` или `POST`;
- одно или несколько полей для ввода данных;
- URL-адрес назначения, по которому будут отправлены данные формы.

В примере 11.1 показана очень простая форма, созданная с использованием кода PHP. Наберите этот код и сохраните его в файле `formtest.php`.

**Пример 11.1.** `formtest.php` – простой обработчик формы на PHP

```
<?php // formtest.php
echo <<<_END
<html>
```

```
    <head>
        <title>Form Test</title>
    </head>
    <body>
    <form method="post" action="formtest.php" />
        Как Вас зовут?
        <input type="text" name="name" />
        <input type="submit" />
    </form>
    </body>
</html>
_END;
?>
```

В первую очередь следует отметить, что в этом примере использован прием, уже встречавшийся в данной книге: вместо многократного входа в PHP-код и выхода из него для вывода многострочного HTML-кода я обычно применяю конструкцию `echo <<<_END..._END`.

Внутри этого многострочного вывода находится стандартный код, с которого начинается HTML-документ, отображающий заголовок документа и обозначающий начало его тела. Затем следует форма, настроенная на отправку своих данных с использованием POST-метода в адрес PHP-программы `formtest.php`, то есть этой самой программы.

Остальной код программы закрывает все открытые элементы: форму, тело HTML-документа и PHP-инструкцию `echo <<<_END`. Результат запуска этой программы в веб-браузере показан на рис. 11.1.

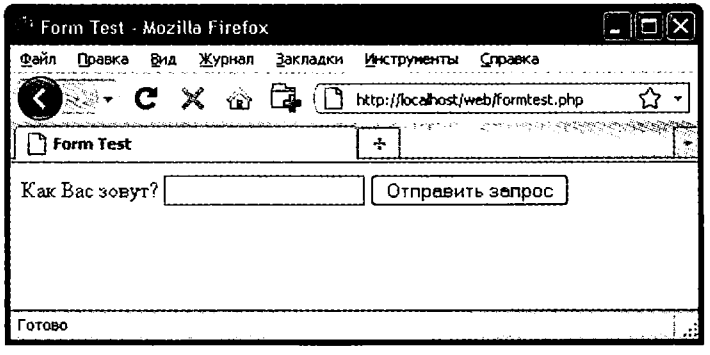



**Рис. 11.1.** Результат запуска программы formtest.php в веб-браузере

# Извлечение отправленных данных

В примере 11.1 представлена только одна из частей многоступенчатого процесса обработки формы. Если ввести имя и щелкнуть на кнопке **Отправить запрос**, то абсолютно ничего, кроме повторного отображения формы, не произойдет. Поэтому сейчас нужно добавить PHP-код, обрабатывающий отправляемые формой данные.

В примере 11.2 показана расширенная версия предыдущей программы, включающая обработку данных. Наберите этот код или измените код программы formtest.php, добавив в него новые строки, сохраните программу в файле formtest2.php и попробуйте ее запустить. Результат запуска программы и введенное имя показаны на рис. 11.2.

**Пример 11.2.** Обновленная версия formtest.php

```
<?php // formtest2.php
if (isset($_POST['name'])) $name = $_POST['name'];
else $name = "(Не введено)";

echo <<<_END
<html>
    <head>
        <title>Form Test</title>
    </head>
    <body>
    Вас зовут: $name<br />
    <form method="post" action="formtest2.php">
        Как Вас зовут?
        <input type="text" name="name" />
        <input type="submit" />
    </form>
    </body>
</html>
_END;
?>
```

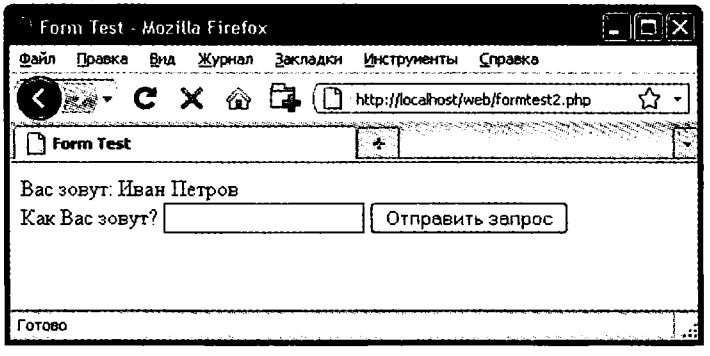



**Рис. 11.2.** Программа formtest.php с обработкой данных

Изменения касаются двух строк в начале программы, в которых проверяется, содержит ли ассоциативный массив $_POST отправленное поле name. Ассоциативный массив $_POST был рассмотрен в предыдущей главе, он содержит элемент для каждого поля HTML-формы. В примере 11.2 для вводимого имени использовалось поле name, а для отправки данных формы был избран метод POST, поэтому значение элемента name массива $_POST содержится в элементе массива $_POST['name'].

PHP-функция isset используется для проверки наличия значения у элемента $_POST['name']. Если значение не было отправлено, программа присваивает переменной $name значение «(Не введено)». А если значение было отправлено, то оно сохраняется в этой переменной. После тега <body> была введена еще одна строка, предназначенная для отображения значения, сохраненного в переменной $name.

## register_globals: склонность к использованию устаревших решений

Еще до того как вопросы безопасности приобрели столь большое значение, PHP по умолчанию присваивал значения массивов $_POST и $_GET непосредственно PHP-переменным. К примеру, в использовании инструкции $name=$_POST['name']; не было необходимости, поскольку PHP автоматически присваивал это значение переменной $name, как только запускалась программа.

Первоначально (до появления версии PHP 4.2.0) это считалось весьма продуктивной идеей, избавляющей от набора большого объема дополнительного кода, но в данный момент такая практика уже не приветствуется, и по умолчанию это свойство отключено. Если обнаружится, что на рабочем веб-сервере, для которого выполняется разработка, свойство register_globals включено, нужно срочно потребовать от администратора сервера его отключения.

Зачем отключать свойство register_globals? Оно дает возможность кому угодно ввести GET-запрос в конце URL-адреса, например: http://myserver.com?override=1, и если в вашем коде где-нибудь используется переменная $override и вы забыли ее инициализировать (воспользовавшись, к примеру, инструкцией $override=0;), то это действие может поставить работу программы под угрозу.

Фактически из-за того, что многие установки во Всемирной сети сохраняют эту лазейку, я советую вам всегда инициализировать любую, используемую вами переменную на тот случай, что ваш код будет когда-нибудь запущен на подобной системе. В программировании инициализация является правилом хорошего тона, поскольку каждую инициализацию можно прокомментировать, чтобы оставить напоминание самому себе и другим программистам о назначении этой переменной.

Если когда-нибудь придется обслуживать код, в котором, на ваш взгляд, некоторым переменным присвоены значения, не имеющие явного смысла, вы можете вполне обоснованно предположить, что программист создавал этот код, используя свойство register_globals, и эти значения предполагается извлечь из отправленных POST- или GET-запросов. Если так оно и есть, я рекомендую вам переписать код так, чтобы значения для этих переменных загружались в явном виде из соответствующего массива $_POST или $_GET.

## Значения по умолчанию

Иногда представляется удобным предложить посетителям вашего сайта принять в веб-форме значения по умолчанию. Предположим, к примеру, что вы разместили на сайте по недвижимости виджет, представляющий собой калькулятор погашения кредита. При этом есть смысл ввести в него значения по умолчанию,

скажем, 25 лет и 6 % годовых, чтобы пользователю осталось только ввести либо основную сумму заимствования, либо посильную для него сумму ежемесячных выплат. HTML-код для этих двух значений мог бы иметь вид, показанный в примере 11.3.

**Пример 11.3.** Установка значений по умолчанию

```
<form method="post" action="calc.php"><pre>
          Сумма заимствования <input type="text" name="principle" />
  Ежемесячная выплата <input type="text" name="monthly" />
       Количество лет <input type="text" name="years" value="25" />
      Процент годовых <input type="text" name="rate" value="6" />
                      <input type="submit" />
</pre></form>
```

Если есть желание испробовать в работе этот HTML-код (а также другие подобные примеры), наберите и сохраните его в файле с расширением HTML, например в файле test.html, а затем загрузите этот файл в свой браузер.

Обратите внимание на третий и четвертый элементы ввода данных. За счет указания значения для параметра `value` в поле отображается значение по умолчанию, которое пользователи в дальнейшем смогут изменить, если у них появится такое желание. Задавая вполне обоснованные значения по умолчанию, можно добиться более дружелюбного поведения от своих веб-форм за счет минимизации необязательного ввода данных. Результат работы предыдущего кода показан на рис. 11.3. Разумеется, он был создан только для иллюстрации значений по умолчанию, и поскольку программа `calc.php` не была написана, форма после передачи данных никак на это не отреагирует.

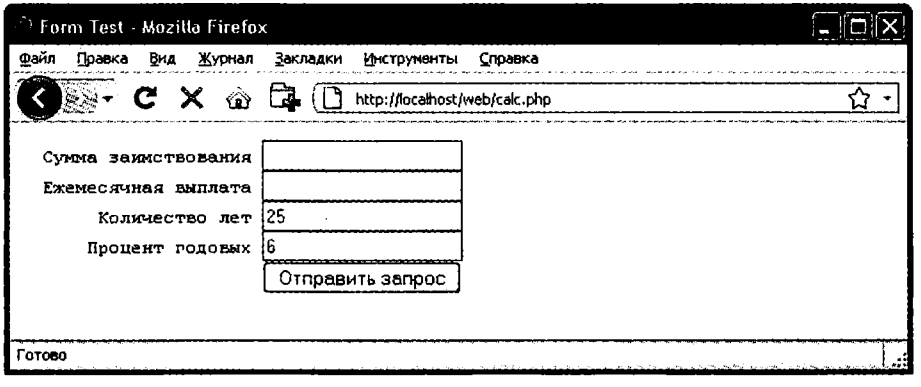



**Рис. 11.3.** Использование значений по умолчанию для избранных полей формы

Значения по умолчанию используются также для скрытых полей, которые применяются тогда, когда вы хотите наряду с данными, введенными пользователем, отправить из веб-страницы в адрес программы какую-нибудь дополнительную информацию. Скрытые поля будут рассмотрены в этой главе чуть позже.

# Типы элементов ввода данных

HTML-формы обладают завидной универсальностью, позволяя отправлять данные из довольно широкого диапазона различных типов элементов ввода, начиная с текстовых полей и текстовых областей и заканчивая флажками, переключателями и т. п.

## Текстовое поле

Наверное, самым распространенным типом элемента, применяемого для ввода данных, является текстовое поле. Оно воспринимает широкий диапазон буквенно-цифрового текста и других символов в пределах однострочного окна. Типовой формат текстового поля для ввода информации имеет следующий вид:

```
<input type="text" name="имя" size="размер" maxlength="длина" value="значение" />
```

Параметры `name` (имя) и `value` (значение) мы уже рассматривали, но здесь представлены еще два параметра: `size` (размер) и `maxlength` (максимальная длина). Параметр `size` определяет ширину поля в символах текущего шрифта, каким оно появится на экране, а `maxlength` определяет максимальное количество символов, которое пользователю разрешено вводить в это поле.

Единственными обязательными параметрами являются `type` (тип), сообщающий веб-браузеру ожидаемый тип элемента ввода данных, и `name` (имя), дающий вводимым данным имя, которое используется в дальнейшем для обработки поля после получения отправленной формы.

## Текстовая область

Когда нужно принять вводимые данные, превышающие по объему короткую строку текста, используется текстовая область. Она похожа на текстовое поле, но поскольку в нее разрешает вводить сразу несколько строк, имеет несколько иные параметры. Ее типовой формат выглядит следующим образом:

```
<textarea name="имя" cols="ширина" rows="высота" wrap="тип">
</textarea>
```

Первое, на что следует обратить внимание, — использование текстовой областью собственного тега <textarea>, который не является подвидом тега <input>, поэтому для него нужен закрывающий тег </textarea>, чтобы закрыть элемент ввода данных.

Если есть какой-нибудь текст, который нужно отобразить по умолчанию, то вместо использования параметра, позволяющего задавать подобное значение, нужно поместить этот текст перед закрывающим тегом </textarea>, и тогда он будет отображен и сможет редактироваться пользователем:

```
<textarea name="имя" cols="ширина" rows="высота" wrap="тип">
Это текст, отображаемый по умолчанию.
</textarea>
```

Для управления шириной и высотой текстовой области используются параметры `cols` (графы) и `rows` (строки). Для задания размеров области в обоих параметрах в качестве единицы измерения применяются пространства, занимаемые символом текущего шрифта. Если эти значения опустить, будет создана текстовая область

с размерами по умолчанию, которые зависят от используемого браузера, поэтому, чтобы точно знать, в каком виде должна появиться ваша веб-форма, нужно всегда задавать значения этих параметров.

И наконец, при помощи параметра wrap (перенос) можно управлять порядком переноса вводимого в область текста (и тем, как этот перенос будет отправляться на сервер). В табл. 11.1 показаны доступные типы переноса. Если не указывать значение параметра wrap, будет задействован мягкий перенос (soft wrapping).

**Таблица 11.1.** Типы переноса, доступные в области ввода <textarea>

| Тип | Действие |
|---|---|
| off | Текст не переносится, и строки появляются в строгом соответствии с тем, как их вводит пользователь |
| soft | Текст переносится, но отправляется на сервер одной длинной строкой без символов возврата каретки и перевода строки |
| hard | Текст переносится и отправляется на сервер в формате переноса с «мягким» возвратом в начало следующей строки и переводом строки |

## Флажки

Если пользователю нужно предложить выбор из нескольких вариантов данных, при котором он может остановиться на одном или нескольких вариантах, то для этого всегда используются флажки. Формат флажков выглядит следующим образом:

```
<input type="checkbox" name="имя" value="значение" checked="checked" />
```

Если в этот формат включается параметр checked (установлен), флажок появляется в браузере в уже установленном виде (строка, присваиваемая этому параметру, не имеет никакого значения, он просто должен присутствовать в формате). Если данный параметр не включать в формат, флажок будет отображен в неустановленном виде. Примером задания такого флажка может послужить следующий код:

```
Я согласен <input type="checkbox" name="agree" />
```

Если пользователь не установит флажок, значение передано не будет. Но если флажок будет установлен, то для поля по имени agree будет передано значение on. Если вы предпочитаете вместо on отправить собственное значение (например, число 1), можно воспользоваться следующим синтаксисом:

```
Я согласен <input type="checkbox" name="agree" value="1" />
```

В то же время, если вы хотите при отправке формы предложить своим читателям информационный бюллетень, то может появиться желание отобразить флажок установленным по умолчанию:

```
Подписаться? <input type="checkbox" name="news" checked="checked" />
```

Если есть потребность в одновременном выборе группы элементов, то всем им нужно присвоить одинаковые имена. Но при этом нужно иметь в виду, что если в качестве имени не будет передано имя массива, будет отправлен только последний

отмеченный элемент формы. Код из примера 11.4 позволяет пользователю выбрать любимые сорта мороженого (результат работы этого кода в браузере показан на рис. 11.4).

**Пример 11.4.** Предложение сделать выбор, установив сразу несколько флажков

```
    Ванильное <input type="checkbox" name="ice" value="Vanilla" />
   Шоколадное <input type="checkbox" name="ice" value="Chocolate" />
  Земляничное <input type="checkbox" name="ice" value="Strawberry" />
```

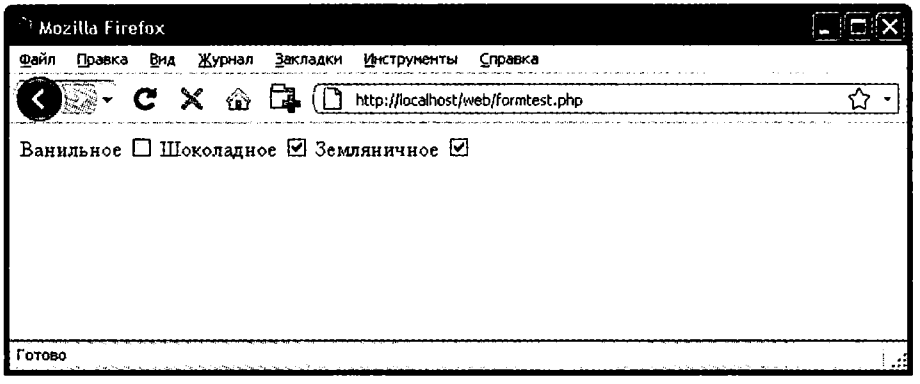



**Рис. 11.4.** Использование флажков для быстрого выбора

Если установлен только один флажок, например второй, то будет передан только этот элемент (полю с именем ice будет присвоено значение Шоколадное). Но если будут выбраны два и более флажка, будет отправлено только последнее значение, а все предыдущие будут проигнорированы.

Если нужно добиться исключающего поведения, то есть передачи только одного элемента, следует воспользоваться переключателями (см. следующий подраздел), но чтобы позволить отправку сразу нескольких значений, нужно немного изменить код HTML, как показано в примере 11.5 (обратите внимание на добавление квадратных скобок [] после значений ice):

**Пример 11.5.** Отправка нескольких значений с помощью массива

```
   Ванильное <input type="checkbox" name="ice[]" value="Vanilla" />
  Шоколадное <input type="checkbox" name="ice[]" value="Chocolate" />
 Земляничное <input type="checkbox" name="ice[]" value="Strawberry" />
```

Теперь, если при отправке формы установлены какие-нибудь из этих флажков, будет отправлен массив по имени ice, содержащий любые значения. В любом случае можно извлечь в переменную либо отдельное значение, либо массив значений:

```
$ice = $_POST['ice'];
```

Если поле ice было отправлено в виде отдельного значения, переменная $ice будет содержать отдельную строку, например Земляничное. Но если в форме под именем ice был определен массив (как в примере 11.5), переменная $ice будет массивом и номера элементов этого массива будут номерами отправленных значений.

В табл. 11.2 показаны семь возможных наборов значений, которые могут быть переданы этим HTML-кодом для одного, двух или трех установленных флажков. В каждом из случаев будет создаваться массив из одного, двух или трех элементов.

**Таблица 11.2.** Семь возможных наборов значений для массива $ice

| При отправке одного значения | При отправке двух значений | При отправке трех значений |
|---|---|---|
| $ice[0] => Ванильное | $ice[0] => Ванильное | $ice[0] => Ванильное |
| $ice[0] => Шоколадное | $ice[1] => Шоколадное | $ice[1] => Шоколадное |
| $ice[0] => Земляничное | | $ice[2] => Земляничное |
| | $ice[0] => Ванильное | |
| | $ice[1] => Земляничное | |
| | | |
| | $ice[0] => Шоколадное | |
| | $ice[1] => Земляничное | |

Если переменная `$ice` является массивом, то для отображения ее содержимого можно использовать очень простой PHP-код:

```
foreach($ice as $item) echo "$item<br />";
```

В нем используется стандартная PHP-конструкция `foreach`, осуществляющая последовательный перебор элементов массива `$ice` и передающая значение каждого элемента переменной `$item`, содержимое которой затем отображается с помощью команды `echo`. Тег `<br />` служит только для HTML-форматирования, чтобы после отображения каждого сорта осуществлялся перевод на новую строку. По умолчанию поля флажков имеют квадратную форму.

## Переключатели

Переключатели (radio buttons — кнопки радиоприемника) названы так по аналогии с утапливаемыми кнопками настройки на фиксированные частоты, которые встречались на многих старых радиоприемниках, где любая ранее утопленная кнопка при нажатии другой кнопки возвращалась в первоначальную позицию. Переключатели используются в тех случаях, когда нужно из двух и более вариантов выбрать и вернуть только один. Все кнопки группы должны использовать одно и то же имя, и поскольку возвращается только одно значение, массив передавать не требуется.

К примеру, если веб-сайт предлагает выбор времени доставки покупок из вашего магазина, то для этого можно воспользоваться HTML-кодом, показанным в примере 11.6 (результат его работы изображен на рис. 11.5).

**Пример 11.6.** Использование переключателей

```
 8.00-12.00<input type="radio" name="time" value="1" />|
12.00-16.00<input type="radio" name="time" value="2" checked="checked" />|
16.00-20.00<input type="radio" name="time" value="3" />
```

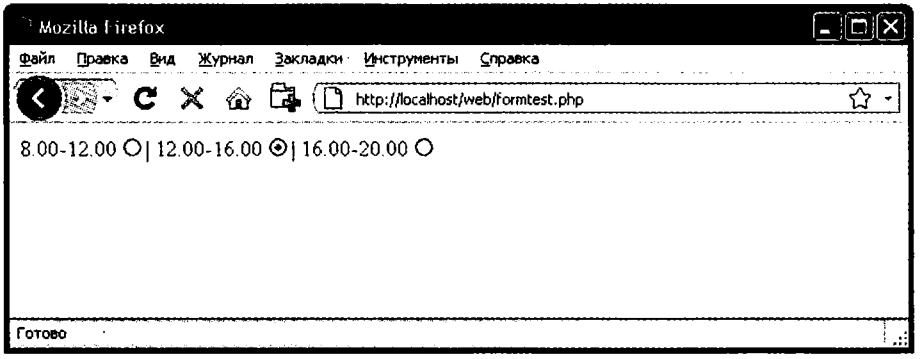



**Рис. 11.5.** Выбор единственного значения с помощью переключателей

Здесь по умолчанию задается второй вариант: `12.00-16.00`. Наличие значения по умолчанию гарантирует, что пользователь выберет хотя бы одно время доставки, которое затем может быть изменено на любое из двух оставшихся в соответствии с их предпочтениями. Если бы не был заранее выбран один из этих вариантов, пользователь мог забыть сделать свой выбор и для времени доставки не было бы передано никакого значения. По умолчанию переключатели имеют форму окружностей.

## Скрытые поля

Иногда бывает удобно пользоваться скрытыми полями формы, чтобы получить возможность отслеживать состояние ее ввода. Например, может потребоваться узнать, отправлена форма или нет. Эти сведения можно получить, добавив к PHP-коду фрагмент кода HTML:

```
echo '<input type="hidden" name="submitted" value="yes" />'
```

Это простая PHP-инструкция `echo`, добавленная к полю ввода HTML-формы. Предположим, что форма была создана вне программы и показана пользователю. При первом получении PHP-программой введенных данных эта строка кода не будет запущена, поэтому поля с именем `submitted` не будет. Программа PHP воссоздает форму, добавляя к ней поле ввода.

Поэтому, когда пользователь отправит форму еще раз, PHP-программа получит ее с полем `submitted`, имеющим значение `yes`. Существование этого поля можно легко проверить с помощью следующего кода:

```
if (isset($_POST['submitted']))
{...
```

Скрытые поля могут пригодиться также для хранения других сведений, например идентификационной строки сеанса, которая может быть создана для идентификации пользователя, и т. д.

Скрытые поля нельзя считать безопасными, поскольку они этим свойством не обладают. Код HTML, которым задаются эти поля, может быть легко просмотрен с помощью свойства браузера, позволяющего просматривать исходный код страницы.

## Список

Тег select позволяет создавать раскрывающийся список, предлагающий выбор одного или нескольких значений. Для его создания используется следующий синтаксис:

```
<select name="имя" size="размер" multiple="multiple">
```

Параметром size (размер) задается количество отображаемых строк. Щелчок на кнопке отображения приводит к раскрытию списка, показывающего все варианты. Если применяется параметр multiple (множественный выбор), из списка путем удерживания во время щелчка клавиши Ctrl могут быть выбраны сразу несколько вариантов. Чтобы спросить у пользователя, какой из пяти овощей он любит больше всего, можно воспользоваться кодом, предлагающим единичный выбор (пример 11.7).

**Пример 11.7.** Использование поля со списком

```
Овощи <select name="veg" size="1">
<option value="Горох">Горох</option>
<option value="Фасоль">Фасоль</option>
<option value="Морковь">Морковь</option>
<option value="Капуста">Капуста</option>
<option value="Брокколи">Брокколи</option>
</select>
```

Этот HTML-код предлагает пять вариантов, на первый из которых, Горох, выпадает предварительный выбор (благодаря тому что он стоит первым в списке). На рис. 11.6 показан внешний вид раскрытого щелчком списка, после того как был выделен вариант Морковь. Если требуется выбрать другой вариант, предлагаемый по умолчанию (например, Фасоль), нужно воспользоваться параметром selected:

```
<option selected="selected" value="Фасоль">Фасоль</option>
```

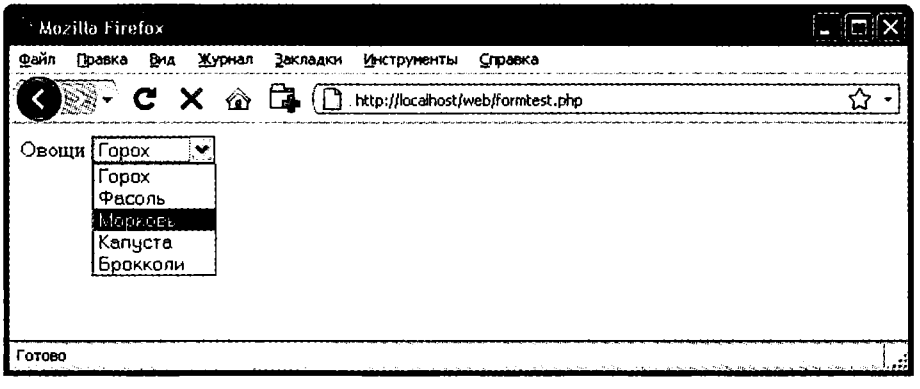



**Рис. 11.6.** Создание раскрывающегося списка с помощью тега select

Можно также дать пользователям возможность выбрать более одного варианта (пример 11.8).

**Пример 11.8.** Использование select с параметром multiple

```
Овощи <select name="veg" size="5" multiple="multiple">
<option value="Горох">Горох</option>
<option value="Фасоль">Фасоль</option>
<option value="Морковь">Морковь</option>
<option value="Капуста">Капуста</option>
<option value="Брокколи">Брокколи</option>
</select>
```

Код HTML не претерпел при этом значительных изменений, было лишь изменено значение параметра size на 5 и добавлен параметр multiple. Но теперь, судя по рис. 11.7, можно выбрать более одного варианта, удерживая при щелчке клавишу Ctrl. При желании можете отказаться от параметра size, внешний вид от этого не изменится, но для более длинного списка может понадобиться намного больше места на экране, поэтому я рекомендую, чтобы вы подобрали подходящее количество строк и придерживались своего выбора. Я также советую не делать поля со списком, работающие в режиме множественного выбора, меньше двух строк в высоту, поскольку некоторые браузеры могут при этом некорректно отображать полосы прокрутки, необходимые для доступа к данным.

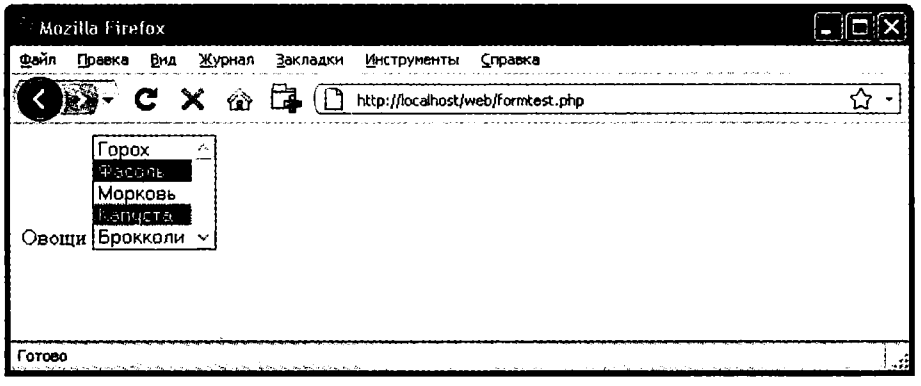



**Рис. 11.7.** Использование поля со списком с параметром multiple

При определении поля со списком, работающего в режиме множественного выбора, можно также воспользоваться параметром selected для задания в случае необходимости более одного заранее выбранного варианта.

## Теги label

За счет использования тегов <label> можно сделать работу пользователя еще удобнее. В эти теги можно заключить элемент формы, обеспечивая его выбор щелчком на любой видимой части, содержащейся между открывающим и закрывающим тегами <label>.

Возвращаясь к примеру выбора времени доставки, можно позволить пользователю щелкать как на самом переключателе, так и на связанном с ним тексте:

```
<label>8.00-12.00<input type="radio" name="time" value="1" /></label>
```

При этом текст не будет подчеркиваться при прохождении над ним указателя мыши, как это происходит с гиперссылкой, но указатель мыши из текстового курсора будет превращаться в стрелку, показывая, что щелкать можно на всем тексте.

### Кнопка отправки

Чтобы согласовать текст на кнопке отправки с разновидностью отправляемой формы, его можно изменить по своему усмотрению, воспользовавшись параметром `value`:

```
<input type="submit" value="Поиск" />
```

Можно также заменить стандартный текст на кнопке выбранным вами графическим изображением, используя следующий код HTML:

```
<input type="image" name="submit" src="image.gif" />
```

## Обезвреживание введенных данных

Вернемся к программированию на PHP. Нелишне будет еще раз напомнить, что обработка данных, введенных пользователем, представляет большую угрозу для безопасности системы, и поэтому очень важно научиться с самого начала работать с этими данными предельно осторожно. На самом деле очистить введенные пользователем данные от потенциальных попыток взлома не так уж сложно, но сделать это совершенно необходимо.

Прежде всего нужно запомнить, что, невзирая на те ограничения, которые были наложены на HTML-форму в отношении типов элементов и размеров вводимых данных, взломщику ничего не стоит воспользоваться свойством браузера, позволяющим просмотреть исходный код страницы, извлечь форму и внести в нее изменения для отправки на ваш веб-сайт вредоносного кода под видом введенных данных.

Поэтому не следует доверять какой-либо переменной, извлеченной из массива `$_GET` или `$_POST`, до тех пор, пока она не пройдет соответствующую обработку. Если эту обработку не провести, пользователи могут предпринять попытку внедрить в данные код JavaScript, мешающий работе ваших сайтов, или даже добавить команды MySQL, подвергающие угрозе содержимое вашей базы данных.

Следовательно, нужно не только считывать введенные пользователем данные с помощью следующего кода:

```
$variable = $_POST['user_input'];
```

но также воспользоваться еще одной или несколькими строками кода. К примеру, чтобы предотвратить внедрение escape-символов в строку, которая будет представлена MySQL, можно применить следующий код (следует напомнить, что эта функция учитывает текущий набор символов, используемый при подключении к MySQL, поэтому она может использоваться только с открытым подключением):

```
$variable = mysql_real_escape_string($variable);
```

Чтобы избавиться от нежелательных слеш-символов, например вставленных с помощью уже устаревшей директивы magic_quotes_gpc, применяется следующий код:

```
$variable = stripslashes($variable);
```

А для удаления из строки любого HTML-кода используется такой код PHP:

```
$variable = htmlentities($variable);
```

Например, этот код интерпретируемого HTML hi заменяется строкой <b>hi</b>, которая отображается как простой текст и не будет интерпретироваться как теги HTML.

И наконец, если нужно полностью очистить введенные данные от HTML, используется следующий код:

```
$variable = strip_tags($variable);
```

А пока вы не решите, какое именно обезвреживание требуется для вашей программы, рассмотрите показанные в примере 11.9 две функции, в которых собраны вместе все эти ограничения, обеспечивающие довольно высокий уровень безопасности.

**Пример 11.9.** Функции sanitizeString и sanitizeMySQL

```
<?php
function sanitizeString($var)
{
    if (get_magic_quotes_gpc()) $var = stripslashes($var);
    $var = htmlentities($var);
    $var = strip_tags($var);
    return $var;
}

function sanitizeMySQL($var)
{
    $var = mysql_real_escape_string($var);
    $var = sanitizeString($var);
    return $var;
}
?>
```

Добавьте этот код в последние строки своих программ, и тогда вы сможете вызвать его для обезвреживания всех вводимых пользователями данных:

```
$variable = sanitizeString($_POST['user_input']);
```

или, если имеется открытое подключение к MySQL:

```
$variable = sanitizeMySQL($_POST['user_input']);
```

# Пример программы

Рассмотрим, как происходит настоящее объединение PHP-программы с HTML-формой, для чего создадим программу convert.php, код которой показан в примере 11.10. Наберите этот код и проверьте его работу.

**Пример 11.10.** Программа перевода значений между шкалами Фаренгейта и Цельсия

```
<?php // convert.php
$f = $c = "";

if (isset($_POST['f'])) $f = sanitizeString($_POST['f']);
if (isset($_POST['c'])) $c = sanitizeString($_POST['c']);

if ($f != '')
{
    $c = intval((5 / 9) * ($f - 32));
    $out = "$f °f равно $c °c";
}
elseif($c != '')
{
    $f = intval((9 / 5) * $c + 32);
    $out = "$c °c равно $f °f";
}
else $out = "";

echo <<<_END
<html><head><title>Программа перевода температуры</title>
</head><body><pre>
Введите температуру по Фаренгейту или по Цельсию и щелкните на кнопке Перевести

<b>$out</b>
<form method="post" action="convert.php">
По Фаренгейту <input type="text" name="f" size="7" />
   По цельсию <input type="text" name="c" size="7" />
           <input type="submit" value="Перевести" />
</form></pre></body></html>
_END;

function sanitizeString($var)
{
    $var = stripslashes($var);
    $var = htmlentities($var);
    $var = strip_tags($var);
    return $var;
}
?>
```

Когда программа convert.php будет вызвана в браузере, результат будет похож на копию экрана, показанную на рис. 11.8.

Если проанализировать эту программу, то в первой строке инициализируются переменные $c $f на тот случай, если их значения не были отправлены программе. В следующих двух строках извлекаются значения либо из поля f, либо из поля c. Эти поля предназначены для ввода значений температуры по Фаренгейту или по Цельсию. Если пользователь введет оба значения, то значение по Цельсию будет проигнорировано, а переведено будет значение по Фаренгейту. В качестве меры безопасности в программе также используется новая функция sanitizeString из примера 11.9.

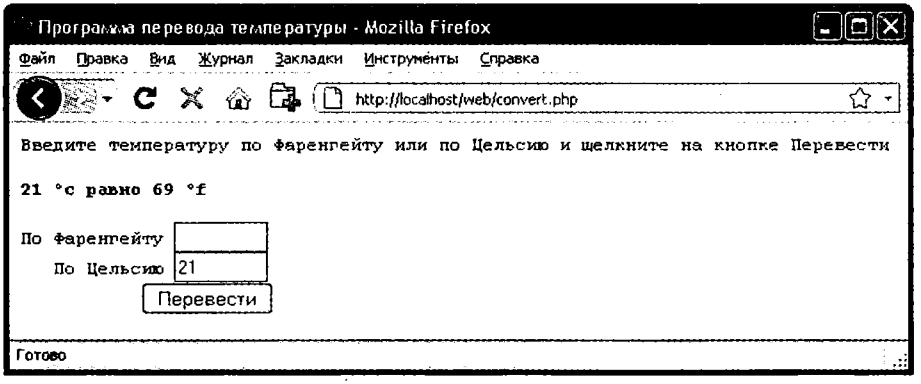



**Рис. 11.8.** Работающая программа перевода температуры

Итак, располагая либо отправленными значениями, либо пустыми строками в обеих переменных `$f` и `$c`, следующая часть кода использует структуру `if... elseif...else`, которая сначала проверяет, имеет ли значение переменная `$f`. Если эта переменная не имеет значения, проверяется переменная `$c`. Если переменная `$c` также не имеет значения, переменной `$out` присваивается пустая строка (к этому месту мы еще вернемся).

При обнаружении того, что у переменной `$f` есть значение, переменной `$c` присваивается простое математическое выражение, которое переводит значение переменной `$f` со значения по Фаренгейту в значение по Цельсию. Для этого используется формула *По_Цельсию = (5 / 9) × (По_Фаренгейту – 32)*. Затем переменной `$out` присваивается строковое значение, в котором содержится сообщение о результатах перевода.

Если же окажется, что у переменной `$f` есть значение, а у переменной `$c` его нет, выполняется обратная операция по переводу значения `$c` из значения по Цельсию в значение по Фаренгейту с присвоением результата переменной `$f`. При этом используется следующая формула: *По_Фаренгейту = (9 / 5) × По_Цельсию + 32*. Как и в предыдущем разделе, переменной `$out` затем присваивается строковое значение, в котором содержится сообщение о результатах перевода.

Для превращения результатов перевода в целое число в обоих переводах вызывается PHP-функция `intval`. В этом нет особой необходимости, но результат выглядит лучше.

Теперь, после выполнения всех арифметических вычислений, программа выдает HTML-код, который начинается с базовых элементов `head` и `title` и содержит вводный текст, предшествующий отображению значения переменной `$out`. Если перевода температуры не осуществлялось, переменная `$out` будет иметь значение `NULL` и выводиться на экран ничего не будет, что, собственно, до тех пор, пока не будут отправлены данные формы, нам и нужно. Но если перевод состоялся, переменная `$out` содержит результат, который отображается на экране.

Затем следует форма, настроенная на отправку данных файлу `convert.php` (то есть самой программе) с использованием метода `POST`. Внутри формы содержатся два поля для ввода температуры как по Фаренгейту, так и по Цельсию. Затем отображается кнопка отправки данных, имеющая надпись **Перевести**, и форма закрывается.

После вывода HTML-кода, закрывающего документ, программа завершается функцией `sanitizeString` из примера 11.9.

Все примеры, показанные в данной главе, используют для отправки данных формы метод POST. Я рекомендую применять именно этот метод как наиболее подходящий и безопасный. Разумеется, формы можно легко перестроить под использование метода GET, тогда значения нужно будет извлекать не из массива $_POST, а из массива $_GET. Причины применения другого метода могут заключаться в предоставлении возможности создания закладок или непосредственных ссылок из другой страницы на результаты поиска.

В следующей главе будет показано, как можно задействовать механизм шаблонов Smarty для реализации среды, позволяющей отделить код вашего приложения от способа представления вашим пользователям всего информационного наполнения.

# Проверьте ваши знания

1. Данные, содержащиеся в форме, могут быть отправлены с использованием одного из двух методов — `POST` или `GET`. Какие ассоциативные массивы используются для передачи этих данных PHP-программе?
2. Что представляет собой свойство `register_globals` и почему его использование не приветствуется?
3. Чем отличаются друг от друга текстовое поле и текстовая область?
4. Если форма предлагает пользователю три варианта выбора, каждый из которых исключает все оставшиеся (то есть выбран может быть только один из вариантов), то какой тип элемента ввода данных нужно использовать в этом случае, если есть выбор между флажками и переключателями?
5. Как из веб-формы отправить группу значений из поля со списком, используя только одно имя поля?
6. Как отправить данные поля формы, не отображая их на экране браузера?
7. В какой HTML-тег можно поместить элемент формы, чтобы весь его текст или изображение превратились в область выбора этого элемента по щелчку кнопкой мыши?
8. Какая PHP-функция используется для преобразования кода HTML в формат, который может быть отображен на экране, но не может интерпретироваться браузером в качестве кода HTML?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 11».

# 12 Cookie, сессии и аутентификация

По мере укрупнения и усложнения вашего проекта будут возрастать и потребности в учете его пользователей. Даже если не предлагается ввод имени пользователя и пароля, то довольно часто возникает необходимость в хранении сведений о ходе текущей сессии пользователя и, возможно, в том, чтобы узнать его при возвращении на ваш сайт.

Подобное взаимодействие с пользователем поддерживается с помощью нескольких технологий, от простых браузерных cookie до обработки сессий и HTTP-аутентификации. В совокупности они позволяют настроить сайт на пользовательские предпочтения, обеспечивая комфортное путешествие по его страницам.

## Использование cookie в PHP

Cookie представляет собой элемент данных, который веб-сервер с помощью браузера сохраняет на жестком диске вашего компьютера. Этот элемент может содержать практически любую буквенно-цифровую информацию (объемом не более 4 Кбайт) и может быть извлечен из вашего компьютера и возвращен на сервер. Чаще всего cookie используются для отслеживания хода сессий, обобщения данных нескольких визитов, хранения содержимого корзины покупателя, хранения сведений, необходимых для входа в систему, и т. д.

В силу своей закрытости cookie могут быть считаны только из создавшего их домена. Иными словами, если cookie, к примеру, был создан на oreilly.com, он может быть извлечен только веб-сервером, использующим этот домен. Это не позволяет другим веб-сайтам получить доступ к сведениям, на владение которыми у них нет разрешения.

Из-за особенностей работы Интернета многие элементы веб-страницы могут быть вставлены из нескольких доменов, каждый из которых может создавать свои собственные cookie. Они называются *сторонними* cookie. Чаще всего они создаются рекламными компаниями с целью отслеживания пользователей на нескольких веб-сайтах.

Большинство веб-браузеров позволяют пользователями отключать cookie либо от домена текущего сервера, либо от сторонних серверов, либо и от тех и от других. К счастью, большинство пользователей, отключающих cookie, делают это только в отношении сторонних веб-сайтов.

Обмен cookie осуществляется во время передачи заголовков еще до того, как будет отправлен код HTML веб-страницы. Отправить cookie после передачи HTML-кода уже невозможно. Поэтому четкое планирование использования cookie приобретает особую важность. На рис. 12.1 показан типичный диалог с передачей cookie в форме «запрос — ответ» между веб-браузером и веб-сервером.

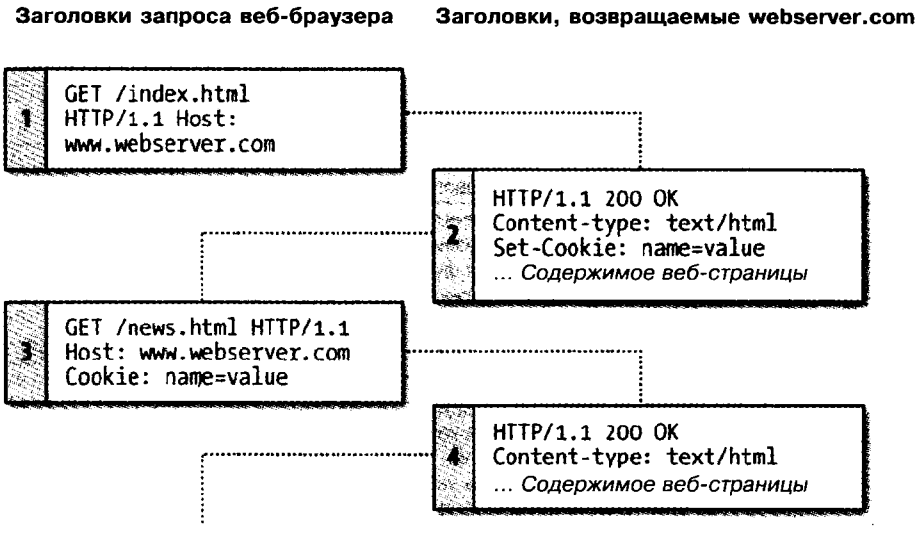



**Рис. 12.1.** Диалог браузера и сервера в режиме «запрос — ответ» с использованием cookie

В этом обмене данными показан браузер, получающий две страницы.

1. Браузер выдает запрос на извлечение главной страницы index.html с веб-сайта http://www.webserver.com. В первом заголовке указывается файл, а во втором — сервер.
2. Когда веб-сервер на webserver.com получает эту пару заголовков, он возвращает несколько своих заголовков. Во втором заголовке определяется тип отправляемого содержимого (text/html), а в третьем отправляется cookie по имени *name*, имеющий значение *value*. И только после этого передается содержимое веб-страницы.
3. После того как браузер получит cookie, он должен возвращать его с каждым последующим запросом, сделанным в адрес сервера, создавшего cookie, пока у cookie не истечет срок действия или этот cookie не будет удален. Поэтому, когда браузер запрашивает новую страницу /news.html, он также возвращает cookie *name* со значением *value*.
4. Поскольку на момент отправки /news.html cookie уже был установлен, сервер не должен заново посылать этот cookie и возвращает только запрошенную страницу.

# Установка cookie

Установка cookie в PHP осуществляется довольно просто. До передачи кода HTML нужно вызвать функцию `setcookie`, для чего используется следующий синтаксис (табл. 12.1):

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

**Таблица 12.1.** Параметры функции setcookie

| Параметр | Описание | Пример |
|---|---|---|
| name | Имя cookie. Это имя ваш сервер будет использовать для доступа к cookie при последующих запросах браузера | username |
| value | Значение cookie или его содержимое. Объем может составлять до 4 Кбайт буквенно-цифрового текста | Hannah |
| expire | (Необязательный.) Время истечения срока действия в формате метки времени Unix. Как правило, для установки этого параметра будет использоваться функция time(), к которой будет прибавляться количество секунд. Если параметр не установлен, срок действия cookie заканчивается с закрытием браузера | time() + 2592000 |
| path | (Необязательный.) Путь к cookie на сервере. Если в качестве пути используется прямой слеш (/), cookie доступен для всего домена, например для домена www.webserver.com. Если указан подкаталог, cookie доступен только в пределах этого подкаталога. По умолчанию путь указывает на текущий каталог, где был установлен cookie, и, как правило, используется именно такая настройка | / |
| domain | (Необязательный.) Интернет-домен, которому принадлежит cookie. Если это webserver.com, то cookie доступен для всего домена webserver.com и его поддоменов, например www.webserver.com и для images.webserver.com. Если это images.webserver.com, то cookie доступен только для images.webserver.com и его поддоменов, например sub.images.webserver.com, но не для www.webserver.com | .webserver.com |
| secure | (Необязательный.) Определяет, должен ли cookie использовать безопасное подключение (https://). Если значение параметра установлено в TRUE, cookie может быть передан только по безопасному подключению. По умолчанию устанавливается значение FALSE | FALSE |
| httponly | (Необязательный; реализован в PHP, начиная с версии 5.2.0.) Определяет, должен ли cookie использовать протокол HTTP. Если значение параметра установлено в TRUE, то такие языки сценариев, как JavaScript, не могут получить доступ к cookie. (Это свойство поддерживается не во всех браузерах). По умолчанию задается значение FALSE | FALSE |

Для создания cookie-файла по имени `username` со значением `Hannah`, к которому имеется доступ со всего веб-сервера текущего домена и который будет удален из браузерного кэша через семь дней, используется следующая строка кода:

```
setcookie('username', 'Hannah', time() + 60 * 60 * 24 * 7, '/');
```

## Доступ к cookie

Для чтения значения cookie нужно просто обратиться к системному массиву $_COOKIE. Например, если нужно посмотреть, хранится ли на текущем браузере cookie по имени username, и, если хранится, прочитать его значение, используется следующая строка кода:

```
if (isset($_COOKIE['username']))
    $username = $_COOKIE['username'];
```

Учтите, что прочитать значение cookie можно только после того, как он был отправлен веб-браузеру. Это означает, что при установке cookie его нельзя прочитать до тех пор, пока браузер не перезагрузит страницу (или не совершит какое-нибудь другое действие с доступом к cookie) с вашего веб-сайта и не передаст cookie в ходе этого процесса обратно на сервер.

## Удаление cookie

Для удаления cookie его нужно повторно установить с настройкой даты истечения срока действия на прошедшее время. При этом важно, чтобы все параметры нового вызова функции setcookie, за исключением timestamp, точно повторяли параметры, указываемые при создании cookie, в противном случае удаление не состоится. Поэтому для удаления ранее созданного cookie нужно воспользоваться следующей строкой кода:

```
setcookie('username', 'Hannah', time() - 2592000, '/');
```

Поскольку указано уже прошедшее время, cookie будет удален. Здесь я использовал время, равное 2 592 000 с в прошлом (что соответствует одному месяцу). Это сделано в расчете на неправильную установку даты и времени на компьютере клиента.

# HTTP-аутентификация

HTTP-аутентификация использует веб-сервер для управления пользователями и паролями при работе приложения. Ее можно применять во многих приложениях, требующих от пользователей входа в приложение, хотя для некоторых приложений нужны особые меры или соблюдение более строгих требований безопасности, для чего следует обратиться к другим технологическим приемам.

Чтобы воспользоваться HTTP-аутентификацией, PHP отправляет заголовок запроса, инициирующий аутентификационный диалог с браузером. Чтобы эта технология заработала, на сервере должно быть включено соответствующее свойство, но, скорее всего, в силу востребованности это свойство на вашем сервере уже включено.

Хотя, как правило, HTTP-аутентификация устанавливается вместе с Apache, это еще не означает, что она установлена на используемом вами сервере. Поэтому при попытке запуска представленных здесь примеров может быть сгенерирована ошибка и выдано сообщение о том, что это свойство недоступно. В таком случае нужно установить соответствующий модуль, изменить для загрузки модуля файл конфигурации или попросить своего системного администратора внести все эти изменения.

Пользователи при вводе в браузер URL-адреса или при переходе по ссылке видят окно с требованием пройти аутентификацию Требуется аутентификация, в котором выводится приглашение заполнить два поля: для имени пользователя и пароля (на рис. 12.2 показано, как это выглядит в браузере Firefox).

Код, обеспечивающий аутентификацию, показан в примере 12.1.

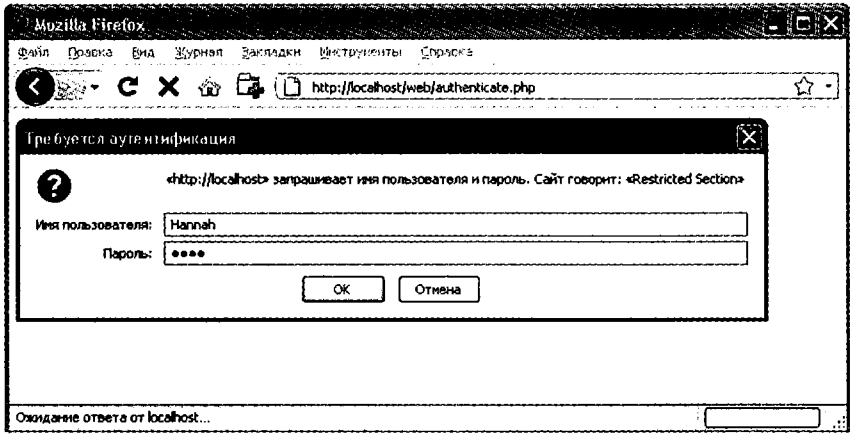



**Рис. 12.2.** Приглашение войти в систему в режиме HTTP-аутентификации

**Пример 12.1.** PHP-аутентификация

```
<?php
if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    echo "Добро пожаловать, пользователь: " . $_SERVER['PHP_AUTH_USER'] .
        " , имеющий пароль: " . $_SERVER['PHP_AUTH_PW'];
}
else
{
    header('WWW-Authenticate: Basic realm="Restricted Section"');
    header('HTTP/1.0 401 Unauthorized');
    die("Пожалуйста, введите имя пользователя и пароль");
}
?>
```

Сначала программа ищет значения конкретных элементов массива: `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']`. В этих элементах в случае их существования содержатся имя пользователя и пароль, введенные пользователем при приглашении пройти аутентификацию.

Если какое-нибудь из значений отсутствует, значит, пользователь еще не прошел аутентификацию, и появляющееся окно с приглашением, показанное на рис.12.2, отображается с выдачей следующего заголовка, составной частью которого является имя защищенного раздела — `Basic realm`:

```
WWW-Authenticate: Basic realm="Restricted Section"
```

Если пользователь заполнит поля, PHP-программа будет запущена снова с самого начала. Но если пользователь щелкнет на кнопке Отмена, программа перейдет к следующим двум строкам, которые отправят следующий заголовок и сообщение об ошибке:

```
HTTP/1.0 401 Unauthorized
```

Инструкция die выведет следующий текст: «Пожалуйста, введите имя пользователя и пароль» (рис. 12.3).

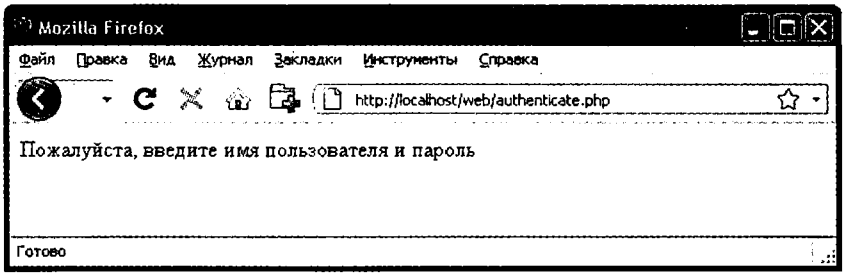



**Рис. 12.3.** Результат щелчка на кнопке Отмена

После прохождения пользователем аутентификации заставить появиться диалоговое окно для аутентификации уже не удастся до тех пор, пока пользователь не закроет и не откроет снова все окна браузера, поскольку веб-браузер будет постоянно возвращать PHP-программе имя пользователя и пароль. Чтобы при изучении этого раздела испытать все возможные режимы работы, может потребоваться несколько раз закрыть и снова открыть ваш браузер.

Теперь проверим допустимость пользовательского имени и пароля. Для добавления этой проверки вносить большие изменения в код примера 12.1 не придется: нам нужно лишь изменить прежнее приветственное сообщение и превратить его в проверку правильности имени пользователя и пароля, за которой последует приветствие. Неудачная аутентификация вызовет отправку сообщения об ошибке (пример 12.2).

**Пример 12.2.** PHP-аутентификация с проверкой вводимой информации

```
<?php
$username = 'admin';
$password = 'letmein';

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    if ($_SERVER['PHP_AUTH_USER'] == $username &&
        $_SERVER['PHP_AUTH_PW'] == $password)
        echo "Регистрация прошла успешно";
    else die("Неверная комбинация "имя пользователя – пароль"");
}
else
```

```
{
    header('WWW-Authenticate: Basic realm="Restricted Section"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
}
?>
```

Кстати, обратите внимание на формулировку сообщения об ошибке: «Неверная комбинация "имя пользователя — пароль"». В ней не сообщается, что именно было введено неправильно: имя пользователя, пароль или же и то и другое — чем меньше информации попадет к потенциальному взломщику, тем лучше.

Теперь у нас есть механизм аутентификации пользователей, но только для одного имени пользователя и пароля. К тому же пароль появляется в файле PHP в виде простого текста, и если кому-нибудь удастся взломать ваш сервер, он тут же вскроет и пароль. Поэтому рассмотрим более подходящий способ работы с именами пользователей и паролями.

Безопасность в современных браузерах становится все строже и достигла такого уровня, при котором проверить HTTP-аутентификацию на локальной файловой системе, пока не будут изменены настройки вашего браузера, не так-то просто. Это связано с необходимостью защиты от потенциально вредоносных файлов, которые могут быть скачаны из Интернета (поскольку локальные файлы обычно представляют собой больший риск с точки зрения безопасности). Если нужно написать код, использующий этот тип аутентификации, то лучше провести тестирование на удаленном сервере, используя интернет-соединение.

## Сохранение имен пользователей и паролей

Безусловно, самым естественным способом сохранения имен пользователей и паролей будет задействование MySQL. Но опять-таки хранить пароли в виде простого текста не хочется, поскольку, если база данных будет взломана, наш веб-сайт подвергнется опасности. Вместо этого будет использован тонкий прием с применением так называемой *односторонней функции*.

Функции этого типа просты в использовании и способны превращать строку текста в строку, напоминающую набор произвольных символов. Односторонние функции практически невозможно применять в обратном направлении, поэтому производимая ими выходная информация может безопасно храниться в базе данных и ее похититель ничего не узнает об используемых паролях.

Задействуемая нами функция называется `md5`. Ей передается строка для хеширования, а она возвращает 32-символьное шестнадцатеричное число. Код, в котором она используется, имеет следующий вид:

```
$token = md5('мой_пароль');
```

При запуске этого кода переменная `$token` может получить следующее значение:

```
34819d7beeabb9260a5c854bc85b3e44
```

Можно также воспользоваться схожей функцией sha1, которая считается более безопасной, имеет более удачный алгоритм и возвращает 40-символьное шестнадцатеричное число.

## Добавление произвольных данных

К сожалению, применения одной только функции md5 для защиты базы данных с паролями недостаточно, поскольку пароль можно вскрыть путем перебора, в котором используется другая база данных, состоящая из известных 32-символьных шестнадцатеричных лексем, выдаваемых функцией md5. В существовании подобных баз данных можно убедиться, воспользовавшись поисковой системой Google.

К счастью, любой поисковой атаке можно поставить заслон путем добавления произвольных данных ко всем паролям перед их отправкой функции md5. Этот прием, ассоциирующийся с «посыпанием солью» (salting), заключается в простом добавлении некоего известного только нам текста к любому кодируемому параметру:

```
$token = md5('Дополнительная_строка_мой_пароль');
```

В данном примере перед паролем был набран текст Дополнительная_строка. Конечно, чем малоизвестнее будет эта дополнительная строка, тем лучше. Я предпочитаю добавлять строку, похожую на эту:

```
$token = md5('hqb%$tмой_парольcg*l');
```

Здесь произвольные символы помещены как до, так и после пароля. На основе лишь тех данных, которые хранятся в базе, не имея доступа к вашему PHP-коду, вскрыть хранящиеся пароли практически невозможно.

Учитывая то, как растет скорость компьютерной обработки, MD5-строки, использующие короткие исходные строки, могут быть потенциально взломаны в течение недель (а не лет). Поэтому был разработан алгоритм SHA1, взломать который значительно сложнее, чем MD5, и который возвращает шестнадцатеричную строку, состоящую из 40 символов.

Чтобы защитить ваш код от будущих перемен, может потребоваться вместо функции md5 воспользоваться PHP-функцией sha1. (Если хранить значения SHA1 в MySQL, нужно обязательно установить длину поля равной 40 символам.) Или же, если требуется более стойкое кодирование, я рекомендую обратить внимание на PHP-функцию crypt, использующую алгоритм CRYPT_BLOWFISH, описание которой дано на веб-странице http://tinyurl.com/phpcrypt.

Для проверки принадлежащего кому-нибудь пароля при входе на сайт нужно будет лишь добавить те же самые произвольные строки в начало и конец пароля, а затем сравнить полученную от функции md5 лексему с той, которая была сохранена в базе данных для данного пользователя.

Создадим таблицу MySQL, в которой будут храниться сведения о пользователе, и добавим к ней пару учетных записей. Наберите код, показанный в примере 12.3, и сохраните его в файле setupusers.php, после чего откройте этот файл в своем браузере.

**Пример 12.3.** Создание таблицы users и добавление к ней двух учетных записей

```
<?php // setupusers.php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

$query = "CREATE TABLE users (
            forename VARCHAR(32) NOT NULL,
            surname VARCHAR(32) NOT NULL,
            username VARCHAR(32) NOT NULL UNIQUE,
            password VARCHAR(32) NOT NULL
            )";

$result = mysql_query($query);
if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());

$salt1 = "qm&h*";
$salt2 = "pg!@";

$forename = 'Bill';
$surname = 'Smith';
$username = 'bsmith';
$password = 'mysecret';
$token = md5("$salt1$password$salt2");
add_user($forename, $surname, $username, $token);

$forename = 'Pauline';
$surname = 'Jones';
$username = 'pjones';
$password = 'acrobat';
$token = md5("$salt1$password$salt2");
add_user($forename, $surname, $username, $token);

function add_user($fn, $sn, $un, $pw)
{
    $query = "INSERT INTO users VALUES('$fn', '$sn', '$un', '$pw')";
    $result = mysql_query($query);
    if (!$result) die ("Сбой при доступе к базе данных: " . mysql_error());
}
?>
```

Эта программа создаст в вашей базе данных publications (или в той базе данных, на которую вы настроились в файле login.php в главе 10) таблицу users. В этой таблице будут созданы два пользователя, Bill Smith и Pauline Jones, с именами пользователей и паролями bsmith — mysecret и pjones — acrobat соответственно.

Теперь, используя данные, имеющиеся в этой таблице, мы можем модифицировать код примера 12.2 для вполне приемлемой аутентификации пользователей.

Необходимый для этого код показан в примере 12.4. Наберите этот код, сохраните его в файле authenticate.php и вызовите эту программу в своем браузере.

Пример 12.4. PHP-аутентификация с использованием MySQL

```
<?php // authenticate.php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    $un_temp = mysql_entities_fix_string($_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($_SERVER['PHP_AUTH_PW']);

    $query = "SELECT * FROM users WHERE username='$un_temp'";
    $result = mysql_query($query);
    if (!$result) die("Сбой при доступе к базе данных: "   mysql_error());
    elseif (mysql_num_rows($result))
    {
        $row = mysql_fetch_row($result);
        $salt1 = "qm&h*";
        $salt2 = "pg!@";
        $token = md5("$salt1$pw_temp$salt2");

        if ($token == $row[3]) echo "$row[0] $row[1] :
          Привет, $row[0], теперь вы зарегистрированы под именем '$row[2]'";
        else die("Неверная комбинация "имя пользователя – пароль"");
    }
    else die("Неверная комбинация имя пользователя-пароль");
}
else
{
    header('WWW-Authenticate: Basic realm="Restricted Section"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
}

function mysql_entities_fix_string($string)
{
    return htmlentities(mysql_fix_string($string));
}

function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return mysql_real_escape_string($string);
}
?>
```

Вы, наверное, ожидали, что настало время для некоторых весьма больших по объему примеров кода (таких как этот пример). Но по этому поводу не стоит переживать. Последние десять строк — не что иное, как пример 10.22, который уже встречался в главе 10. Эти строки присутствуют здесь для выполнения очень важной задачи — обезвреживания введенных пользователем данных.

На данный момент практический интерес для вас должны представлять только те строки, которые выделены жирным шрифтом и начинаются с присваивания значений двум переменным, `$un_temp` и `$pw_temp`, с использованием отправленных имени пользователя и пароля. Затем выдается запрос к MySQL на поиск пользователя с именем `$un_temp`, и, если будет возвращен результат, значение его первой строки присваивается переменной `$row`. (Поскольку имя пользователя уникально, возвращена будет только одна строка.) Потом создаются переменные `$salt1` и `$salt2` с двумя произвольными строками, которые затем добавляются до и после отправленного пароля `$pw_temp`. После этого получившаяся в результате строка передается функции `md5`, которая возвращает 32-символьное шестнадцатеричное значение, присваиваемое переменной `$token`.

Теперь остается лишь сравнить значение переменной `$token` со значением, хранящимся в базе данных в четвертой графе, которая при начале отсчета с нуля соответствует графе 3. То есть `$row[3]` содержит предыдущую лексему, вычисленную для «посыпанного солью» пароля. Если значения совпадают, выдается строка приветствия, в которой содержится обращение к пользователю по его настоящему имени (рис. 12.4). В противном случае выдается сообщение об ошибке. Как уже упоминалось, это сообщение всегда содержит одну и ту же информацию независимо от того, существует такое имя пользователя или нет, поскольку потенциальный взломщик или тот, кто подбирает пароли, получает в результате этого минимум полезной для себя информации.

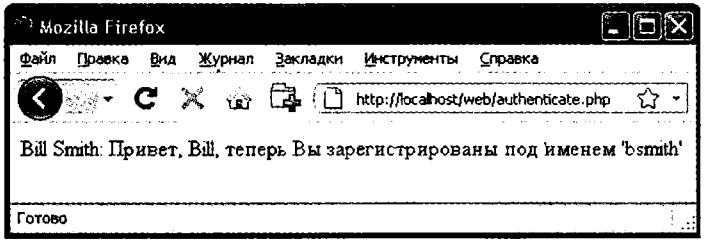



**Рис. 12.4.** Аутентификация пользователя Bill Smith прошла успешно

Вы можете самостоятельно испытать эту программу в работе, вызвав ее в браузере и набрав имя пользователя `bsmith` и пароль `mysecret` (или набрав пару `pjones` и `acrobat`), то есть те значения, которые были сохранены в базе данных программой из примера 12.3.

# Использование сессий

Поскольку ваша программа не может сообщить о том, какие значения были присвоены переменным в других программах, или даже о том, какие значения были

присвоены переменным при ее предыдущем запуске, иногда возникает потребность в отслеживании действий пользователя, переходящего со страницы на страницу. Это можно сделать за счет установки в форме скрытых полей, как было показано в главе 10, и проверки значений этих полей после отправки формы, но PHP предоставляет более простое и действенное решение — *сессии*. Сессии представляют собой группы переменных, которые хранятся на сервере, но относятся только к текущему пользователю. Чтобы обеспечить обращение нужных пользователей к нужным переменным, для уникальной идентификации этих пользователей их веб-браузерами сохраняются файлы cookie.

Эти cookie имеют значение только для веб-сервера и не могут быть использованы для извлечения какой-либо информации о пользователе. Вы можете спросить о том, как быть с теми пользователями, которые отключили cookies. Это не проблема, поскольку в PHP, начиная с версии 4.2.0, такие случаи выявляются и cookie помещаются в область GET-запроса каждого URL-адреса. В любом случае сессии предоставляют надежный способ отслеживания действий ваших пользователей.

## Начало сессии

Чтобы инициировать работу сессии, нужно перед выводом на экран любого кода HTML вызвать PHP-функцию session_start точно так же, как это делалось при отправке cookie в процессе обмена заголовками. Затем, чтобы приступить к сохранению переменных сессии, им нужно присвоить значения как элементам массива $_SESSION:

```
$_SESSION['имя_переменной'] = $переменная_со_значением;
```

При последующих запусках программы их значения можно будет снова прочитать, воспользовавшись следующим кодом:

```
$имя_переменной = $_SESSION['имя_переменной'];
```

Предположим, у вас есть приложение, которому всегда нужен доступ к пользовательскому имени и паролю, а также к настоящему имени и фамилии каждого пользователя в том виде, в котором они сохранены в базе данных в таблице users, созданной совсем недавно. Выполним еще одну модификацию программы authenticate.php из примера 12.4, чтобы инициировать работу сессии сразу же после идентификации пользователя.

Все необходимые изменения показаны в примере 12.5. Единственное отличие касается раздела if ($token == $row[3]), который теперь начинается с открытия сессии и сохранения в ней четырех переменных. Наберите код этой программы (или измените код примера 12.4) и сохраните его в файле authenticate2.php. Но пока не запускайте эту программу в браузере, поскольку нужно будет создать еще и вторую программу.

**Пример 12.5.** Открытие сессии после успешной аутентификации

```
<?php //authenticate2.php
require_once 'login.php';
$db_server = mysql_connect($db_hostname, $db_username, $db_password);
```

```
if (!$db_server) die("Невозможно подключиться к MySQL: " . mysql_error());
mysql_select_db($db_database)
    or die("Невозможно выбрать базу данных: " . mysql_error());

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    $un_temp = mysql_entities_fix_string($_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($_SERVER['PHP_AUTH_PW']);

    $query = "SELECT * FROM users WHERE username='$un_temp'";
    $result = mysql_query($query);
    if (!$result) die("Сбой при доступе к базе данных: " . mysql_error());
    elseif (mysql_num_rows($result))
    {
        $row = mysql_fetch_row($result);
        $salt1 = "qm&h*";
        $salt2 = "pg!@";
        $token = md5("$salt1$pw_temp$salt2");

        if ($token == $row[3])
        {
            session_start();
            $_SESSION['username'] = $un_temp;
            $_SESSION['password'] = $pw_temp;
            $_SESSION['forename'] = $row[0];
            $_SESSION['surname'] = $row[1];
            echo "$row[0] $row[1] : Привет, $row[0],
                теперь вы зарегистрированы под именем '$row[2]'";
            die ("<p><a href=continue.php>Щелкните здесь
                для продолжения</a></p>");
        }
        else die("Неверная комбинация "имя пользователя – пароль"");
    }
    else die("Неверная комбинация "имя пользователя – пароль"");
}
else
{
    header('WWW-Authenticate: Basic realm="Restricted Section"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Пожалуйста, введите имя пользователя и пароль");
}

function mysql_entities_fix_string($string)
{
    return htmlentities(mysql_fix_string($string));
}

function mysql_fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
```

```
    return mysql_real_escape_string($string);
}
?>
```

К программе также добавлена ссылка Щелкните здесь для продолжения с URL-адресом continue.php. Она будет использована для иллюстрации того, как сессия будет перенесена на другую программу или веб-страницу PHP-программы. Поэтому создайте файл continue.php, набрав и сохранив в нем программу из примера 12.6.

**Пример 12.6.** Извлечение переменных сессии

```
<?php // continue.php
session_start();

if (isset($_SESSION['username']))
{
    $username = $_SESSION['username'];
    $password = $_SESSION['password'];
    $forename = $_SESSION['forename'];
    $surname = $_SESSION['surname'];

    echo "С возвращением, $forename.<br />
        Ваше полное имя $forename $surname.<br />
        Ваше имя пользователя '$username'
        и Ваш пароль '$password'.";
}
else echo "Пожалуйста, для входа <a href=authenticate2.php>щелкните здесь</a>.";
?>
```

Теперь можно вызвать в браузере authenticate2.php, после появления приглашения ввести имя пользователя bsmith и пароль mysecret (или pjones и acrobat) и щелкнуть на ссылке для загрузки программы continue.php. Когда браузер вызовет эту программу, появится результат, аналогичный показанному на рис. 12.5.

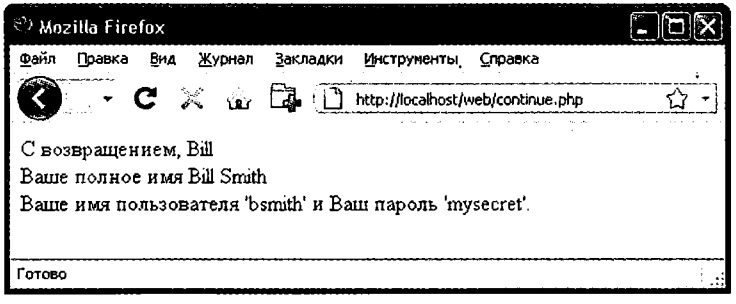



**Рис. 12.5.** Поддержка пользовательских данных с помощью сессий

Сессии искусно ограничивают одной программой весь объемный код, необходимый для аутентификации и регистрации пользователя. После аутентификации пользователя и создания сессии весь остальной программный код действительно упрощается. Нужно лишь вызвать функцию session_start и найти любые переменные, к которым нужен доступ из массива $_SESSION.

В примере 12.6 быстрой проверки наличия значения у элемента $_SESSION['username'] вполне достаточно для того, чтобы узнать об аутентификации текущего пользователя, потому что переменные сессии хранятся на сервере (в отличие от cookie, которые хранятся на машине веб-браузера) и им можно доверять.

Если элементу $_SESSION['username'] значение присвоено не было, значит, активная сессия отсутствует, и поэтому последняя строка кода в примере 12.6 перенаправляет пользователей на страницу регистрации на сайте, которая находится в программе authenticate2.php.

Программа continue.php выводит значение пользовательского пароля, чтобы показать, как работают переменные сессии. На практике вам будет известно, что пользователь уже зарегистрировался, поэтому отслеживать (или выводить) любые пароли нет необходимости, тем более что подобные действия угрожают безопасности системы.

## Завершение сессии

Обычно когда пользователю нужно уйти с вашего веб-сайта, наступает момент завершения работы сессии, для чего, как показано в примере 12.7, вместе с функцией unset можно воспользоваться функцией session_destroy. В этом примере предоставляется полезная функция для полного уничтожения сессии, выхода пользователя и очистки всех переменных сессии.

**Пример 12.7.** Полезная функция уничтожения сессии и ее данных

```
<?php
function destroy_session_and_data()
{
    session_start();
    $_SESSION = array();
    if (session_id() != "" || isset($_COOKIE[session_name()]))
        setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
}
?>
```

Чтобы увидеть этот код в действии, можно модифицировать программу continue.php, как показано в примере 12.8.

**Пример 12.8.** Извлечение переменных сессии перед ее уничтожением

```
<?php
session_start();

if (isset($_SESSION['username']))
{
    $username = $_SESSION['username'];
    $password = $_SESSION['password'];
    $forename = $_SESSION['forename'];
    $surname = $_SESSION['surname'];

    destroy_session_and_data();
```

```
    echo "С возвращением, $forename.<br />
        Ваше полное имя $forename $surname.<br />
        Ваше имя пользователя '$username'
        и ваш пароль '$password'.";

}
else echo "Пожалуйста, для входа <a href=authenticate2.php>щелкните здесь</a>.";

function destroy_session_and_data()
{
    $_SESSION = array();
    if (session_id() != "" || isset($_COOKIE[session_name()]))
        setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
}
?>
```

При первом переходе из `authenticate2.php` в `continue.php` будут выведены все переменные сессии. Но если после этого щелкнуть в браузере на кнопке обновления страницы, в результате предшествующего этому вызова функции `destroy_session_and_data` сессия уже будет уничтожена и будет выведено приглашение вернуться на страницу регистрации.

## Установка времени ожидания

Есть и другие причины, по которым может потребоваться самостоятельное закрытие пользовательской сессии, например, если пользователь забыл зарегистрироваться или проигнорировал этот процесс и нужно, чтобы программа закрыла его сессию в целях собственной безопасности. Это можно сделать, установив время ожидания, по истечении которого, если не предпринять активных действий, произойдет автоматическое завершение работы.

Для этого используется функция `ini_set`. В данном примере время ожидания устанавливается ровно на сутки:

```
ini_set('session.gc_maxlifetime', 60 * 60 * 24);
```

Если нужно узнать текущее время ожидания, его можно отобразить, воспользовавшись следующим кодом:

```
echo ini_get('session.gc_maxlifetime');
```

# Безопасность сессии

Все мои прежние заверения в том, что после аутентификации пользователя и начала сессии можно спокойно предположить, что переменные сессии заслуживают доверия, не вполне соответствуют действительности. Дело в том, что для вскрытия идентификаторов сессий, передаваемых по сети, можно организовать анализ пакетов — *packet sniffing* (перехват набора данных). Кроме того, если идентификатор (ID) сессии передается в области `GET`-запроса URL-адреса, он может появиться в файлах

регистрации внешних сайтов. Единственный по-настоящему безопасный способ предотвращения вскрытия заключается в использовании протокола защищенный сокетов — Secure Socket Layer (SSL) и в запуске веб-страниц, использующих вместо протокола HTTP протокол HTTPS. Эта тема выходит за рамки данной книги, но за подробностями настроек безопасности веб-сервера можно обратиться по адресу http://www.apache-ssl.org.

## Предупреждение хищения сессии

Когда применение SSL не представляется возможным, можно продолжить аутентификацию пользователей за счет хранения наряду с остальными сведениями их IP-адресов. Для этого нужно при сохранении их сессии добавить следующую строку кода:

```
$_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
```

Затем в качестве дополнительной меры контроля при любой загрузке страницы и доступности сессии проводится следующая проверка, которая при несоответствии текущего IP-адреса сохраненному вызывает функцию different_user:

```
if ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR']) different_user();
```

Какой код будет у функции different_user — решать вам, но я рекомендую просто удалить текущую сессию и попросить пользователя пройти повторную регистрацию вследствие технической ошибки. Больше ни о чем сообщать не нужно, иначе произойдет утечка потенциально ценной информации.

Разумеется, нужно принимать в расчет, что пользователи, работающие через один и тот же прокси-сервер или использующие одинаковые общие IP-адреса в домашней или офисной сети, будут иметь один и тот же IP-адрес. Если это вызовет проблему, нужно опять обратиться к протоколу SSL. Можно также сохранить копию браузерной строки агента пользователя (той самой строки, которую разработчики помещают в свои браузеры, для того чтобы идентифицировать их по типу и версии), с помощью которой также возможно отличить пользователей друг от друга благодаря существованию широкого выбора типов, версий и компьютерных платформ. Для сохранения агента пользователя можно воспользоваться следующим кодом:

```
$_SESSION['ua'] = $_SERVER['HTTP_USER_AGENT'];
```

А для сравнения текущей строки агента с сохраненной можно воспользоваться следующим кодом:

```
if ($_SESSION['ua'] != $_SERVER['HTTP_USER_AGENT']) different_user();
```

Или, что еще лучше, можно объединить эти две проверки и сохранить их комбинацию в виде шестнадцатеричной строки, получаемой от функции md5:

```
$_SESSION['check'] = md5($_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT']);
```

Затем для сравнения текущей и сохраненной строк можно применить такой код:

```
if ($_SESSION['check'] != md5($_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT'])) different_user();
```

## Предотвращение фиксации сессии

При фиксации сессии злоумышленник пытается навязать серверу ее идентификационный номер (ID), не позволяя ему самостоятельно присвоить этот номер. Это происходит в том случае, если пользователь злоупотребляет возможностью передачи ID сессии в области GET-запроса URL-адреса:

```
http://yourserver.com/authenticate.php?PHPSESSID=123456789
```

В данном случае серверу будет передан вымышленный ID сессии 123456789. А теперь рассмотрим пример 12.9, код которого восприимчив к фиксации сессии. Чтобы увидеть его в действии, наберите текст примера и сохраните его в файле sessiontest.php.

**Пример 12.9.** Сессия, восприимчивая к фиксации сессии

```
<?php // sessiontest.php
session_start();

if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
else ++$_SESSION['count'];
echo $_SESSION['count'];
?>
```

После того как код будет сохранен, вызовите программу в вашем браузере, используя следующий URL (предваряя его правильным путевым именем, например http://localhost/web/):

```
sessiontest.php?PHPSESSID=1234
```

Через некоторое время, щелкнув на кнопке обновления страницы, вы увидите увеличение значения счетчика. Теперь попробуйте ввести в браузер следующий URL-адрес:

```
sessiontest.php?PHPSESSID=5678
```

Щелкните несколько раз на кнопке обновления страницы и увидите, что счет опять начался с нуля. Оставьте показания счетчика на номере, отличающемся от его показаний при использовании первого URL-адреса, вернитесь на первый URL-адрес и посмотрите на то, как показания счетчика вернулись к первоначальному значению. Вы по собственному усмотрению создали две разные сессии и без особого труда можете создать их в любом количестве.

Особая опасность такого подхода состоит в том, что затеявший атаку злоумышленник может попытаться распространить такие URL-адреса среди ничего не подозревающих пользователей, и если кто-нибудь из них перейдет по ссылкам с этими адресами, атакующий сможет вернуться и перехватить любую сессию, которая не была удалена или срок действия которой еще не истек!

Для предотвращения фиксации сессии нужно для изменения ID сессии воспользоваться функцией session_regenerate_id. Эта функция сохраняет значения всех переменных текущей сессии, но заменяет ID сессии новым, о котором не может знать атакующий.

Теперь при получении запроса можно проверить факт существования специальной переменной сессии, которую вы выдумали произвольным образом. Если

ее не существует, вы будете знать, что создана новая сессия, поэтому вы просто меняете ID сессии и устанавливаете значение ее специальной переменной, позволяющей заметить изменение.

В примере 12.10 показано, как может выглядеть код, использующий переменную сессии `initiated`.

**Пример 12.10.** Регенерация сессии

```
<?php
session_start();

if (!isset($_SESSION['initiated']))
{
    session_regenerate_id();
    $_SESSION['initiated'] = 1;
}

if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
else ++$_SESSION['count'];
echo $_SESSION['count'];
?>
```

Атакующий может вернуться на ваш сайт, используя любые сгенерированные им ID-номера сессии, но ни один из них не приведет к вызову другой пользовательской сессии, поскольку все они будут заменены регенерированными ID-номерами. Если вы склонны к крайним мерам, то можете даже регенерировать ID сессии при каждом запросе.

## Преднамеренная настройка на сессии, использующие исключительно cookie

Если вы собираетесь потребовать от своих пользователей включить cookies при просмотре ваших веб-сайтов, то можете воспользоваться функцией `ini_set`:

```
ini_set('session.use_only_cookies', 1);
```

С такой настройкой трюк с `?PHPSESSID=` будет полностью проигнорирован. Если вы воспользуетесь этой мерой безопасности, я также рекомендую проинформировать ваших пользователей о том, что для работы с вашим сайтом требуются cookie, чтобы они знали причину, по которой им не удается получить требуемых результатов.

## Использование общего сервера

Если вы делите общий сервер с владельцами других учетных записей, то вам наверняка не захочется, чтобы все данные ваших сессий хранились в том же каталоге, где хранятся данные других пользователей. Вместо этого нужно выбрать для хранения ваших сессий каталог, доступный только пользователю с вашей учетной записью (и поэтому невидимый в сетевом окружении). Для этого нужно ближе к началу программы поместить вызов функции `ini_set`:

```
ini_set('session.save_path', '/home/user/myaccount/sessions');
```

В результате такой настройки конфигурации новое значение будет сохраняться только на время выполнения программы, а как только она завершит свою работу, будут возвращены исходные настройки.

Папка с данными *сессий* будет быстро заполняться, и в зависимости от занятости вашего сервера может потребоваться ее периодическая очистка от данных устаревших сессий. Чем чаще она будет использоваться, тем реже будет возникать желание хранить данные сессий.

Следует помнить о том, что ваши веб-сайты могут и будут подвергаться попыткам взлома. Существуют так называемые боты (bots), или сетевые автоматические программы, будоражащие Интернет попытками отыскать уязвимые для атак сайты. Поэтому, что бы вы ни делали, если в своей программе вы проводите какую-либо обработку данных, не имеющих стопроцентной гарантии безопасности, к ним всегда нужно относиться с предельной осторожностью.

Теперь, когда вы в достаточной степени усвоили и PHP, и MySQL, настало время в следующей главе предложить вам введение в язык JavaScript — третью основную технологическую составляющую, рассматриваемую в данной книге.

# Проверьте ваши знания

1. Почему cookie должны быть переданы в начале работы программы?
2. Какая PHP-функция сохраняет cookie на машине веб-браузера?
3. Как можно удалить cookie?
4. Где в PHP-программе сохраняются имя пользователя и пароль при использовании HTTP-аутентификации?
5. Почему функция `md5` считается мощным средством защиты?
6. Что подразумевается под «посыпанием солью» (salting) строки?
7. Что такое PHP-сессия?
8. Как инициируется PHP-сессия?
9. Что такое хищение сессии?
10. Что такое фиксация сессии?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 12».

# 13 Изучение JavaScript

JavaScript придает вашим веб-сайтам динамическую функциональность. Когда вы видите, как при прохождении указателя мыши над каким-нибудь элементом браузера что-нибудь выделяется, появляется новый текст, изменяется цветовое оформление или изображение, то должны понимать, что все это делается с помощью JavaScript. Этот язык предлагает такие эффекты, которых нельзя достичь никакими другими средствами, поскольку он запускается внутри браузера и имеет непосредственный доступ ко всем элементам веб-документа.

Впервые JavaScript появился в браузере Netscape Navigator в 1995 году наряду с добавлением в браузер поддержки Java-технологии. Поскольку изначально сложилось неверное представление о том, что JavaScript был побочным продуктом Java, возникло устойчивое заблуждение об их взаимосвязанности. Но такое название было всего лишь удачным маркетинговым ходом, призванным помочь новому языку сценариев получить преимущества за счет той популярности, которой пользовался язык программирования Java.

Когда HTML-элементы веб-страницы обрели более четкое, структурированное определение в так называемой объектной модели документа — DOM (Document Object Model), язык JavaScript получил еще бóльшие возможности. Объектная модель документа позволила относительно просто добавлять новый абзац или сфокусироваться на какой-нибудь части текста и внести в нее изменения.

Поскольку как в JavaScript, так и в PHP поддерживаются многие элементы синтаксиса структурного программирования, используемые в языке программирования C, эти два языка очень похожи друг на друга. Оба они относятся к языкам высокого уровня. К примеру, у них весьма слабая типизация, позволяющая легко приводить переменную к новому типу данных всего лишь за счет применения ее в новом контексте.

После знакомства с PHP язык JavaScript должен восприниматься еще проще. И его изучение принесет вам несомненное удовольствие, поскольку этот язык является основой технологии Web 2.0 Ajax, которая предоставляет гибко подстраивающийся пользовательский интерфейс, востребованный в наши дни опытными веб-пользователями.

## JavaScript и текст HTML

JavaScript является языком сценариев, который работает исключительно на стороне клиента внутри веб-браузера. Для вызова этого языка его код помещается между открывающим и закрывающим HTML-тегами `<script>` и `</script>`.

Типовой документ Hello World, созданный на HTML 4.01 с применением JavaScript, может иметь вид, показанный в примере 13.1.

**Пример 13.1.** Фраза Hello World, отображаемая с помощью JavaScript

```
<html>
    <head><title>Hello World</title></head>
    <body>
        <script type="text/javascript">
            document.write("Hello World")
        </script>
        <noscript>
            Ваш браузер не поддерживает JavaScript, или его поддержка отключена
        </noscript>
    </body>
</html>
```

Вам могут встретиться веб-страницы, в которых используется не рекомендуемый в наши дни HTML-тег <script language="javascript">. В данном примере применяется более современный и предпочтительный тег <script type="text/javascript">.

Внутри тегов <script> находится всего одна строка кода JavaScript, в которой используется команда document.write, являющаяся эквивалентом PHP-команды echo или команды print. Как и следовало ожидать, она просто выводит предоставленную ей строку в текущий документ при его отображении на экране.

Можно было также заметить, что, в отличие от PHP, в этой команде отсутствует замыкающая точка с запятой (;). Причина в том, что в JavaScript действием, эквивалентным действию точки с запятой, обладает символ новой строки. Тем не менее, если потребуется разместить в одной строке более одной инструкции, то после каждой инструкции, кроме последней, нужно ставить точку с запятой. Разумеется, при желании можете ставить точку с запятой после каждой инструкции. Это нисколько не помешает работе JavaScript.

В этом примере следует также обратить внимание на пару тегов <noscript> и </noscript>. Они используются в том случае, когда вам хочется предоставить альтернативный HTML пользователям, на чьих браузерах JavaScript не поддерживается или отключен. Эти теги применяются по вашему усмотрению и не являются обязательными, но будет лучше, если вы ими воспользуетесь, поскольку предоставить альтернативу в виде статичного HTML тем операциям, для которых применяется JavaScript, обычно не составляет большого труда. Но в последующих примерах, приводимых в данной книге, теги <noscript> будут опускаться, поскольку основное внимание будет уделяться тому, что можно сделать с использованием JavaScript, а не тому, что можно сделать без него.

Когда будет загружен код примера 13.1, на экран веб-браузера с включенным JavaScript будет выведен следующий текст (рис. 13.1):

```
Hello World
```

А те браузеры, на которых JavaScript отключен, выведут следующее сообщение (рис. 13.2):

```
Ваш браузер не поддерживает JavaScript, или его поддержка отключена
```

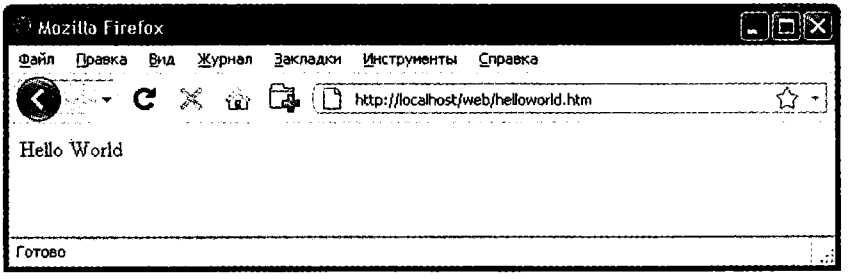



**Рис. 13.1.** Включенный и работающий JavaScript

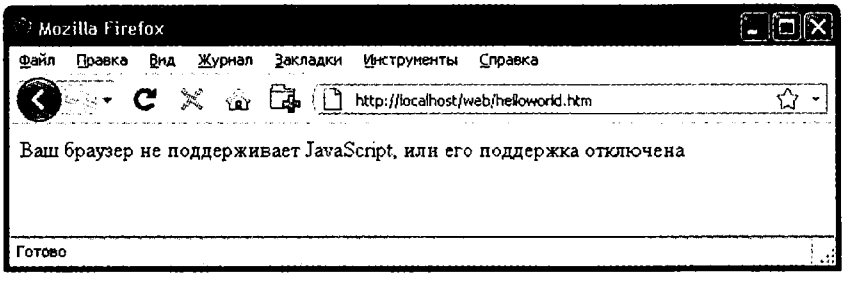



**Рис. 13.2.** JavaScript отключен

## Использование сценариев в заголовке документа

Сценарий можно вставить не только в тело документа, но и в его раздел <head>, являющийся идеальным местом, если нужно выполнить сценарий при загрузке страницы. Присутствие в этом разделе какого-нибудь очень важного кода и функций обеспечивает также их немедленную готовность к использованию в других разделах, имеющих сценарии, в том документе, который зависит от их применения.

Другой причиной для вставки сценария в заголовок документа может быть способность JavaScript записывать в раздел <head> такие элементы, как мета-теги, поскольку то место, куда вставляется сценарий, становится по умолчанию частью документа, куда он осуществляет вывод информации.

## Устаревшие и нестандартные браузеры

Если нужно поддерживать браузеры, не допускающие выполнения сценариев, следует воспользоваться HTML-тегами комментариев (<!-- и -->), препятствующими их встрече с кодом сценария, который они не должны видеть. В примере 13.2 показано, как эти теги добавляются к коду сценария.

**Пример 13.2.** Пример «Hello World», измененный в расчете на использование браузеров, не поддерживающих JavaScript

```
<html>
    <head><title>Hello World</title></head>
    <body>
```

```
        <script type="text/javascript"><!--
            document.write("Hello World")
            // --></script>
    </body>
</html>
```

В данном примере открывающий HTML-тег комментария (<!--) был добавлен сразу же после открывающего тега <script ...>, а тег, закрывающий комментарий, -- непосредственно перед тегом </script>, который закрывает сценарий.

Двойной прямой слеш (//) используется в JavaScript, чтобы показать, что вся остальная строка является комментарием. Он присутствует здесь для того, чтобы браузеры, поддерживающие JavaScript, проигнорировали следующий за ним тег -->, а браузеры, не поддерживающие JavaScript, проигнорировали идущие вначале символы // и обработали тег -->, закрывая тем самым HTML-комментарий.

Хотя такое решение имеет не самый изящный вид, вам при желании поддерживать устаревшие или нестандартные браузеры нужно лишь запомнить используемые в нем две строки, в которые помещается код JavaScript:

```
<script type="text/javascript"><!--
  (Здесь должен быть ваш код JavaScript...)
// --></script>
```

Разумеется, пройдет еще несколько лет, и эти комментарии станут не нужны для любых выпускаемых браузеров.

Есть еще два языка сценариев: VBScript, разработанный корпорацией Microsoft и основанный на языке программирования Visual Basic, и Tcl, представляющий собой язык для быстрой разработки прототипов. Их можно вызвать тем же способом, что и JavaScript, за исключением того, что при объявлении типа нужно указывать соответственно text/vbscript и text/tcl. Язык VBScript работает только в Internet Explorer, его использование в других браузерах требует загрузки дополнительного модуля. Для применения Tcl дополнительный модуль нужен всегда. Поэтому оба этих языка считаются нестандартными и в данной книге не рассматриваются.

## Включение файлов JavaScript

В дополнение к внесению кода JavaScript непосредственно в HTML-документы вы можете включать в них файлы с кодом JavaScript или со своего веб-сайта, или из любого места в Интернете. Для этого используется следующий синтаксис:

```
<script type="text/javascript" src="script.js"></script>
```

А для извлечения файла из Интернета применяется этот синтаксис:

```
<script type="text/javascript" src="http://someserver.com/script.js">
</script>
```

В самих файлах сценариев не должно быть никаких тегов <script> или </script>, поскольку они там не нужны: браузеру и так известно, что будет загружаться файл JavaScript. Применение этих тегов в файлах JavaScript приводит к возникновению ошибки.

Включение файлов сценариев является предпочтительным способом использования на вашем веб-сайте файлов JavaScript, принадлежащих сторонним производителям.

Параметры type="text/javascript" можно не указывать, поскольку все современные браузеры по умолчанию предполагают, что сценарий содержит код JavaScript.

# Отладка кода JavaScript

При изучении JavaScript очень важно иметь возможность отслеживать набранный код или выявлять не связанные с ним ошибки программирования. В отличие от PHP, который отображает сообщения об ошибках в браузере, JavaScript обрабатывает сообщения об ошибках по-другому, и способ их обработки имеет свои особенности, зависящие от используемого браузера. В табл. 13.1 перечислены способы доступа к сообщениям об ошибках JavaScript в каждом из пяти самых распространенных браузеров.

**Таблица 13.1.** Доступ к сообщениям об ошибках JavaScript в различных браузерах

| Браузер | Способ доступа к сообщениям об ошибках JavaScript |
|---|---|
| Apple Safari | В Safari нет консоли ошибок, включенной по умолчанию, но вы можете включить эту функцию, выбрав в меню пункты Safari ▸ Настройки ▸ Дополнения и установив флажок Показывать меню "Разработка" в строке меню. Кроме того, предпочтение можно отдать JavaScript-модулю Firebug Lite, который многие считают более простым в использовании: <script src='http://tinyurl.com/fblite'></script> |
| Google Chrome | Щелкните на значке меню, который похож на страницу с загнутым углом, и выберите пункт Разработчикам ▸ Консоль JavaScript. Можно также воспользоваться сочетанием клавиш Ctrl+Shift+J на PC или Command+Shift+J на Mac |
| Microsoft Internet Explorer | Выберите команду Сервис ▸ Свойства обозревателя ▸ Дополнительно, снимите флажок Отключить отладку сценариев и установите флажок Показывать уведомление о каждой ошибке сценария |
| Mozilla Firefox | Выберите команду Инструменты ▸ Консоль ошибок или воспользуйтесь сочетанием клавиш: Ctrl+Shift+J на PC или Command+Shift+J на Mac |
| Opera | Выберите команду Инструменты ▸ Дополнительно ▸ Консоль ошибок |

Для пользователей Mac OS X: хотя здесь и указан способ использования в Safari консоли ошибок для JavaScript, вы можете отдать предпочтение использованию Google Chrome (для Intel OS X 10.5 или более новой версии), который, на мой взгляд, предлагает разработчикам более широкие функциональные возможности.

Чтобы испытать консоль ошибок, которую вы используете, создадим сценарий, содержащий небольшую ошибку. Пример 13.3 очень похож на пример 13.1, но в нем пропущены закрывающие двойные кавычки в строке `Hello World`, что является довольно распространенной синтаксической ошибкой.

**Пример 13.3.** JavaScript-сценарий Hello World, содержащий ошибку

```
<html>
    <head><title>Hello World</title></head>
    <body>
        <script type="text/javascript">
            document.write("Hello World)
        </script>
    </body>
</html>
```

Наберите этот пример и сохраните его в файле test.html, а затем вызовите в своем браузере. Он сможет лишь вывести название страницы, а в основном окне браузера будет пусто. Теперь вызовите в вашем браузере консоль ошибок и увидите сообщение о незакрытом строковом литерале (в случае использования Firefox):

```
unterminated string literal
document.write("Hello World)
---------------^
```

Обратите внимание на то, что вспомогательная стрелка указывает на начало дефектной части кода. Кроме того, будет сообщено, что дефектный код был обнаружен в строке 5.

В Microsoft Internet Explorer сообщение об ошибке будет иметь вид:

```
Незавершенная строковая константа
```

Здесь нет вспомогательной стрелки, но будет выведено сообщение, что ошибка найдена в строке 5, в позиции 32.

Google Chrome выдаст сообщение о неожиданной недопустимой лексеме:

```
Uncaught SyntaxError: Unexpected token ILLEGAL
```

Будет также сообщено, что ошибка находится в строке 5, без указания ее точного местоположения.

Браузер Opera предоставит сообщение о синтаксической ошибке в строке 2 загружаемого встроенного сценария:

```
Syntax error while loading line 2 of inline script
expected statement
                    document.write("Hello World)
-------------------------------------------^
```

Обратите внимание на то, что, в отличие от других браузеров, Opera сообщает об ошибке в строке 2 встроенного сценария, а не ссылается на номер строки всего HTML-файла.

Opera также пытается указать на начальную позицию проблемного участка, близкую к первой двойной кавычке.

И все же два браузера позволяют обнаружить точное место, где допущена ошибка. Firefox указывает на открывающую двойную кавычку, дающую хорошую отправную точку для поиска, а Internet Explorer сообщает об ошибке в позиции 32, что соответствует точному месту, куда следовало поставить закрывающую двойную кавычку, но из-за отсутствия стрелки, указывающей на эту позицию, для ее поиска нужно подсчитывать символы.

Как видите, в целом, наверное, только Firefox предоставляет самые простые и наиболее точные сообщения, поэтому я рекомендую этот браузер в качестве лучшего для отладки кода JavaScript.

При изучении данной книги вы узнаете о наличии некоторых существенных проблем совместимости, связанных с браузером Microsoft Internet Explorer, который по-прежнему является выбором существенной части веб-серферов. Поэтому, задумываясь о распространении своего продукта, вам перед его выпуском на рабочий сервер следует тестировать свои программы в различных версиях этого браузера.

Вам также следует присмотреться к дополнительному модулю Firebug, разработанному для браузера Firefox (http://getfirebug.com), который пользуется большой популярностью среди разработчиков JavaScript.

Если вам захочется набрать приводимый далее код, чтобы испытать его в работе, не забудьте заключить его в теги <script> и </script>.

# Использование комментариев

В силу общих наследственных черт, приобретенных у языка программирования C, языки PHP и JavaScript имеют много общего и между собой, в частности приемы комментирования кода. В первую очередь это касается однострочных комментариев:

```
// Это комментарий
```

В этой технологии используется пара прямых слешей (//), информирующая JavaScript о том, что все остальные символы должны быть проигнорированы. А затем наступает черед многострочных комментариев:

```
/* Это раздел
   многострочного комментария.
   не подвергаемого
   интерпретации */
```

Многострочный комментарий начинается с последовательности символов /* и заканчивается символами */. Нужно лишь запомнить, что использовать вложенные многострочные комментарии не допускается, поэтому нужно убедиться в отсутствии большого закомментированного участка кода, в котором уже имеются многострочные комментарии.

# Точка с запятой

В отличие от PHP, точка с запятой коду JavaScript, имеющему в строке одну инструкцию, не требуется. Поэтому следующая строка вполне имеет право на существование:

```
x += 10
```

Но при необходимости иметь в строке более одной инструкции, их нужно разделить точками с запятыми:

```
x += 10; y -= 5; z = 0
```

Последнюю точку с запятой можно опустить, поскольку последняя инструкция будет завершена символом новой строки.

В правилах использования точки с запятой есть исключения. При создании микросценариев в виде URL-закладок (bookmarklet) или при завершении инструкции ссылкой на переменную или функцию, когда первым символом расположенной ниже строки будет левая круглая или фигурная скобка, нужно не забыть добавить точку с запятой, иначе JavaScript даст сбой. Поэтому ставьте точку с запятой при любых сомнениях.

# Переменные

В JavaScript нет никаких идентификационных символов переменных, таких как знак доллара ($) в PHP. Вместо этого в отношении имен переменных действуют следующие правила.

- Имена переменных могут включать только буквы a–z, A–Z, цифры 0–9, символ $ и символ подчеркивания (_).
- Никакие другие символы, включая пробелы или знаки пунктуации, использовать в именах переменных не допускается.
- Первым в имени переменной может быть символ из диапазонов a–z, A–Z, символ $ или символ подчеркивания _ (и никаких цифр).
- Имена чувствительны к регистру. Имена Count, count и COUNT принадлежат трем разным переменным.
- Ограничений на длину имени переменной не существует.

И вы наверняка обратили внимание на присутствие в этом перечне символа $. JavaScript допускает его использование, и он может быть первым символом в имени переменной или функции. Такая возможность означает, что благодаря этому можно переносить на JavaScript большие объемы кода PHP значительно быстрее, хотя я не рекомендую его применять в этом качестве.

## Строковые переменные

В JavaScript значения строковых переменных должны быть заключены либо в одиночные, либо в двойные кавычки:

```
greeting = "Привет! "
warning  = 'Осторожно!'
```

В строку в двойных кавычках можно включить одиночную кавычку или же в строку в одинарных кавычках можно включить двойную кавычку. Но кавычка того же типа должна быть отключена путем использования символа обратного слеша:

```
greeting = "\"Привет!\" является приветствием"
warning  = '\'Осторожно!\' является предупреждением'
```

Чтобы прочитать значение из строковой переменной, его можно присвоить другой переменной:

```
newstring = oldstring
```

Или использовать его в функции:

```
status = "Все системы работают успешно"
document.write(status)
```

## Числовые переменные

Создание числовой переменной сводится к простому присваиванию значения, как в следующих примерах:

```
count       = 42
temperature = 98.4
```

Значения числовых переменных могут быть точно так же, как и значения строковых переменных, прочитаны и использованы в выражениях и функциях.

## Массивы

Массивы JavaScript очень похожи на массивы в PHP тем, что они могут содержать строковые или числовые данные, а также другие массивы. Чтобы присвоить массиву значения, используется следующий синтаксис (с помощью которого в данном случае создается строковый массив):

```
toys = ['bat', 'ball', 'whistle', 'puzzle', 'doll']
```

Для создания многомерного массива более мелкие массивы вкладываются в более крупный массив. Для создания двумерного массива цветных квадратов, расположенных на одной из сторон кубика Рубика (в котором используются следующие цвета: красный (R), зеленый (G), оранжевый (O), желтый (Y), синий (B) и белый (W), представленные прописными буквами, указанными в скобках), можно воспользоваться следующим кодом:

```
face =
[
     ['R', 'G', 'Y'],
     ['W', 'R', 'O'],
     ['Y', 'W', 'G']
]
```

Предыдущий пример был отформатирован так, чтобы было понятно, что именно происходит, но его можно переписать и в следующем виде:

```
face = [['R', 'G', 'Y'], ['W', 'R', 'O'], ['Y', 'W', 'G']]
```

или даже так:

```
top = ['R', 'G', 'Y']
mid = ['W', 'R', 'O']
bot = ['Y', 'W', 'G']

face = [top, mid, bot]
```

Для доступа к элементу, расположенному в этой матрице во второй сверху и в третьей слева ячейке, нужно воспользоваться следующим кодом (нужно помнить, что позиционирование элементов массива начинается с нуля):

```
document.write(face[1][2])
```

Эта инструкция выведет букву O, означающую оранжевый цвет.

Массивы JavaScript являются мощной структурой, предназначенной для хранения данных, поэтому в главе 15 они рассматриваются более подробно.

# Операторы

Как и в PHP, операторы в JavaScript могут использоваться в математических операциях, для внесения изменений в строки, а также в операциях сравнения и логических операциях (И, ИЛИ и т. д.). Математические операторы JavaScript во многом похожи на обычные арифметические операторы. Например, следующая инструкция выводит число 15:

```
document.write(13 + 2)
```

Все разнообразие операторов будет рассмотрено в следующих разделах.

## Арифметические операторы

Арифметические операторы предназначены для осуществления математических операций. Из можно использовать для осуществления четырех основных операций (сложения, вычитания, умножения и деления), а также для нахождения модулей (остатков от деления) и инкремента (увеличения на единицу) или декремента (уменьшения на единицу) значения (табл. 13.2).

**Таблица 13.2.** Арифметические операторы

| Оператор | Описание | Пример |
|---|---|---|
| + | Сложение | j + 12 |
| - | Вычитание | j – 22 |
| * | Умножение | j * 7 |
| / | Деление | j / 3.13 |
| % | Деление по модулю (остаток от деления) | j % 6 |
| ++ | Инкремент | ++j |
| -- | Декремент | --j |

# Операторы присваивания

Эти операторы используются для присваивания значений переменным. Их линейка начинается простым знаком равенства (=) и продолжается сочетаниями +=, -= и т. д. Оператор += добавляет значение, находящееся справа, к переменной, находящейся слева, вместо того чтобы целиком заменить значение переменной в левой части.

Поэтому, если изначально значение переменной count было 6, то оператор:

```
count += 1
```

установит для нее значение 7 точно так же, как и более привычный оператор присваивания:

```
count = count + 1
```

В табл. 13.3 перечислены различные операторы присваивания, доступные в JavaScript.

**Таблица 13.3.** Операторы присваивания

| Оператор | Пример | Эквивалентен оператору |
|---|---|---|
| = | j = 99 | j = 99 |
| += | j += 2 | j = j + 2 |
| += | j += 'string' | j = j + 'string' |
| -= | j -= 12 | j = j – 12 |
| *= | j *= 2 | j = j * 2 |
| /= | j /= 6 | j = j / 6 |
| %= | j %= 7 | j = j % 7 |

# Операторы сравнения

Операторы сравнения обычно используются с такими конструкциями, как инструкция if, в которой требуется сравнивать два элемента. Например, может потребоваться узнать, достигло ли значение переменной, подвергаемой автоприращению, определенной величины, или меньше ли значение другой переменной установленной величины и т. д. (табл. 13.4).

**Таблица 13.4.** Операторы сравнения

| Оператор | Описание | Пример |
|---|---|---|
| == | Равно | j == 42 |
| != | Не равно | j != 17 |
| > | Больше | j > 0 |
| < | Меньше | j < 100 |
| >= | Больше или равно | j >= 23 |
| <= | Меньше или равно | j <= 13 |
| === | Равно (и того же типа) | j === 56 |
| !== | Не равно (и того же типа) | j !== '1' |

# Логические операторы

У логических операторов JavaScript, в отличие от PHP, нет эквивалентов and и or для && и || и отсутствует оператор xor (табл. 13.5).

**Таблица 13.5.** Логические операторы

| Оператор | Описание | Пример |
|---|---|---|
| && | И | j == 1 && k == 2 |
| \|\| | ИЛИ | j < 100 \|\| j > 0 |
| ! | НЕ | ! (j role="strong">== k) |

# Инкремент и декремент переменной

Следующие используемые в PHP и уже изучавшиеся вами формы инкремента (приращения) и декремента (отрицательного приращения), осуществляемые как после операции сравнения, так и перед ней, поддерживаются также и в JavaScript:

```
++x
 y
x += 22
y -= 3
```

# Объединение строк

Объединение (конкатенация) строк в JavaScript осуществляется немного иначе, чем в PHP. Вместо оператора . (точка) используется знак «плюс» (+):

```
document.write("У вас " + messages + " сообщения.")
```

Если предположить, что переменной messages присвоено значение 3, эта строка кода выведет следующую информацию:

```
У вас 3 сообщения.
```

Оператор += точно так же, как и при добавлении значения к числовой переменной, позволяет добавить одну строку к другой:

```
name =  "James"
name += " Dean"
```

# Управляющие символы

Управляющие символы, использование которых вы видели при вставке в строку кавычек, могут применяться также для вставки различных специальных символов: табуляции, новой строки и возврата каретки. В следующем примере символы табуляции используются для разметки заголовка; они включены в строку исключительно для иллюстрации использования управляющих символов, поскольку в вебстраницах существуют более подходящие способы разметки:

```
heading = "Name\tAge\tLocation"
```

В таблице 13.6 приведены управляющие символы, доступные в JavaScript.

**Таблица 13.6.** Управляющие символы JavaScript

| Символ | Назначение |
|---|---|
| \b | Забой |
| \f | Перевод страницы |
| \n | Новая строка |
| \r | Возврат каретки |
| \t | Табуляция |
| \' | Одиночная кавычка |
| \" | Двойная кавычка |
| \\ | Обратный слеш |
| \XXX | Восьмеричное число в диапазоне от 000 до 377, представляющее эквивалент символа Latin-1 (например, \251 для символа ©) |
| \xXX | Шестнадцатеричное число в диапазоне от 00 до FF, представляющее эквивалент символа Latin-1 (например, \xA9 для символа ©) |
| \uXXXX | Шестнадцатеричное число в диапазоне от 0000 до FFFF, представляющее эквивалент символа Unicode (например, \u00A9 для символа ©) |

# Типизация переменных

Как и PHP, JavaScript относится к весьма слабо типизированным языкам; тип переменной определяется только при присваивании ей значения и может изменяться в случае появления переменной в другом контексте. Как правило, о типе переменной волноваться не приходится: язык JavaScript сам определяет, что именно вам нужно, и просто делает это.

Посмотрите на пример 13.4, в котором выполняются следующие действия.

1. Переменной n присваивается строковое значение 838102050, в следующей строке осуществляется вывод ее значения, а чтобы посмотреть на ее тип, используется инструкция typeof.
2. Переменной n присваивается значение, получаемое при перемножении чисел 12 345 и 67 890. Это значение также равно 838 102 050, но оно является числом, а не строкой. Затем определяется и выводится на экран тип переменной.
3. К числу n добавляется текст, и результат отображается на экране.

**Пример 13.4.** Установка типа переменной путем присваивания ей значения

```
<script>
n = '838102050'         // Присваивание 'n' строкового значения
document.write('n = ' + n + '. и имеет тип ' + typeof n + '<br />')

n = 12345 * 67890;      // Присваивание 'n' числа
document.write('n = ' + n + '. и имеет тип ' + typeof n + '<br />')

n += ' плюс текст' // Изменение типа 'n' с числового на строковое
document.write('n = ' + n + '. и имеет тип ' + typeof n + '<br />')
</script>
```

Этот сценарий выведет следующую информацию:

```
n = 838102050. и имеет тип string
n = 838102050. и имеет тип number
n = 838102050 плюс текст. и имеет тип string
```

Если в отношении типа переменной есть какие-то сомнения или нужно обеспечить, чтобы переменная относилась к определенному типу, вы можете принудительно привести ее к этому типу, используя операторы, показанные в следующем примере (которые превращают строку в число и число в строку соответственно):

```
n = "123"
n *= 1    // Превращение 'n' в число

n = 123
n += ""   // Превращение 'n' в строку
```

Или же тип переменной можно всегда определить с помощью инструкции typeof.

# Функции

Как и в PHP, в JavaScript функции используются для выделения фрагментов кода, выполняющих конкретную задачу. Для создания функции ее нужно объявить, как показано в примере 13.5.

**Пример 13.5.** Объявление простой функции

```
<script>
function product(a, b)
{
    return a*b
}
</script>
```

Эта функция принимает два переданных ей параметра, перемножает их и возвращает произведение.

# Глобальные переменные

К глобальным относятся переменные, определенные за пределами любых функций (или внутри функций, но без использования ключевого слова var). Они могут быть определены следующими способами:

```
    a = 123              // Глобальная область видимости
var b = 456              // Глобальная область видимости
if (a == 123) var c = 789 // Глобальная область видимости
```

Независимо от применения ключевого слова var, если переменная определена за пределами функции, она приобретает глобальную область видимости. Это означает, что к ней может быть получен доступ из любой части сценария.

## Локальные переменные

Параметры, переданные функции, автоматически приобретают локальную область видимости, то есть к ним можно обращаться только из тела этой функции. Но есть одно исключение. Массивы передаются функции по ссылке, поэтому, если вы внесете изменения в любые элементы массива, переданного в качестве параметра, то элементы исходного массива также будут изменены.

Для определения локальной переменной, имеющей область видимости только внутри текущей функции и не переданной ей в качестве параметра, используется ключевое слово var. В примере 13.6 показана функция, которая создает одну переменную с глобальной и две переменные – с локальными областями видимости.

**Пример 13.6.** Функция, создающая переменные с глобальной и локальной областями видимости

```
<script>
function test()
{
        a = 123                 // Глобальная область видимости
    var b = 456                 // Локальная область видимости
    if (a == 123) var c = 789 // Локальная область видимости
}
</script>
```

В PHP, чтобы проверить работоспособность установки области видимости, можно воспользоваться функцией isset. Но в JavaScript такой функции нет, поэтому в примере 13.7 применяется инструкция typeof, возвращающая строку undefined, если переменная не определена.

**Пример 13.7.** Проверка области видимости переменных, определенных в функции test

```
<script>
test()

if (typeof a != 'undefined') document.write('a = "' + a + '"<br />')
if (typeof b != 'undefined') document.write('b = "' + b + '"<br />')
if (typeof c != 'undefined') document.write('c = "' + c + '"<br />')

function test()
{
        a = 123
    var b = 456

    if (a == 123) var c = 789
}
```

Этот сценарий выведет только одну строку:

```
a = "123"
```

Это свидетельствует о том, что, как и предполагалось, глобальная область видимости была определена только для одной переменной, потому что для перемен-

ных b и c установкой перед ними ключевого слова var была задана локальная область видимости.

Если ваш браузер выведет предупреждение о том, что переменная b не определена, то, при всей своей справедливости, оно может быть проигнорировано.

# Объектная модель документа

Разработчики JavaScript были очень умными людьми. Вместо того чтобы просто создать еще один язык написания сценариев (который имел бы на момент создания весьма неплохие усовершенствования), они проявили дальновидность и построили его вокруг объектной модели документа, или DOM (Document Object Model). Эта модель разбивает части HTML-документа на отдельные *объекты*, у каждого из которых есть собственные *свойства* и *методы* и каждым из которых можно управлять с помощью JavaScript.

В JavaScript объекты, свойства и методы разделяются с помощью точек (это одна из причин того, что в качестве оператора объединения строк в JavaScript используется знак +, а не точка). Рассмотрим, к примеру, в качестве объекта с именем card визитную карточку. Этот объект содержит такие свойства, как имя — name, адрес — address, номер телефона — phone и т. д. В синтаксисе JavaScript эти свойства будут иметь следующий вид:

```
card.name
card.phone
card.address
```

Его методами будут функции, занимающиеся извлечением, изменением и другими действиями со свойствами. Например, для вызова метода, отображающего свойства объекта card, можно воспользоваться следующим синтаксисом:

```
card.display()
```

Взгляните на некоторые из представленных ранее в этой главе примеров и обратите внимание на те из них, в которых используется инструкция document.write. Уяснив, что JavaScript основан на работе с объектами, вы поймете, что write — это метод объекта document.

Внутри JavaScript выстраивается иерархия из родительских и дочерних объектов. Эта иерархия и называется объектной моделью документа — DOM (рис. 13.3).

На этом рисунке используются уже знакомые вам HTML-теги, иллюстрирующие родительско-дочерние взаимоотношения между различными объектами документа. Например, URL-адрес внутри ссылки является частью тела (body) HTML-документа. В JavaScript на него можно сослаться следующим образом:

```
url = document.links.linkname.href
```

Обратите внимание на то, как эта ссылка идет сверху вниз по центральному столбцу. Первая часть, document, ссылается не теги <html> и <body>, links.linkname — на тег <a ...>, а href — на элемент href=....

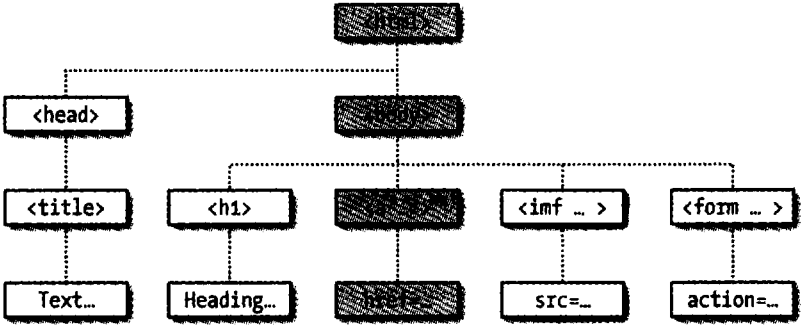



**Рис. 13.3.** Пример иерархии объектов DOM

Давайте превратим это в некий код HTML и в сценарий для чтения свойств ссылки. Наберите код примера 13.8 и сохраните его в файле с именем `linktest.html`, а затем вызовите в своем браузере.

Если в качестве основного разработочного браузера вы используете Microsoft Internet Explorer, пожалуйста, ограничьтесь только чтением этого подраздела, затем прочитайте следующий подраздел «Но не все так просто», вернитесь к данному подразделу и воспользуйтесь в примере модификацией getElementById, рассмотренной в следующем подразделе. Без нее код примера работать не будет.

**Пример 13.8.** Чтение ссылки на URL-адрес с помощью JavaScript

```
<html>
    <head>
        <title>Тестирование ссылки</title>
    </head>
    <body>
        <a id="mylink" href="http://mysite.com">Щелкни</a><br />
        <script>
            url = document.links.mylink.href
            document.write('URL-адрес - ' + url)
        </script>
    </body>
</html>
```

Обратите внимание на краткую форму тегов `script`, в которой для экономии времени на набор текста был опущен параметр `type="text/JavaScript"`. При желании ради проверки этого (и других примеров) можете также опустить все, кроме тегов `<script>` и `</script>`. Код этого примера выведет следующую информацию:

```
Щелкни
URL-адрес - http://mysite.com
```

Вторая строка выведенной информации появилась благодаря работе метода `document.write`. Обратите внимание на то, как код следует сверху вниз по дереву

документа от document к links, к mylink (идентификатору, присвоенному ссылке) и к href (значению, содержащему URL-адрес назначения).

Есть также краткая форма, работающая не менее успешно, которая начинается со значения, присвоенного атрибуту id: mylink.href. Поэтому следующую строку:

```
url = document.links.mylink.href
```

можно заменить строкой

```
url = mylink.href
```

# Но не все так просто

Код примера 13.8 будет великолепно работать в Safari, Firefox, Opera или Chrome, но только не в Internet Explorer, поскольку реализация языка JavaScript, созданная в Microsoft под именем JScript, имеет множество коварных отличий от общепризнанных стандартов. Добро пожаловать в мир современной веб-разработки!

А как же справиться с этой проблемой? В данном случае вместо использования дочернего объекта links, принадлежащего родительскому объекту document, который Internet Explorer отказывается воспринимать таким образом, нужно применять метод, извлекающий элемент по его идентификатору. Поэтому следующую строку:

```
url = document.links.mylink.href
```

можно заменить строкой:

```
url = document.getElementById('mylink').href
```

и теперь сценарий будет работать на всех основных браузерах. Кстати, когда не нужно искать элемент по его идентификатору, следующая краткая форма будет работать и в Internet Explorer, как, собственно, и во всех остальных браузерах:

```
url = mylink.href
```

## Еще одно использование знака $

Как уже упоминалось, символ $ разрешено использовать в именах переменных и функций JavaScript. По этой причине иногда можно встретить код, имеющий довольно странный вид:

```
url = $('mylink').href
```

Некоторые изобретательные программисты решили, что метод getElementById слишком часто применяется в JavaScript, и написали взамен него свою функцию, показанную в примере 13.9, присвоив ей имя $.

**Пример 13.9.** Функция, заменяющая метод getElementById

```
<script>
function $(id)
{
    return document.getElementById(id)
}
</script>
```

Поэтому, как только функция $ будет вставлена в ваш код, синтаксис:

```
$('mylink').href
```

может заменить следующий код:

```
document.getElementById('mylink').href
```

## Использование DOM

На самом деле объект links является массивом, состоящим из URL-адресов, поэтому на URL mylink в примере 13.8 можно спокойно ссылаться во всех браузерах, используя для этого следующий код (поскольку это первая и единственная ссылка):

```
url = document.links[0].href
```

Если нужно узнать, сколько ссылок содержится во всем документе, можно запросить свойство length объекта links:

```
numlinks = document.links.length
```

Благодаря этому свойству можно извлечь и отобразить все имеющиеся в документе ссылки:

```
for (j=0 : j < document.links.length : ++j)
    document.write(document.links[j].href + '<br />')
```

Длина — length является свойством каждого массива, а также многих других объектов. Например, можно запросить количество записей в истории вашего веб-браузера:

```
document.write(history.length)
```

Но чтобы исключить перехват ваших веб-сайтов с помощью истории веб-браузера, в объекте history хранится только количество сайтов в массиве и вы не можете прочитать или записать значения, относящиеся к этим сайтам. Но вы можете заменить текущую страницу одной из тех, что хранятся в истории, если знаете, на какой позиции она там находится. Это может пригодиться в тех случаях, когда вам известны конкретные страницы, попавшие в историю при переходах с вашей страницы, или вы просто хотите вернуть браузер назад на одну или несколько страниц, что делается с помощью метода go объекта history. Например, чтобы отправить браузер назад на три страницы, нужно выдать следующую команду:

```
history.go(-3)
```

Для перехода вперед и назад на одну страницу можно воспользоваться также следующими методами:

```
history.back()
history.forward()
```

Подобным способом можно заменить только что загруженный URL-адрес другим по вашему выбору:

```
document.location.href = 'http://google.com'
```

Конечно, DOM можно использовать для решения куда более широкого круга задач, чем чтение и модификация ссылок. По мере изучения следующих глав, посвященных JavaScript, знакомство с DOM и с доступом к этой модели станет еще более тесным.

# Проверьте ваши знания

1. Какие теги используются для заключения в них кода JavaScript?
2. К какой части документа будет по умолчанию добавлена информация, выводимая кодом JavaScript?
3. Как в ваши документы может быть включен код JavaScript из другого источника?
4. Какая функция JavaScript является эквивалентом PHP-команд `echo` или `print`?
5. Как можно создать комментарий в JavaScript?
6. Какой оператор используется в JavaScript для объединения строк?
7. Какое ключевое слово можно применять внутри функции JavaScript для определения переменной, имеющей локальную область видимости?
8. Покажите два метода, работающие на всех браузерах и позволяющие отобразить присвоенный ссылке URL-адрес на основе `id` этой ссылки.
9. Какие две команды JavaScript заставят браузер загрузить предыдущую страницу, содержащуюся в его массиве `history`?
10. Какой командой JavaScript вы воспользуетесь для замены текущего документа главной страницей веб-сайта **oreilly.com**?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 13».

# 14 Выражения и управление процессом выполнения сценариев в JavaScript

В предыдущей главе были изложены основы JavaScript и DOM. А теперь настало время рассмотреть порядок построения в JavaScript сложных выражений и способов управления процессом выполнения ваших сценариев с помощью условных инструкций.

## Выражения

Выражения в JavaScript очень похожи на выражения в PHP. В главе 4 мы выяснили, что выражение представляет собой сочетание значений, переменных, операторов и функций, в результате вычисления которого получается значение, могущее быть числовым, строковым или булевым (вычисляемым либо как истина — `true`, либо как ложь — `false`).

В примере 14.1 показаны несколько простых выражений. Код каждой строки выводит буквы от `a` до `d`, за которыми следуют двоеточие и результат вычисления выражения (тег `<br />` служит для перехода на новую строку и разделения выводимой информации на четыре строки).

**Пример 14.1.** Четыре примера булевых выражений

```
<script>
document.write("a: " + (42 > 3) + "<br />")
document.write("b: " + (91 < 4) + "<br />")
document.write("c: " + (8 == 2) + "<br />")
document.write("d: " + (4 < 17) + "<br />")
</script>
```

Этот код выведет следующую информацию:

```
a: true
b: false
c: false
d: true
```

Заметьте, что выражения a: и d: вычисляются как true, а выражения b: и c: — как false. В отличие от языка PHP (который вывел бы соответственно число 1 и пустое место), JavaScript выводит строки true и false.

В JavaScript при проверке значения на истинность или ложность все выражения вычисляются как true, за исключением следующих, которые вычисляются как false: самой строки false, 0, –0, пустой строки, null, неопределенного значения (undefined) и NaN (Not a Number — «не число», понятие в вычислительной технике для недопустимых операций с числами с плавающей точкой, таких как деление на ноль).

Учтите, что, в отличие от PHP, в JavaScript значения true и false пишутся в нижнем регистре. Поэтому из следующих двух инструкций информацию будет отображать только первая, выводя на экран слово true в нижнем регистре, поскольку вторая инструкция вызовет сообщение об ошибке, где будет сказано, что переменная TRUE не определена:

```
if (1 == true) document.write('true') // Истина
if (1 == TRUE) document.write('TRUE') // Приводит к ошибке
```

Следует помнить, что любые фрагменты кода, которые вам захочется набрать и опробовать, запустив их в HTML-файле, нужно заключать в теги <script> и </script>.

## Литералы и переменные

Простейшей формой выражения является *литерал*, означающий нечто, вычисляемое само в себя, например, число 22 или строка «Нажмите клавишу Enter». Выражение может также быть переменной, которая вычисляется в присвоенное ей значение. И литералы, и переменные относятся к типам выражений, поскольку они возвращают значение.

В примере 14.2 показаны пять разных литералов, и все они возвращают значения, хотя и разных типов.

**Пример 14.2.** Пять типов литералов

```
<script>
myname = "Peter"
myage  = 24
document.write("a: " + 42     + "<br />") // Числовой литерал
document.write("b: " + "Hi"   + "<br />") // Строковый литерал
document.write("c: " + true   + "<br />") // Литерал константы
document.write("d: " + myname + "<br />") // Литерал строковой переменной
document.write("e: " + myage  + "<br />") // Литерал числовой переменной
</script>
```

Как и можно было ожидать, в выводимой информации будут показаны значения, возвращаемые всеми этими литералами:

```
a: 42
b: Hi
c: true
d: Peter
e: 24
```

Операторы позволяют создавать более сложные выражения, вычисляемые в полезные результаты. При объединении присваивания или управляющей конструкции с выражениями получается *инструкция*.

В примере 14.3 показано по одной инструкции каждого вида. В первой результат выражения `366 - day_number` присваивается переменной `days_to_new_year`, а во второй приятное сообщение выводится только в том случае, если выражение `days_to_new_year < 30` вычисляется как `true`.

**Пример 14.3.** Две простые инструкции JavaScript

```
<script>
days_to_new_year = 366 - day_number;
if (days_to_new_year < 30) document.write("Скоро Новый год!")
</script>
```

# Операторы

JavaScript предлагает большое количество мощных операторов, начиная с арифметических, строковых и логических и заканчивая операторами присваивания, сравнения и т. д. (табл. 14.1).

**Таблица 14.1.** Типы операторов JavaScript

| Оператор | Описание | Пример |
|---|---|---|
| Арифметический | Основные математические операции | a + b |
| Для работы с массивами | Работа с массивами | a + b |
| Присваивания | Присваивание значений | a = b + 23 |
| Поразрядный | Обработка битов в байтах | 12 ^ 9 |
| Сравнения | Сравнение двух значений | a < b |
| Инкремента и декремента | Прибавление или вычитание единицы | a++ |
| Логический | Булевы операции | a && b |
| Строковый | Объединение | a + 'строка' |

Различные типы операторов воспринимают разное количество операндов.

- *Унарные* операторы, к примеру операторы инкремента (`a++`) или изменения знака числа (`-a`), воспринимают один операнд.
- *Бинарные* операторы, представленные основной массой операторов JavaScript (включая операторы сложения, вычитания, умножения и деления), воспринимают два операнда.

- И один *трехкомпонентный* (ternary) оператор, имеющий форму `? x : y`, является краткой однострочной формой инструкции `if`, которая выбирает одно из двух выражений на основе значения третьего выражения. Этот условный оператор воспринимает три операнда.

## Приоритетность операторов

Как и в PHP, в JavaScript используется приоритетность операторов, благодаря которой некоторые операторы в выражении считаются более важными, чем другие, и поэтому вычисляются в первую очередь.

В табл. 14.2 перечислены операторы JavaScript, расположенные по уровню их приоритетности.

**Таблица 14.2.** Операторы JavaScript, расположенные по уровню их приоритетности (сверху вниз)

| Оператор(ы) | Тип(ы) |
|---|---|
| () [] . | Скобки, вызов и составляющая объекта |
| ++ -- | Инкремент и декремент |
| + - ~ ! | Унарные, поразрядные и логические |
| * / % | Арифметические |
| + - | Арифметические и строковые |
| << >> >>> | Поразрядные |
| < > <= >= | Сравнения |
| == != === !== | Сравнения |
| & ^ \| | Поразрядные |
| && | Логический |
| \|\| | Логический |
| ? : | Трехкомпонентный |
| = += -= *= /= %= <<= >>= >>>= &= ^= \|= | Присваивания |
| , | Последовательного вычисления |

## Взаимосвязанность

Большинство операторов JavaScript обрабатываются в уравнении слева направо. Но для некоторых операторов требуется обработка справа налево. Направление обработки обусловливается *взаимосвязанностью* операторов.

Эта взаимосвязанность вступает в силу в отсутствие явно заданной приоритетности. Рассмотрим, например, следующие операторы присваивания, благодаря которым трем переменным присваивается значение 0:

```
level = score = time = 0
```

Это множественное присваивание становится возможным только благодаря тому, что самая правая часть выражения вычисляется первой, а затем обработка

продолжается справа налево. В табл. 14.3 перечислены операторы, имеющие взаимосвязанность справа налево.

**Таблица 14.3.** Операторы, имеющие взаимосвязанность справа налево

| Оператор | Описание |
|---|---|
| New | Создание нового объекта |
| ++ -- | Инкремент и декремент |
| + - ~ ! | Унарные и поразрядные операции |
| ? : | Условный оператор |
| = *= /= %= += -= <<= >>= >>>= &= ^= \|= | Присваивание |

# Операторы отношения

Операторы отношения проверяют значения двух операндов и возвращают логический результат, равный либо true, либо false. Существует три типа операторов отношения: операторы *равенства*, *сравнения* и *логические*.

## Операторы равенства

Оператор равенства имеет вид == (его не следует путать с оператором присваивания =). В примере 14.4 первая инструкция присваивает значение, а вторая проверяет это значение на равенство. В данных условиях на экран ничего не будет выведено, потому что переменной month присвоено значение July и его сравнение со значением October будет неудачным.

**Пример 14.4.** Присваивание значения и проверка на равенство

```
<script>
month = "July"
if (month == "October") document.write("It's the Fall")
</script>
```

Если два операнда выражения равенства принадлежат к разным типам данных, JavaScript приведет их к тому типу, который имеет для него наибольший смысл. Например, любые строки, полностью состоящие из цифр, при сравнении с числом будут преобразованы в числа. В примере 14.5 у переменных a и b два разных значения (одно из них является числом, а второе — строкой), и поэтому вряд ли стоило ожидать, что какая-нибудь из инструкций if выведет результат.

**Пример 14.5.** Операторы равенства и тождественности

```
<script>
a = 3.1415927
b = "3.1415927"
if (a == b) document.write("1")
if (a === b) document.write("2")
</script>
```

Тем не менее, если запустить код этого примера, вы увидите, что он выведет цифру 1, что будет означать, что первая инструкция if была вычислена как true.

Это произошло благодаря тому, что строковое значение переменной b сначала было временно преобразовано в число, и поэтому обе половины уравнения получили числовое значение 3.1415927.

В отличие от этого вторая инструкция if использует оператор тождественности — три знака равенства подряд, который препятствует автоматическому преобразованию типов в JavaScript. Этот означает, что в данном случае значения a и b считаются разными, поэтому на экран ничего не выводится.

Точно так же, как при принудительном задании приоритета операторов, если у вас появятся сомнения, связанные с преобразованиями типов операндов, производимыми JavaScript, то для отключения этого поведения можно воспользоваться оператором тождественности.

## Операторы сравнения

Используя операторы сравнения, можно проверить не только равенство или неравенство. JavaScript предоставляет в ваше распоряжение также операторы > (больше), < (меньше), >= (больше или равно) и <= (меньше или равно). Код примера 14.6 показывает эти операторы в действии.

**Пример 14.6.** Четыре оператора сравнения

```
<script>
a = 7; b = 11
if (a > b)  document.write("a больше b<br />")
if (a < b)  document.write("a меньше b<br />")
if (a >= b) document.write("a больше или равно b<br />")
if (a <= b) document.write("a меньше или равно b<br />")
</script>
```

Если a равно 7, а b равно 11, то код этого примера выведет следующую информацию (потому что 7 меньше 11, а также меньше или равно 11):

```
a меньше b
a меньше или равно b
```

## Логические операторы

Логические операторы выдают истинные или ложные результаты, поэтому их также называют булевыми. В JavaScript используются три логических оператора (табл. 14.4).

**Таблица 14.4.** Логические операторы JavaScript

| Логический оператор | Описание |
|---|---|
| && | (И) Возвращает истинное значение (true), если оба операнда имеют истинные значения |
| \|\| | (ИЛИ) Возвращает истинное значение (true), если любой из операторов имеет истинное значение |
| ! | (НЕ) Возвращает истинное значение (true), если операнд имеет ложное значение, или ложное значение (false), если операнд имеет истинное значение |

Код примера 14.7 показывает возможности использования этих операторов, при которых возвращаются 0, 1 и true.

**Пример 14.7.** Использование логических операторов

```
<script>
a = 1; b = 0
document.write((a && b) + "<br />")
document.write((a || b) + "<br />")
document.write(( !b ) + "<br />")
</script>
```

Оператору && для возвращения значения true нужно, чтобы оба операнда имели значение true. Оператору || для возвращения значения true необходимо, чтобы любой операнд имел значение true, а третий оператор применяет к значению переменной b операцию НЕ, превращая ее значение из 0 в true.

При использовании оператора || могут возникнуть непредвиденные проблемы, поскольку второй операнд не будет вычисляться, если при вычислении первого будет получен результат true. В примере 14.8 функция getnext никогда не будет вызвана, если переменная finished имеет значение 1.

**Пример 14.8.** Инструкция, использующая оператор ||

```
<script>
if (finished == 1 || getnext() == 1) done = 1
</script>
```

Если *нужно*, чтобы getnext была вызвана при каждом выполнении инструкции if, следует переписать код, как показано в примере 14.9.

**Пример 14.9.** Инструкция if...ИЛИ, измененная для гарантированного вызова getnext

```
<script>
gn = getnext()
if (finished == 1 || gn == 1) done = 1
</script>
```

В данном случае код функции getnext будет выполнен, и он вернет значение, которое будет сохранено в переменной gn до выполнения инструкции if.

В табл. 14.5 показаны все допустимые варианты использования логических операторов. Следует заметить, что выражение !true эквивалентно false, а выражение !false эквивалентно true.

**Таблица 14.5.** Все логические выражения, допустимые в JavaScript

| Входные данные | | Операторы и результаты | |
|---|---|---|---|
| **A** | **b** | **&&** | **\|\|** |
| true | true | true | true |
| true | false | false | true |
| false | true | true | true |
| false | false | false | false |

# Инструкция with

Инструкция with в предыдущих главах, посвященных PHP, не встречалась, поскольку она имеется только в JavaScript. Используя эту инструкцию (если вы понимаете, что я под этим подразумеваю), можно упростить некоторые типы инструкций JavaScript, сократив множество ссылок на объект до всего одной ссылки. Предполагается, что ссылки на свойства и методы внутри блока with должны применяться к указанному объекту.

Рассмотрим код примера 14.10, в котором функция document.write нигде не ссылается на переменную string по имени.

**Пример 14.10.** Использование инструкции with

```
<script>
string = "Шустрая бурая лисица перепрыгивает через ленивую собаку"

with (string)
{
    document.write("В строке " + length + " символов <br />")
    document.write("В верхнем регистре: " + toUpperCase())
}
</script>
```

Даже при том, что в document.write нет никакой непосредственной ссылки на string, этот код все равно справляется с выводом следующей информации:

```
В строке 55 символов
В верхнем регистре: ШУСТРАЯ БУРАЯ ЛИСИЦА ПЕРЕПРЫГИВАЕТ ЧЕРЕЗ ЛЕНИВУЮ СОБАКУ
```

Код примера работает следующим образом: интерпретатор JavaScript распознает, что свойство length и метод toUpperCase() должны быть применены к какому-то объекту. Поскольку они указаны только сами по себе, интерпретатор предполагает, что они применяются к объекту string, указанному в инструкции with.

# Использование события onerror

Рассмотрим еще одну конструкцию, недоступную в PHP. Используя либо событие onerror, либо сочетание ключевых слов try и catch, можно перехватить ошибки JavaScript и самостоятельно справиться с ними.

*Событиями* называются действия, которые могут быть обнаружены JavaScript. Каждый элемент на веб-странице имеет конкретные события, которыми могут быть приведены в действие функции JavaScript. Например, событие щелчка — onclick, принадлежащее элементу button (кнопка), может быть настроено на вызов функции, которая будет запущена, если пользователь щелкнет на кнопке. В примере 14.11 показано, как можно воспользоваться событием onError.

**Пример 14.11.** Сценарий, использующий событие onerror

```
<script>
onerror = errorHandler
```

```
document.writ("Добро пожаловать на этот веб-сайт!") // Преднамеренная ошибка

function errorHandler(message, url, line)
{
    out  = "К сожалению, обнаружена ошибка.\n\n";
    out += "Ошибка: " + message + "\n";
    out += "URL: "   + url + "\n";
    out += "Строка: "  + line + "\n\n";
    out += "Щелкните на кнопке OK для продолжения работы.\n\n";
    alert(out);
    return true;
}
</script>
```

В первой строке сценария событию ошибки предписывается впредь использовать новую функцию `errorHandler`. Эта функция воспринимает три параметра: сообщение, URL-адрес и номер строки, — поэтому остается лишь отобразить все это в появляющемся окне метода `alert`.

Затем для проверки работы новой функции в коде сценария преднамеренно допускается ошибка: вызывается `document.writ` вместо `document.write` (не ставится последняя буква «е»). На рис. 14.1 показан результат запуска этого сценария в браузере. Подобное использование события `onerror` может пригодиться при отладке сценария.

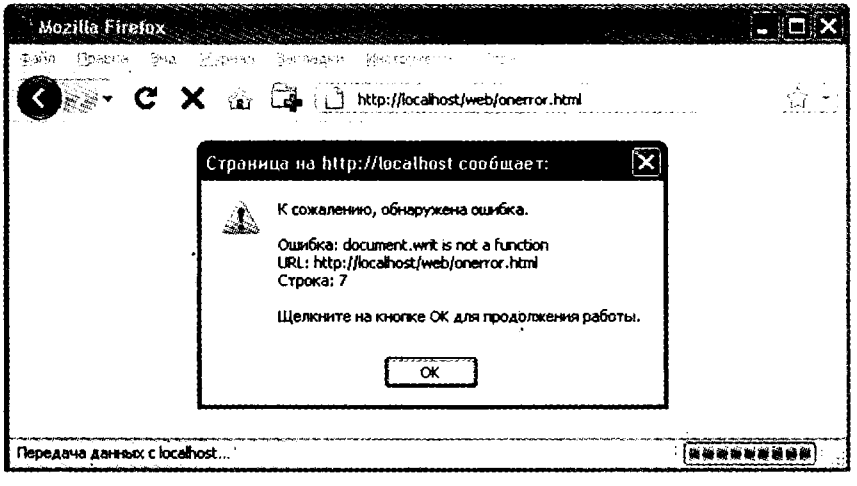



**Рис. 14.1.** Использование события onerror с методом alert для вывода информации

# Использование конструкции try...catch

Технология, в которой используются ключевые слова `try` и `catch`, считается более стандартной и гибкой, чем обработка события `onerror`, показанная в предыдущем

разделе. Эти ключевые слова позволяют перехватывать ошибки для избранного раздела кода, а не для всех сценариев, имеющихся в документе. Но данная технология не позволяет перехватывать синтаксические ошибки, для чего приходится применять обработку события onerror.

Конструкция try...catch поддерживается всеми основными браузерами и задействуется в тех случаях, когда нужно перехватить управление при наступлении определенных условий, о которых известно то, что они могут сложиться в определенной части вашего кода.

Например, в главе 17 будет рассматриваться технология Ajax, в которой используется объект XMLHttpRequest. К сожалению, в браузере Internet Explorer этот объект недоступен (хотя во всех основных браузерах он присутствует). Поэтому для обеспечения совместимости нужно применять try и catch, чтобы перехватить управление в случае отсутствия этого объекта и задействовать какие-нибудь другие технологии. Как это делается, показано в примере 14.12.

**Пример 14.12.** Перехват ошибки с помощью ключевых слов try и catch

```
<script>
try
{
    request = new XMLHTTPRequest()
}
catch(err)
{
    // Использование другого метода для создания запроса
    // XML HTTP Request object
}
</script>
```

Я не хочу здесь вникать в подробности реализации отсутствующего в Internet Explorer объекта, но вы можете увидеть, как работает эта система. Со словами try и catch связано еще одно ключевое слово — finally. Блок кода, следующий за этим словом, выполняется всегда, независимо от того, возникла или не возникла ошибка при выполнении блока кода под ключевым словом try. Чтобы воспользоваться этим ключевым словом, нужно после инструкции catch просто добавить что-нибудь, похожее на следующий пример:

```
finally
{
    alert("Был обнаружен блок кода 'try'.")
}
```

# Условия

Условия изменяют процесс выполнения программы. Они позволяют задавать конкретные вопросы и по-разному реагировать на полученные ответы. Существуют три типа условий, не связанных с циклами: инструкция if, инструкция switch и оператор ?.

# Инструкция if

Инструкции `if` уже использовались в нескольких примерах данной главы. Код внутри этой инструкции выполняется только в том случае, если заданное выражение вычисляется как `true`. Многострочные инструкции `if` заключаются в фигурные скобки, но, как и в PHP, для однострочных инструкций скобки можно опустить. Поэтому допустимы все следующие инструкции:

```
if (a > 100)
{
    b=2
    document.write("a больше 100")
}

if (b == 10) document.write("b равно 10")
```

## Инструкция else

Если условие не было соблюдено, то с помощью инструкции `else` может быть выполнен альтернативный блок кода:

```
if (a > 100)
{
    document.write("a больше 100")
}
else
{
    document.write("a меньше или равно 100")
}
```

В JavaScript, в отличие от PHP, нет инструкции `elseif`, но ее отсутствие компенсируется возможностью использования инструкции `else`, за которой следует еще одна инструкция `if`, чем создается эквивалент инструкции `elseif`:

```
if (a > 100)
{
    document.write("a больше 100")
}
else if(a < 100)
{
    document.write("a меньше 100")
}
else
{
    document.write("a равно 100")
}
```

За инструкцией `else` после новой инструкции `if` точно так же может следовать еще одна инструкция `if` и т. д. Несмотря на то что в этих инструкциях использованы фигурные скобки, наличие внутри каждой пары этих скобок всего одной строки кода позволяет переписать весь пример следующим образом:

```
if (a > 100) document.write("a больше 100")
else if(a < 100) document.write("a меньше 100")
else document.write("a равно 100")
```

## Инструкция switch

Инструкция switch применяется в том случае, когда одиночная переменная или результат вычисления выражения может иметь несколько значений, для каждого из которых нужно применить свою функцию. Например, в следующем коде за основу берется система меню PHP, составленная в главе 4, которая переводится в инструкции JavaScript. Эта система работает путем передачи одного строкового значения, соответствующего пожеланию пользователя, коду основного меню. Предположим, что пользователю доступны следующие варианты: Home, About, News, Login и Links — и переменной page присваивается одно из этих значений, соответствующее тому, что ввел пользователь.

Код, созданный для этого с использованием конструкции if...else if..., может иметь вид, показанный в примере 14.13.

**Пример 14.13.** Многострочная конструкция if...else if

```
<script>
if      (page == "Home")  document.write("Вы выбрали Home")
else if (page == "About") document.write("Вы выбрали About")
else if (page == "News")  document.write("Вы выбрали News")
else if (page == "Login") document.write("Вы выбрали Login")
else if (page == "Links") document.write("Вы выбрали Links")
</script>
```

А при использовании конструкции switch код может иметь вид, показанный в примере 14.14.

**Пример 14.14.** Конструкция switch

```
<script>
switch (page)
{
    case "Home":  document.write("Вы выбрали Home")
        break
    case "About": document.write("Вы выбрали About")
        break
    case "News":  document.write("Вы выбрали News")
        break
    case "Login": document.write("Вы выбрали Login")
        break
    case "Links": document.write("Вы выбрали Links")
        break
}
</script>
```

Переменная page здесь присутствует только в самом начале инструкции switch. После чего совпадения проверяются командой case. При совпадении выполняется

условная инструкция. Разумеется, в настоящей программе здесь будет код для отображения страницы или для перехода на нее, а не простое сообщение пользователю о том, что он выбрал.

### Прекращение работы инструкции switch

Рассматривая код примера 14.14, можно заметить, что, как и в PHP, команда `break` позволяет сценарию прекратить работу инструкции `switch` в случае соблюдения условия. Если вы не хотите, чтобы выполнение всех инструкций, начиная со следующей `case`, продолжилось, не забудьте поставить команду `break`.

### Действие по умолчанию

С помощью ключевого слова `default` для инструкции `switch` можно определить действие по умолчанию на тот случай, когда не будет выполнено ни одно из условий. В примере 14.15 показан фрагмент кода, который может быть вставлен в код примера 14.14.

**Пример 14.15.** Инструкция `default`, предназначенная для кода примера 14.14

```
default: document.write("Нераспознанный выбор")
         break
```

## Оператор ?

Трехкомпонентный оператор, состоящий из вопросительного знака (?), применяемого в сочетании с символом двоеточия (:), является упрощенной формой теста `if...else`. Используя этот оператор, можно поставить за вычисляемым выражением знак ? и код, выполняемый в том случае, если выражение вычисляется как `true`. После этого кода ставится знак : и код, который будет выполнен, если выражение будет вычислено как `false`.

В примере 14.16 показан трехкомпонентный оператор, используемый для вывода сообщения о том, что значение переменной `a` меньше или равно 5, или для вывода другого сообщения, если это утверждение не соответствует действительности.

**Пример 14.16.** Использование трехкомпонентного оператора

```
<script>
document.write(
    a <= 5 ?
    "a меньше или равно 5" :
    "a больше 5"
    )
</script>
```

Для более понятного представления этот оператор был разбит на несколько строк, но вы, скорее всего, воспользуетесь его однострочной формой:

```
size = a <= 5 ? "короткий" : "длинный"
```

# Циклы

При рассмотрении циклов нам опять встретится множество параллелей между JavaScript и PHP. В обоих языках поддерживаются циклы while, do...while и for.

## Циклы while

В JavaScript в циклах while сначала проверяется значение выражения, а выполнение инструкций внутри цикла начинается лишь в том случае, если выражение вычисляется как true. Если выражение вычисляется как false, управление переходит к следующей инструкции JavaScript (если таковая имеется).

После завершения итерации цикла выражение опять проверяется на истинность, и процесс продолжается до тех пор, пока не наступит момент, когда выражение будет вычислено как false, или пока выполнение не будет остановлено по какой-нибудь другой причине. Этот цикл показан в примере 14.17.

**Пример 14.17.** Цикл while

```
<script>
counter=0

while (counter < 5)
{
    document.write("Счетчик: " + counter + "<br />")
    ++counter
}
</script>
```

Этот сценарий выведет следующую информацию:

```
Счетчик: 0
Счетчик: 1
Счетчик: 2
Счетчик: 3
Счетчик: 4
```

Если бы переменная counter не увеличивалась на единицу внутри цикла, то вполне возможно, что некоторые браузеры перестали бы откликаться из-за входа в бесконечный цикл и работу со страницей было бы трудно остановить даже с помощью клавиши Esc или кнопки остановки загрузки страницы. Поэтому к циклам в JavaScript нужно относиться с большой осторожностью.

## Циклы do...while

Когда нужен цикл, в котором еще до того, как будет проверено выражение, должна пройти хотя бы одна итерация, используется цикл do...while, который похож на цикл while, за исключением того, что проверка выражения осуществляется только после каждой итерации цикла. Поэтому для вывода первых семи результатов таблицы умножения на 7 можно воспользоваться кодом, показанным в примере 14.18.

**Пример 14.18.** Цикл do...while

```
<script>
count = 1
do
{
    document.write(count + " умножить на 7 равно " + count * 7 + "<br />")
} while (++count <= 7)
</script>
```

Как и ожидалось, этот цикл выведет следующую информацию:

```
1 умножить на 7 равно 7
2 умножить на 7 равно 14
3 умножить на 7 равно 21
4 умножить на 7 равно 28
5 умножить на 7 равно 35
6 умножить на 7 равно 42
7 умножить на 7 равно 49
```

## Циклы for

Цикл for объединяет все лучшие качества организации цикла в одной конструкции, которая позволяет передать каждой инструкции три параметра:

- выражение инициализации;
- выражение условия;
- выражение модификации.

Эти параметры отделяются друг от друга точками с запятыми: for (выражение1; выражение2; выражение3). С началом первой итерации цикла выполняется выражение инициализации. Для кода вывода таблицы умножения на 7 переменная count будет инициализирована значением 1. Затем после каждого прохождения цикла будет проверено выражение условия (в данном случае count <= 7), и новое вхождение в цикл произойдет только в том случае, если выражение условия будет вычислено как true. И в завершение в конце каждой итерации будет вычислено выражение модификации. В случае с таблицей умножения на 7 значение переменной count увеличивается на единицу. В примере 14.19 показан код такого цикла.

**Пример 14.19.** Использование цикла for

```
<script>
for (count = 1 ; count <= 7 ; ++count)
{
    document.write(count + " умножить на 7 равно " + count * 7 + "<br />");
}
</script>
```

Как и в PHP, в первом параметре цикла for можно присваивать значения сразу нескольким переменным, разделяя выражения запятыми:

```
for (i = 1, j = 1 ; i < 10 ; i++)
```

Точно так же в последнем параметре можно осуществлять сразу несколько модификаций:

```
for (i = 1 : i < 10 : i++, --j)
```

Или можно одновременно делать и то и другое:

```
for (i = 1, j = 1 : i < 10 : i++, --j)
```

## Прекращение работы цикла

Команду break, о важности использования которой в инструкции switch вы уже знаете, можно применять и внутри циклов for. Например, она может пригодиться при поиске совпадений определенного вида. Как только совпадение будет найдено, продолжение поиска станет пустой тратой времени и заставит вашего посетителя ждать его завершения. Использование команды break показано в примере 14.20.

**Пример 14.20.** Использование команды break в цикле for

```
<script>
haystack = new Array()
haystack[17] = "Иголка"

for (j = 0 : j < 20 : ++j)
{
    if (haystack[j] == "Иголка")
    {
        document.write("<br />- найдена в элементе " + j)
        break
    }
    else document.write(j + ". ")
}
</script>
```

Этот сценарий выводит следующую информацию:

```
0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16.
- найдена в элементе 17
```

## Инструкция continue

Иногда нужно не выйти из цикла, а пропустить выполнение тех инструкций, которые остались в данной итерации. В таких случаях можно воспользоваться командой continue. Ее применение показано в примере 14.21.

**Пример 14.21.** Использование команды continue в цикле for

```
<script>
haystack     = new Array()
haystack[4]  = "Иголка"
haystack[11] = "Иголка"
haystack[17] = "Иголка"

for (j = 0 : j < 20 : ++j)
```

```
{
    if (haystack[j] == "Иголка")
    {
        document.write("<br />- найдена в элементе " + j + "<br />")
        continue
    }

    document.write(j + ". ")
}
</script>
```

Обратите внимание на то, что второй вызов метода `document.write` не нужно помещать в инструкцию `else` (как было в предыдущем примере), поскольку если будет найдено совпадение, то в результате выполнения команды `continue` данный вызов будет пропущен. Этот сценарий выводит следующую информацию:

```
0. 1. 2. 3.
- найдена в элементе 4
5. 6. 7. 8. 9. 10.
- найдена в элементе 11
12. 13. 14. 15. 16.
  найдена в элементе 17
18. 19.
```

# Явное преобразование типов

В отличие от PHP, в JavaScript нет явного преобразования типов, осуществляемого с помощью операторов `(int)` или `(float)`. Когда нужно, чтобы значение имело определенный тип данных, используется одна из встроенных функций JavaScript, показанных в табл. 14.6.

**Таблица 14.6.** Функции изменения типа, используемые в JavaScript

| Преобразование в тип данных | Используемая функция |
|---|---|
| Int, Integer | parseInt() |
| Bool, Boolean | Boolean() |
| Float, Double, Real | parseFloat() |
| String | String() |
| Array | split() |

Например, чтобы преобразовать число с плавающей точкой в целое число, можно использовать следующий код (который выводит значение 3):

```
n = 3.1415927
i = parseInt(n)
document.write(i)
```

Или можно воспользоваться составной формой:

```
document.write(parseInt(3.1415927))
```

На этом рассмотрение выражений и способов управления процессом выполнения сценариев завершается. В следующей главе будет рассмотрено использование в JavaScript функций, объектов и массивов.

# Проверьте ваши знания

1. Чем отличается обработка булевых значений в PHP от их обработки в JavaScript?
2. В чем разница между унарными, бинарными и трехкомпонентными операторами?
3. Как лучше всего принудительно установить собственный приоритет для оператора?
4. В каком случае следует использовать оператор тождественности (`===`)?
5. Какие две формы выражений считаются самыми простыми?
6. Назовите три типа условных инструкций.
7. Как в инструкциях `if` и `while` интерпретируются условные выражения, в которых используются данные, относящиеся к разным типам?
8. Почему цикл `for` считается мощнее цикла `while`?
9. Для чего предназначена инструкция `with`?
10. Как можно заставить JavaScript изящно разобраться с ошибкой в случае ее возникновения?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 14».

# 15 Функции, объекты и массивы JavaScript

В JavaScript предоставляется практически такой же доступ к функциям и объектам, как и в PHP. Язык JavaScript фактически основан на объектах, поскольку, как вы уже поняли, у него есть доступ к DOM — модели, которая реализует доступ к каждому элементу HTML-документа, позволяя работать с ним как с объектом. Способы применения функций и объектов, а также используемый для этого синтаксис очень похожи на все то, что мы видели в PHP, поэтому при изучении этого материала, а также при углубленном рассмотрении массивов JavaScript вы будете чувствовать себя вполне уверенно.

## Функции JavaScript

В JavaScript наряду с доступом к десяткам встроенных функций (или методов), среди которых и метод `write`, который, как мы уже видели, использовался в вызовах `document.write`, можно создавать и собственные функции. Как только появляется какой-нибудь непростой фрагмент кода с перспективами на многократное использование, он становится кандидатом на оформление в виде функции.

### Определение функции

В общем виде синтаксис функции выглядит следующим образом:

```
function имя_функции ([параметр [, ...]])
{
    инструкции
}
```

Из первой строки синтаксиса следует, что:

- определение начинается со слова `function`;
- следующее за этим словом имя должно начинаться с буквы или символа подчеркивания, за которым следует любое количество букв, цифр, символов доллара или подчеркивания;
- необходимо использовать скобки;
- дополнительно могут применяться один или несколько параметров, разделенных запятыми (о чем свидетельствуют квадратные скобки, не являющиеся частью синтаксиса функции).

Имена функций чувствительны к регистру букв, поэтому строки getInput, GETINPUT и getinput ссылаются на разные функции.

В JavaScript для имен функций действует общепринятое соглашение: первая буква каждого слова в имени, за исключением первой буквы всего имени, должна быть прописной. Поэтому в приведенных примерах имен предпочтение следует отдать имени getInput, имеющему формат, используемый большинством программистов. Это соглашение часто называют bumpyCaps (неровностями из прописных букв) или camelCase (как горбы у верблюда).

Инструкции, которые будут выполняться после вызова функции, начинаются с открывающей фигурной скобки, а составляющая ей пару закрывающая фигурная скобка должна завершать перечень этих инструкций. Среди инструкций обязаны присутствовать одна или несколько инструкций return, которые заставляют функцию прекратить выполнение и вернуть управление вызвавшему ее коду. Если к инструкции return прилагается какое-нибудь значение, то вызывающий код может его извлечь.

## Массив аргументов

Составной частью каждой функции является массив аргументов — arguments. Благодаря ему можно определить количество переменных, переданных функции, и понять, что они собой представляют. Рассмотрим, например, функцию displayItems. Один из способов ее создания показан в примере 15.1.

**Пример 15.1.** Определение функции

```
<script>
displayItems("Собака", "Кошка", "Пони", "Хомяк", "Черепаха")

function displayItems(v1, v2, v3, v4, v5)
{
    document.write(v1 + "<br />")
    document.write(v2 + "<br />")
    document.write(v3 + "<br />")
    document.write(v4 + "<br />")
    document.write(v5 + "<br />")
}
</script>
```

После вызова этого сценария на экране браузера отобразится следующая информация:

```
Собака
Кошка
Пони
Хомяк
Черепаха
```

А что делать, если функции нужно будет передать больше пяти аргументов? К тому же использование вместо цикла многократного вызова метода document.write считается слишком расточительным приемом программирования. К счастью, массив arguments позволяет приспособиться к обработке любого количества аргументов.

В примере 15.2 показана возможность использования этого массива для придания предыдущему примеру более рациональной формы.

**Пример 15.2.** Модификация функции для использования массива аргументов

```
<script>
function displayItems()
{
    for (j = 0 ; j < displayItems.arguments.length ; ++j)
        document.write(displayItems.arguments[j] + "<br />")
}
</script>
```

Обратите внимание на использование свойства `length`, которое уже встречалось в предыдущей главе, а также на то, как с помощью переменной `j`, являющейся индексным смещением внутри массива, осуществляется ссылка на элементы `displayItems.arguments`. Поскольку тело цикла `for` состоит всего из одной инструкции, я решил не заключать его в фигурные скобки, чтобы не загромождать код функции.

Теперь благодаря данной технологии у вас есть функция, способная принимать любое количество аргументов и делать с каждым аргументом все, что вам захочется.

## Возвращение значения

Функции используются не только для отображения информации. Чаще всего они применяются для выполнения вычислений или обработки данных с возвращением полученного результата. Функция `fixNames`, показанная в примере 15.3, задействует массив `arguments` (рассмотренный в предыдущем пункте) для приема переданной ей последовательности строк и возвращения всех этих строк в виде одной строки. Слово `fix` (исправление) в ее имени означает, что она переводит каждый символ в аргументах в нижний регистр, делая исключение для первой буквы каждого аргумента, которую она превращает в прописную.

**Пример 15.3.** Приведение в порядок полного названия

```
<script>
document.write(fixNames("the", "DALLAS", "CowBoys"))

function fixNames()
{
    var s = ""

    for (j = 0 ; j < fixNames.arguments.length ; ++j)
        s += fixNames.arguments[j].charAt(0).toUpperCase() +
             fixNames.arguments[j].substr(1).toLowerCase() + " "

   return s.substr(0, s.length-1)
}
</script>
```

К примеру, если вызвать эту функцию с параметрами the, DALLAS и CowBoys, то она вернет строку The Dallas Cowboys. Проанализируем работу этой функции. Сначала она инициализирует временную (и локальную) переменную s, присваивая ей значение пустой строки. Затем с помощью цикла for осуществляется последовательный перебор каждого переданного параметра с выделением его первой буквы с помощью метода charAt и переводом ее в верхний регистр с помощью метода toUpperCase. Все методы, показанные в этом примере, встроены в JavaScript и доступны по умолчанию.

После этого для извлечения оставшейся части каждой строки используется метод substr, а для перевода букв этой части строки в нижний регистр — метод toLowerCase. Если здесь применить полную версию вызова метода substr, то в ней вторым аргументом нужно указать, из какого количества символов будет состоять извлекаемая подстрока:

```
substr(1, (arguments[j].length) - 1 )
```

Иными словами, в этом вызове метода substr говорится следующее: «Начни с символа, который находится в позиции 1 (это второй символ), и верни оставшуюся часть строки (равную длине строки — length за вычетом одного символа)». Но что особенно приятно, метод substr заранее предполагает, что если второй аргумент опущен, то вам нужна вся оставшаяся часть строки.

После того как весь аргумент будет переделан для получения нужного результата, к нему добавляется пробел и результат присоединяется к значению временной переменной s.

На завершающей стадии к содержимому переменной s опять применяется метод substr. Поскольку последний пробел нам не нужен, мы используем метод substr, чтобы вернуть всю строку, за исключением ее последнего символа.

Этот пример особенно интересен тем, что в нем показано использование в одном выражении сразу нескольких свойств и методов, например:

```
fixNames.arguments[j].substr(1).toLowerCase()
```

Чтобы понять суть этой инструкции, ее нужно мысленно разделить на части, используя в качестве разделителей точки. JavaScript вычисляет эти элементы инструкции слева направо.

1. Берется имя функции: fixNames.
2. Из массива аргументов этой функции извлекается элемент j.
3. К извлеченному элементу применяется метод substr с параметром 1. Благодаря этому следующей части выражения будет передан весь извлеченный элемент, за исключением первого символа.
4. К той строке, которая только что была передана, применяется метод toLowerCase.

Такие построения часто называют *составлением цепочки методов*. Если, к примеру, приведенному здесь выражению передать строку mixedCASE, то она пройдет через следующие преобразования:

```
ixedCase
ixedCase
ixedcase
```

И последнее напоминание: созданная внутри функции переменная s имеет локальную область видимости и не может быть доступна за ее пределами. Возвращая s с помощью инструкции return, мы делаем ее значение доступным вызывавшему функцию коду, который может его сохранить или использовать любым другим способом.

Но с завершением работы функции сама переменная s исчезает. Хотя мы можем заставить функцию работать и с глобальными переменными (что иногда просто необходимо), все-таки лучше просто вернуть те значения, которые нужно сохранить, и дать возможность JavaScript избавиться от всех остальных переменных, которые использовались функцией.

## Возвращение массива

В примере 15.3 функция возвращает только один параметр. А что делать, если нужно вернуть сразу несколько параметров? Решить задачу поможет возвращение массива, показанное в примере 15.4.

**Пример 15.4.** Возвращение массива значений

```
<script>
words = fixNames("the", "DALLAS", "CowBoys")

for (j = 0 ; j < words.length ; ++j)
    document.write(words[j] + "<br />")

function fixNames()
{
    var s = new Array()

    for (j = 0 ; j < fixNames.arguments.length ; ++j)
        s[j] = fixNames.arguments[j].charAt(0).toUpperCase() +
               fixNames.arguments[j].substr(1).toLowerCase()

    return s
}
</script>
```

Здесь переменная words автоматически определяется в виде массива и заполняется результатами, возвращенными вызовом функции fixNames. Затем цикл for осуществляет последовательный перебор элементов массива, отображая каждый элемент.

Что касается функции fixNames, то она практически идентична использованной в примере 15.3, за исключением того, что переменная s теперь является массивом, возвращаемым с помощью инструкции return.

Эта функция позволяет извлекать отдельные параметры из возвращенных ею элементов, например, как в следующем коде (который выводит строку The Cowboys):

```
words = fixNames("the", "DALLAS", "CowBoys")
document.write(words[0] + " " + words[2])
```

# Объекты JavaScript

По сравнению с переменными, которые в каждый конкретный момент могут содержать только одно значение, объекты JavaScript — это значительный шаг вперед, поскольку в них могут содержаться несколько значений и даже функций. Объект группирует вместе данные и функции для работы с ними.

## Объявление класса

При создании сценария, в котором используются объекты, необходимо спроектировать структуру из данных и кода, называемую *классом*. Каждый новый объект, основанный на конкретном классе, называется *экземпляром* класса. Вам уже известно, что данные, связанные с объектом, называются его *свойствами*, а функции, которые им используются, называются *методами*.

Посмотрим, как объявляется класс для объекта по имени User, который будет содержать сведения о текущем пользователе. Для создания класса нужно просто создать функцию с именем этого класса.

Эта функция может воспринимать аргументы (позже будет показано, как она вызывается) и создавать свойства и методы для объектов класса. Такая функция называется *конструктором*.

В примере 15.5 показан конструктор для класса User, имеющий три свойства: имя — forename, пользовательское имя — username и пароль — password. В классе также определяется метод демонстрации сведений о пользователе — showUser.

**Пример 15.5.** Объявление класса User и его метода

```
<script>
function User(forename, username, password)

    this.forename = forename
    this.username = username
    this.password = password

    this.showUser = function()
    {
        document.write("Имя: " + this.forename + "<br />")
        document.write("Пользовательское имя: " + this.username + "<br />")
        document.write("Пароль: " + this.password + "<br />")
    }

</script>
```

От ранее рассмотренных эту функцию отличают две особенности.

- Она ссылается на объект по имени this. Когда программа благодаря запуску этой функции создает экземпляр класса User, объект this является ссылкой на создаваемый экземпляр. Одна и та же функция может быть многократно вызвана с разными аргументами, всякий раз создавая новый экземпляр класса User с разными значениями для свойств forename и т. д.

- Внутри этой функции создается новая функция по имени showUser. Здесь показан новый, усложненный синтаксис. Его задача — привязать showUser к классу User. Таким образом, showUser становится методом класса User.

Здесь используется соглашение о выборе имен, согласно которому все свойства получают имена, состоящие из букв в нижнем регистре, а в имени метода в соответствии с упомянутым в данной главе соглашением bumpyCaps есть по крайней мере одна прописная буква.

В примере 15.5 соблюдается рекомендуемый способ создания конструктора класса, который заключается в том, что методы включаются в функции конструктора. Но в примере 15.6 показано, что можно также ссылаться и на те функции, которые определены за пределами конструктора.

**Пример 15.6.** Раздельное объявление класса и метода

```
<script>
function User(forename, username, password)
{
    this.forename = forename
    this.username = username
    this.password = password
    this.showUser = showUser
}

function showUser()
{
    document.write("Имя: " + this.forename + "<br />")
    document.write("Пользовательское имя: " + this.username + "<br />")
    document.write("Пароль: " + this.password + "<br />")
}
</script>
```

Эта форма объявления класса показана с расчетом на то, что вам наверняка придется сталкиваться с использованием кода, созданного другими программистами.

# Создание объекта

Для создания экземпляра класса User можно воспользоваться следующей инструкцией:

```
details = new User("Wolfgang", "w.a.mozart", "composer")
```

Или же можно создать пустой объект:

```
details = new User()
```

а затем наполнить его содержимым:

```
details.forename = "Wolfgang"
details.username = "w.a.mozart"
details.password = "composer"
```

К объекту также можно добавлять новые свойства:

```
details.greeting = "Привет"
```

Проверить работу только что добавленного свойства можно с помощью следующей инструкции:

```
document.write(details.greeting)
```

## Доступ к объектам

Для доступа к объекту можно сослаться на его свойства, как показано в следующих не связанных друг с другом примерах инструкций:

```
name = details.forename
if (details.username == "Admin") loginAsAdmin()
```

А для доступа к методу showUser, принадлежащему объекту класса User, нужно воспользоваться следующим синтаксисом, в котором используется уже созданный и заполненный данными объект details:

```
details.showUser()
```

В соответствии с ранее предоставленными объекту данными этот код отобразит следующую информацию:

```
Имя: Wolfgang
Пользовательское имя: w.a.mozart
Пароль: composer
```

## Ключевое слово prototype

Использование ключевого слова prototype позволяет добиться существенной экономии оперативной памяти. Каждый экземпляр класса User будет содержать три свойства и один метод. Поэтому, если в памяти содержится тысяча таких объектов, метод showUser также будет растиражирован тысячу раз. Но, поскольку в каждом экземпляре присутствует один и тот же метод, можно предписать новому объекту ссылаться на единственный экземпляр этого метода и не создавать его копию. Итак, вместо использования в конструкторе класса строки кода:

```
this.showUser = function()
```

можно воспользоваться следующей строкой:

```
User.prototype.showUser = function()
```

Код обновленного конструктора показан в примере 15.7.

**Пример 15.7.** Объявление класса с использованием для метода ключевого слова prototype

```
<script>
function User(forename, username, password)
{
```

```
    this.forename = forename
    this.username = username
    this.password = password

    User.prototype.showUser = function()
    {
        document.write("Имя: " + this.forename + "<br />")
        document.write("Пользовательское имя: " + this.username + "<br />")
        document.write("Пароль: " + this.password + "<br />")
    }
}
</script>
```

Этот код работает благодаря тому, что у всех функций имеется свойство по имени prototype, разработанное для хранения свойств и методов, не тиражируемых в каждом объекте, создаваемом на основе класса. Вместо этого они передаются объектам данного класса по ссылке.

Это означает, что свойства или методы prototype могут быть добавлены в любое время и они будут унаследованы всеми объектами (даже теми, которые уже были созданы), что можно проиллюстрировать следующими инструкциями:

```
User.prototype.greeting = "Привет"
document.write(details.greeting)
```

Первая инструкция добавляет к классу User нрототинное свойство prototype.greeting, имеющее значение Привет. Во второй строке уже созданный объект details вполне корректно отображает это новое свойство.

Можно также добавлять к классу методы или вносить в них изменения, как показано в следующих инструкциях:

```
User.prototype.showUser = function() { document.write("Имя " +
this.forename + " Пользователь " + this.username + " Пароль " + this.password) }
details.showUser()
```

Эти строки можно поместить в свой сценарий, в инструкцию условия (например, в if), чтобы они запускались только в том случае, когда действия пользователя наталкивают на принятие решения о применении другого метода showUser. После запуска этих строк кода даже для уже созданного объекта details при всех последующих вызовах метода details.showUser будет запускаться новая версия, а старое определение showUser будет стерто.

## Статические методы и свойства

При изучении объектов PHP вы узнали, что у классов могут быть статические свойства и методы, а также свойства и методы, связанные с конкретным экземпляром класса. JavaScript также поддерживает статические свойства и методы, которые легко и просто могут сохраняться в принадлежащие классу прототипы и извлекаться из них. Следующие инструкции устанавливают в класс User и считывают из него статическую строку:

```
User.prototype.greeting = "Привет"
document.write(User.prototype.greeting)
```

## Расширение объектов JavaScript

Ключевое слово prototype позволяет даже добавлять функциональные возможности встроенным объектам. Предположим, к примеру, что нужно добавить возможность замены всех пробелов в строке неразрываемыми пробелами, чтобы избежать переноса ее части на новую строку. Это можно сделать добавлением к имеющемуся в JavaScript определению исходного объекта String прототипного метода:

```
String.prototype.nbsp =
    function() { return this.replace(/ /g, ' ') }
```

В коде этого метода для поиска всех одиночных пробелов и замены их строкой   используются метод replace и регулярное выражение (см. главу 16). Если после запуска этого кода будет введена следующая команда:

```
document.write("Шустрая бурая лиса".nbsp())
```

то в результате ее работы будет выведена следующая строка: Шустрая бурая лиса. Посмотрите также на метод, который можно добавить для удаления всех пробелов, с которых начинается и которыми заканчивается строка (в нем опять используется регулярное выражение):

```
String.prototype.trim =
    function() { return this.replace(/^\s+|\s+$/g, '') }
```

Если выдать следующую инструкцию, то на выходе будет получена строка Пожалуйста, избавьте меня от лишних пробелов (то есть из нее будут удалены все начальные и замыкающие пробелы):

```
document.write("  Пожалуйста, избавьте меня от лишних пробелов    ".trim())
```

Если разбить выражение на составные части, то два символа / помечают начало и конец выражения, а завершающий символ g задает глобальный поиск. Внутри выражения его часть ^\s+ задает поиск одного или нескольких пробельных символов применительно к началу строки, в которой ведется поиск, а его часть \s+$ задает поиск одного или нескольких пробельных символов применительно к концу строки, в которой ведется поиск. Расположенный в середине символ | служит разделителем альтернативных вариантов регулярного выражения.

В результате при соответствии любого из этих выражений соответствующая часть заменяется пустой строкой, возвращая тем самым усеченную версию строки без лидирующих и замыкающих пустых пространств.

# Массивы в JavaScript

Работа с массивами в JavaScript очень напоминает работу с ними в PHP, хотя синтаксис имеет несколько иной вид. Тем не менее с учетом уже приобретенных знаний о массивах освоить материал этого раздела будет относительно несложно.

# Числовые массивы

Чтобы создать новый массив, нужно воспользоваться следующим синтаксисом:

```
arrayname = new Array()
```

или же его более краткой формой:

```
arrayname = []
```

## Присваивание значений элементам массива

В PHP можно было добавить к массиву новый элемент простым присваиванием ему значения, без указания смещения элемента относительно начала массива:

```
$arrayname[] = "Элемент 1";
$arrayname[] = "Элемент 2";
```

В JavaScript для этих же целей используется метод push:

```
arrayname.push("Элемент 1")
arrayname.push("Элемент 2")
```

Он позволяет добавлять к массиву элементы, не отслеживая их количество. Когда потребуется узнать, сколько элементов содержится в массиве, можно будет воспользоваться свойством length:

```
document.write(arrayname.length)
```

Если нужно будет проконтролировать размещение элементов, расставляя их по конкретным местам, можно воспользоваться другим синтаксисом:

```
arrayname[0] = "Элемент 1"
arrayname[1] = "Элемент 2"
```

В примере 15.8 показан простой сценарий, в котором создается массив, в него загружается несколько элементов, после чего эти элементы отображаются на экране.

**Пример 15.8.** Создание, построение и вывод массива на экран

```
<script>
numbers = []
numbers.push("Один")
numbers.push("Два")
numbers.push("Три")

for (j = 0 ; j < numbers.length ; ++j)
    document.write("Элемент " + j + " = " + numbers[j] + "<br />")
</script>
```

Этот сценарий выводит следующую информацию:

```
Элемент 0 = Один
Элемент 1 = Два
Элемент 2 = Три
```

## Присваивание с использованием ключевого слова array

С помощью ключевого слова Array можно также создать массив с несколькими исходными элементами:

```
numbers = Array("Один", "Два", "Три")
```

После этого ничто не помешает добавить к данному массиву дополнительные элементы.

Теперь в вашем распоряжении есть несколько способов добавления элементов к массиву и один способ ссылки на них, но JavaScript предлагает куда более обширный арсенал способов, к рассмотрению которых мы скоро перейдем. Но сначала рассмотрим еще один тип массива.

## Ассоциативные массивы

К *ассоциативным* относятся такие массивы, в которых ссылки на элементы осуществляются по именам, а не по числовому смещению. Чтобы создать ассоциативный массив, нужно определить блок элементов, заключенный в фигурные скобки. Для каждого элемента слева от двоеточия (:) указывается его ключ, а справа — содержимое. В примере 15.9 показано, как можно создать ассоциативный массив для хранения данных о товаре в разделе мячей (balls) интернет-магазина спортивного инвентаря.

**Пример 15.9.** Создание и отображение ассоциативного массива

```
<script>
balls = {"гольф":     "Мячи для гольфа, 6",
         "теннис":    "Мячи для тенниса, 3",
         "футбол":    "Футбольный мяч, 1",
         "пинг-понг": "Мячи для пинг-понга, 12 шт."}

for (ball in balls)
    document.write(ball + " = " + balls[ball] + "<br />")
</script>
```

Для проверки факта создания и заполнения массива я воспользовался еще одной разновидностью цикла `for`, в которой применяется ключевое слово `in`. В этом цикле создается новая переменная, которая задействуется только внутри массива (в данном примере — `ball`), и вызывается последовательный перебор всех элементов массива, указанных справа от ключевого слова `in` (в данном примере — `balls`). Цикл обрабатывает каждый элемент массива `balls`, помещая значение ключа в переменную `ball`.

Используя значение ключа, сохраненное в переменной `ball`, можно также получить значение текущего элемента массива `balls`. Результат вызова сценария этого примера в браузере будет иметь следующий вид:

```
гольф = Мячи для гольфа, 6
теннис = Мячи для тенниса, 3
футбол = Футбольный мяч, 1
пинг-понг = Мячи для пинг-понга, 12 шт.
```

Чтобы получить значение конкретного элемента ассоциативного массива, нужно в явном виде указать его ключ (в данном случае будет выведено значение `Футбольный мяч, 1`):

```
document.write(balls['футбол'])
```

## Многомерные массивы

В JavaScript для создания многомерного массива нужно просто поместить массивы внутри других массивов. Например, чтобы создать массив, содержащий сведения о двумерной шахматной доске (8 × 8 клеток), можно воспользоваться кодом примера 15.10.

**Пример 15.10.** Создание многомерного числового массива

```
<script>
checkerboard = Array(
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array('o', ' ', 'o', ' ', 'o', ' ', 'o', ' '),
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array('O', ' ', 'O', ' ', 'O', ' ', 'O', ' '),
    Array(' ', 'O', ' ', 'O', ' ', 'O', ' ', 'O'),
    Array('O', ' ', 'O', ' ', 'O', ' ', 'O', ' '))

document.write("<pre>")

for (j = 0 ; j < 8 ; ++j)
{
    for (k = 0 ; k < 8 ; ++k)
        document.write(checkerboard[j][k] + " ")
    document.write("<br />")
}
document.write("</pre>")
</script>
```

В данном примере буквами нижнего регистра обозначены черные, а буквами верхнего регистра — белые фигуры. Два цикла `for`, один из которых является вложенным, осуществляют последовательный перебор элементов массива и отображают его содержимое.

Внешний цикл содержит две инструкции, поэтому они заключены в фигурные скобки. Внутренний цикл обрабатывает каждую клетку в горизонтали, выводя символ, находящийся в позиции `[j][k]`, за которым следует пробел (чтобы придать выводимой информации квадратную конфигурацию). В этом цикле содержится всего одна инструкция, поэтому заключать ее в фигурные скобки не имеет смысла. Теги `<pre>` и `</pre>` обеспечивают корректный вывод информации:

```
  o   o   o   o
o   o   o   o
  o   o   o   o

0   0   0   0
  0   0   0   0
0   0   0   0
```

Можно также получить непосредственный доступ к любому элементу данного массива, применив для этого квадратные скобки:

```
document.write(checkerboard[7][2])
```

Эта инструкция выводит букву O верхнего регистра, то есть содержимое восьмой сверху и третьей справа клетки — напоминаю, что индексация элементов в массиве начинается с нуля, а не с единицы.

# Использование методов массивов

Реализовать возможности, предоставленные массивами, помогают имеющиеся в JavaScript готовые к использованию методы для работы с ними и с содержащимися в них данными. Рассмотрим подборку, состоящую из наиболее востребованных методов.

## Метод concat

Метод concat объединяет два массива или ряд значений в массив. Например, следующий код выведет Банан,Виноград,Морковь,Капуста:

```
fruit = ["Банан", "Виноград"]
veg   = ["Морковь", "Капуста"]
document.write(fruit.concat(veg))
```

В качестве аргументов можно указать несколько массивов, тогда метод concat добавит все их элементы в порядке указания массивов.

А вот еще один способ использования метода concat, где с массивом pets объединяются простые значения и на экран выводится строка Кошка,Собака,Рыба,Кролик,Хомяк:

```
pets      = ["Кошка", "Собака", "Рыба"]
more_pets = pets.concat("Кролик", "Хомяк")
document.write(more_pets)
```

## Метод forEach (для браузеров не из семейства IE)

Используемый в JavaScript метод forEach является еще одним способом получения функциональных возможностей, аналогичных тем, которые предоставляются ключевым словом PHP foreach, но он работает только в тех браузерах, которые не относятся к семейству Internet Explorer. Воспользоваться этим методом можно, передав ему имя функции, которая будет вызвана для каждого элемента массива. Как это делается, показано в примере 15.11.

Пример 15.11. Использование метода forEach

```
<script>
pets = ["Кошка", "Собака", "Кролик", "Хомяк"]
pets.forEach(output)

function output(element, index, array)
{
    document.write("Элемент с индексом " + index + " содержит значение " +
        element + "<br />")
}
</script>
```

В данном случае функция, передаваемая методу forEach, называется output. Она воспринимает три параметра: элемент, его индекс и массив. Как они используются,

зависит от потребностей вашей функции. В данном примере они просто отображают значения индекса и элемента с помощью метода document.write.

После того как массив будет заполнен, можно вызвать рассматриваемый метод:

```
pets.forEach(output)
```

На выходе будет получена следующая информация:

```
Элемент с индексом 0 содержит значение Кошка
Элемент с индексом 1 содержит значение Собака
Элемент с индексом 2 содержит значение Кролик
Элемент с индексом 3 содержит значение Хомяк
```

## Метод forEach (кросс-браузерное решение)

Разумеется, по уже сложившейся традиции Microsoft решила не поддерживать метод forEach, поэтому предыдущий пример будет работать только на браузерах, не принадлежащих семейству Internet Explorer. Чтобы обеспечить кросс-браузерную совместимость, в то время как IE не поддерживает эту функцию, вместо pets.forEach(output) нужно воспользоваться следующей инструкцией:

```
for (j = 0 ; j < pets.length ; ++j) output(pets[j], j)
```

## Метод join

Метод join позволяет превратить все значения массива в строки, а затем объединить их в одну большую строку, расставляя между значениями необязательные разделители. В примере 15.12 показаны три способа использования этого метода.

**Пример 15.12.** Использование метода join

```
<script>
pets = ["Кошка", "Собака", "Кролик", "Хомяк"]
document.write(pets.join()      + "<br />")
document.write(pets.join(' ')   + "<br />")
document.write(pets.join(' : ') + "<br />")
</script>
```

Если не указывать параметр, метод join использует в качестве разделителя элементов запятую, в противном случае между элементами вставляется переданная методу join строка. Код примера 15.12 выводит следующую информацию:

```
Кошка,Собака,Кролик,Хомяк
Кошка Собака Кролик Хомяк
Кошка : Собака : Кролик : Хомяк
```

## Методы push и pop

Применение метода push для вставки значения в массив уже было рассмотрено. Противоположным ему по действию является метод pop. Он удаляет последний вставленный элемент из массива и возвращает значение этого элемента. Порядок его использования показан в примере 15.13.

**Пример 15.13.** Использование методов push и pop

```
<script>
sports = ["Футбол", "Теннис", "Бейсбол"]
```

```
document.write("Изначально = "      + sports +  "<br />")
sports.push("Hockey")
document.write("После вставки = " + sports +  "<br />")
removed = sports.pop()
document.write("После удаления = "  + sports +  "<br />")
document.write("Удаленный элемент = "    + removed + "<br />")
< script>
```

Три основные инструкции этого сценария выделены полужирным шрифтом. Сначала в сценарии создается массив по имени sports, содержащий три элемента, затем в него вставляется четвертый элемент, после чего сценарий удаляет этот элемент. В процессе этих действий с помощью метода document.write отображаются разные значения массива. Сценарий выводит следующую информацию:

```
Изнчально = Футбол,Теннис,Бейсбол
После вставки = Футбол,Теннис,Бейсбол,Хоккей
После удаления = Футбол,Теннис,Бейсбол
Удаленный элемент = Хоккей
```

Как показано в примере 15.14, методы push и pop применяются в тех случаях, когда нужно отвлечься от каких-нибудь действий на другие, а затем вернуться к прежним действиям.

**Пример 15.14.** Использование методов push и pop внутри цикла и за его пределами

```
<script>
numbers = []

for (j=0 ; j<3 ; ++j)
{
    numbers.push(j);
    document.write("Вставлен элемент " + j + "<br />")
}

// Здесь осуществляются какие-нибудь другие действия
document.write("<br />")

document.write("Удален элемент" + numbers.pop() + "<br />")
document.write("Удален элемент " + numbers.pop() + "<br />")
document.write("Удален элемент " + numbers.pop() + "<br />")
</script>
```

Код этого примера выведет следующую информацию:

```
Вставлен элемент 0
Вставлен элемент 1
Вставлен элемент 2

Удален элемент 2
Удален элемент 1
Удален элемент 0
```

## Использование метода reverse

Метод reverse осуществляет простую перестановку элементов массива в обратном порядке. Его действие показано в примере 15.15.

**Пример 15.15.** Использование метода reverse

```
<script>
sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
sports.reverse()
document.write(sports)
</script>
```

Исходный массив подвергается изменению, и сценарий выводит следующую информацию:

```
Хоккей,Бейсбол,Теннис,Футбол
```

## Метод sort

Метод sort позволяет расставить все элементы массива в алфавитном или в каком-нибудь другом порядке в зависимости от применяемых параметров. В примере 15.16 показаны четыре типа сортировки.

**Пример 15.16.** Использование метода sort

```
<script>
// Сортировка по алфавиту
sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
sports.sort()
document.write(sports + "<br />")

// Сортировка по алфавиту в обратном порядке
sports = ["Футбол", "Теннис", "Бейсбол", "Хоккей"]
sports.sort().reverse()
document.write(sports + "<br />")

// Сортировка чисел по возрастанию
numbers = [7, 23, 6, 74]
numbers.sort(function(a,b){return a - b})
document.write(numbers + "<br />")

// Сортировка чисел по убыванию
numbers = [7, 23, 6, 74]
numbers.sort(function(a,b){return b - a})
document.write(numbers + "<br />")
</script>
```

В первом из четырех блоков этого примера применяется *сортировка по алфавиту*, во втором — возвращение к исходному виду, а затем метод reverse, чтобы получить *сортировку по алфавиту в обратном порядке*.

Третий и четвертый блоки усложнены использованием функции для сравнения взаимоотношений между a и b. У нее отсутствует имя, поскольку она используется только при сортировке. Функция по имени function, которая применяется для

создания анонимных функций, уже встречалась при определении метода класса (метода showUser).

Здесь function создает анонимную функцию, отвечающую запросам метода sort. Если функция возвращает значение больше нуля, сортировка предполагает, что a ставится перед b. Если функция возвращает значение меньше нуля, сортировка предполагает, что b ставится перед a. Если возвращается нуль, порядок следования a и b остается неизменным, поскольку они равны друг другу. Сортировка запускает эту функцию применительно ко всем значениям массива для определения порядка их следования.

За счет манипуляции возвращаемыми значениями (a - b или b - a) в третьем и четвертом блоках примера 15.16 осуществляется выбор между сортировкой чисел по возрастанию и по убыванию.

И на этом я заканчиваю введение в JavaScript. Теперь у вас должно сложиться представление о трех из четырех основных технологиях, рассматриваемых в данной книге. В следующей главе будут рассмотрены некоторые современные технические приемы, основанные на применении всех этих технологий, в частности проверка соответствия шаблонам и проверка допустимости введенных значений.

# Проверьте ваши знания

1. Обладают ли имена функций и переменных в JavaScript чувствительностью к регистру используемых в них букв?
2. Как создать функцию, которая воспринимает и обрабатывает неограниченное количество параметров?
3. Назовите способ возвращения из функции сразу нескольких значений.
4. Какое ключевое слово для ссылки на текущий объект используется при определении класса?
5. Должны ли все методы класса определяться внутри определения самого класса?
6. Какое ключевое слово применяется для создания объекта?
7. Как обеспечить доступность свойства или метода всем объектам класса без его тиражирования внутри объекта?
8. Как создать многомерный массив?
9. Какой синтаксис используется для создания ассоциативного массива?
10. Создайте инструкцию для сортировки массива чисел в убывающем порядке.

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 15».

# 16 Проверка данных и обработка ошибок в JavaScript и PHP

После приобретения основательных знаний по программированию на PHP и JavaScript настало время объединить эти две технологии воедино. В этой главе будет рассмотрено создание максимально удобных для пользователей веб-форм.

PHP будет использоваться для создания форм, а JavaScript – для проверки приемлемости и полноты данных на стороне клиента, насколько это возможно до их отправки на сервер. После чего окончательная проверка приемлемости введенных данных будет выполняться программой PHP, которая при необходимости снова выведет форму, чтобы пользователь мог внести в нее изменения.

В процессе изложения данной главы будут рассмотрены проверка данных и применение регулярных выражений как в JavaScript, так и в PHP.

## Проверка данных, введенных пользователем, средствами JavaScript

Проверка данных средствами JavaScript должна рассматриваться в качестве помощи пользователям, а не веб-сайтам, поскольку, как я уже неоднократно подчеркивал, нельзя доверять абсолютно ничему, что отправлено на ваш сервер, даже если предположить, что полученные данные проверены с помощью JavaScript. Дело в том, что взломщики могут без особых усилий создать имитацию ваших веб-форм и отправить любые нужные им данные.

Еще одна причина, не позволяющая полагаться на JavaScript при проверке введенных данных, заключается в том, что некоторые пользователи отключают JavaScript или используют браузеры, не поддерживающие этот язык.

Поэтому лучшее, что можно сделать при проверке данных средствами JavaScript, это выполнить проверку информационного наполнения тех полей, которые не должны оставаться пустыми, обеспечить приведение адресов электронной почты в надлежащий формат и гарантировать то, что введенные значения находятся в пределах ожидаемых границ.

# Документ validate.html (часть первая)

Рассмотрим стандартную регистрационную форму, используемую большинством веб-сайтов, на которых работать можно только зарегистрированным пользователям. В форме будут запрашиваться имя, фамилия, пользовательское имя, пароль, возраст и адрес электронной почты. В примере 16.1 показан шаблон, который можно применять для этой формы.

**Пример 16.1.** Форма с проверкой данных средствами JavaScript (часть первая)

```
<html><head><title>Пример формы</title>
<style>.signup { border: 1px solid #999999;
    font: normal 14px helvetica; color:#444444; }</style>

<script>
function validate(form) {
    fail  = validateForename(form.forename.value)
    fail += validateSurname(form.surname.value)
    fail += validateUsername(form.username.value)
    fail += validatePassword(form.password.value)
    fail += validateAge(form.age.value)
    fail += validateEmail(form.email.value)
    if (fail == "") return true
    else { alert(fail); return false }
}
</script></head><body>

<table class="signup" border="0" cellpadding="2"
    cellspacing="5" bgcolor="#eeeeee">
<th colspan="2" align="center">Регистрационная форма</th>
<form method="post" action="adduser.php"
    onSubmit="return validate(this)">
     <tr><td>Имя</td><td><input type="text" maxlength="32"
    name="forename" /></td>
</tr><tr><td>Фамилия</td><td><input type="text" maxlength="32"
    name="surname" /></td>
</tr><tr><td>Пользовательское имя</td><td><input type="text" maxlength="16"
    name="username" /></td>
</tr><tr><td>Пароль</td><td><input type="text" maxlength="12"
    name="password" /></td>
</tr><tr><td>Возраст</td><td><input type="text" maxlength="3"
    name="age" /></td>
</tr><tr><td>Электронный адрес</td><td><input type="text" maxlength="64"
    name="email" /></td>
</tr><tr><td colspan="2" align="center">
    <input type="submit" value="Зарегистрироваться" /></td>
</tr></form></table>
```

В данном виде эта форма будет только отображаться, но не сможет заниматься самопроверкой, поскольку к ней еще не добавлены основные проверочные функции. Но, несмотря на это, если набрать данный код, сохранить его в файле

validate.html, а затем вызвать файл в браузере, будет получен результат, показанный на рис. 16.1.

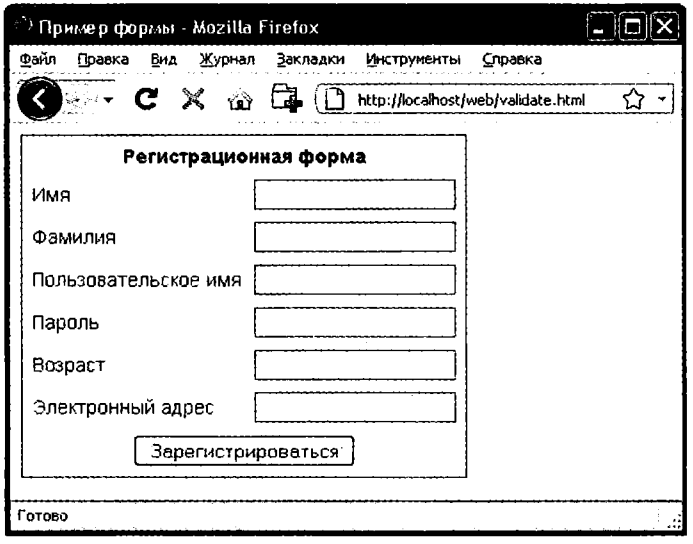



**Рис. 16.1.** Форма, выведенная кодом примера 16.1

## Порядок работы этого кода

Рассмотрим, из чего состоит этот документ. В первых трех строках осуществляется настройка документа и используется небольшой фрагмент кода CSS, предназначенный для улучшения внешнего вида формы. Затем следует выделенная полужирным шрифтом часть документа, относящаяся к JavaScript.

В теги <script> и </script> заключена всего одна функция по имени validate, которая, в свою очередь, вызывает шесть других функций, проверяющих каждое из имеющихся в форме полей. Все они вскоре будут рассмотрены. А сейчас я ограничусь объяснением того, что они возвращают либо пустую строку, если поле проходит проверку, либо сообщение об ошибке, если оно эту проверку не проходит. При наличии любых ошибок сообщения о них выводятся в окне предупреждения, появляющемся благодаря последней строке сценария.

Если поле проходит проверку, то проводившая ее функция возвращает значение true, а если не проходит — false. Значения, возвращаемые функцией validate, учитываются при отправке данных формы: если она возвращает false, данные не отправляются. При этом пользователь получает возможность закрыть появившееся окно предупреждения и внести изменения в данные. Если будет возвращено значение true, значит, ошибок в полях формы не найдено и форму можно отправлять на сервер.

Во второй части этого примера показан код HTML для формы, где каждое поле и его имя помещены в отдельную строку таблицы. В этом HTML нет ничего сложного, за исключением инструкции onSubmit="return validate(this)", помещенной в открывающий тег <form>. Использование атрибута onSubmit позволяет при отправ-

ке формы вызвать избранную вами функцию. Эта функция может выполнить проверку и вернуть значение либо true, либо false, для того чтобы известить о том, разрешена или нет отправка формы.

Параметр this указывает на текущий объект (то есть на данную форму). Он передается только что рассмотренной функции validate, которая получает этот параметр в виде объекта form.

Как видите, внутри HTML-формы JavaScript используется только для того, чтобы вызвать инструкцию return, помещенную в атрибут onSubmit. Браузеры, у которых JavaScript отключен или не поддерживается, просто проигнорируют атрибут onSubmit и беспрепятственно отобразят HTML.

## Документ validate.html (часть вторая)

Теперь обратимся к коду примера 16.2, содержащему набор из шести функций, осуществляющих проверку полей формы. Я предлагаю набрать весь код этой второй части и добавить его к первой половине, которая уже сохранена в файле validate.html. Включение в один и тот же HTML-файл сразу нескольких <script>-разделов вполне допустимо. Если хотите, можете объединить дополнительный код с первым <script>-разделом, показанным в примере 16.1.

**Пример 16.2.** Вторая часть формы, проверяемой средствами JavaScript

```
<script>
function validateForename(field) {
    if (field == "") return "Не введено имя.\n"
    return ""
}

function validateSurname(field) {
    if (field == "") return "Не введена фамилия.\n"
    return ""
}

function validateUsername(field) {
    if (field == "") return "Не введено имя пользователя.\n"
    else if (field.length < 5)
        return "В имени пользователя должно быть не менее 5 символов.\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))
        return "В имени пользователя разрешены только a-z, A-Z, 0-9, - и _.\n"
    return ""
}

function validatePassword(field) {
    if (field == "") return "Не введен пароль.\n"
    else if (field.length < 6)
        return "В пароле должно быть не менее 6 символов.\n"
    else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) ||
             !/[0-9]/.test(field))
        return "Пароль требует 1 символа из каждого набора a-z, A-Z и 0-9.\n"
    return ""
```

```
}

function validateAge(field) {
    if (isNaN(field)) return "Не введен возраст.\n"
    else if (field < 18 || field > 110)
        return "Возраст должен быть между 18 и 110.\n"
    return ""
}

function validateEmail(field) {
    if (field == "") return "Не введен адрес электронной почты.\n"
        else if (!((field.indexOf(".") > 0) &&
                   (field.indexOf("@") > 0)) ||
                  /[^a-zA-Z0-9.@_-]/.test(field))
        return "Электронный адрес имеет неверный формат.\n"
    return ""
}
</script></body></html>
```

Чтобы понять, как работает проверка, рассмотрим по очереди все эти функции, начиная с `validateForename`.

## Проверка имени

Предельно лаконичная функция `validateForename` воспринимает параметр `field`, являющийся значением имени (`forename`), переданным ей функцией `validate`. Если это значение является пустой строкой, возвращается сообщение об ошибке, если нет, возвращается пустая строка, свидетельствующая о том, что ошибка не обнаружена.

Если пользователь введет в это поле пробелы, то они будут приняты функцией `validateForename`, хотя в качестве имени они не годятся. Этот просчет можно исправить, добавив еще одну инструкцию, удаляющую из поля пустые пространства перед его проверкой на незаполненность, затем воспользоваться регулярным выражением, чтобы убедиться в том, что в поле находится еще что-нибудь, кроме пробелов, или — как это сделано в данном случае — позволить пользователю допустить эту ошибку и «отловить» ее на сервере.

## Проверка фамилии

Код функции `validateSurname` похож на код функции `validateForename`, он также возвращает сообщение об ошибке, если в качестве фамилии (`surname`) была предоставлена пустая строка. Я решил не накладывать ограничений на символы обоих полей, чтобы пользователь мог вводить символы, не входящие в английский алфавит, имеющие дополнительные знаки и т. д.

## Проверка имени пользователя

Код функции `validateUsername` немного интереснее, поскольку он выполняет более сложную работу. Он должен разрешить использование только тех символов, которые входят в набор `a-z`, `A-Z`, `0-9`, `_` и `-`, и гарантировать, что имена пользователей состоят не менее чем из пяти символов.

Код структуры `if...else` начинается с возвращения сообщения об ошибке в том случае, если поле не было заполнено. Если значение поля не является пустой строкой, но состоит менее чем из пяти символов, то возвращается другое сообщение об ошибке.

Затем вызывается JavaScript-функция `test`, которая сравнивает регулярное выражение (соответствующее любому символу, не входящему в перечень разрешенных) с содержимым поля (см. раздел «Регулярные выражения» данной главы). Встретив хотя бы один недопустимый символ, функция `test` возвращает `true`, в результате чего функция `validateUsername` возвращает сообщение об ошибке.

## Проверка пароля

Такая же технология используется и в функции `validatePassword`. Сначала функция проверяет поле на пустоту, возвращая сообщение об ошибке при незаполненном поле. Затем сообщение об ошибке возвращается в том случае, если пароль короче шести символов.

Одно из требований, предъявляемых к паролям, заключается в том, что в них должно быть хотя бы по одному символу в нижнем и в верхнем регистре, а также хотя бы одна цифра, поэтому функция `test` вызывается три раза, по одному разу на каждую из этих проверок. Если при любом из таких вызовов будет возвращено значение `false`, это будет означать, что одно из условий не выполнено, поэтому будет возвращено сообщение об ошибке. В противном случае будет возвращена пустая строка, свидетельствующая о том, что с паролем все в порядке.

## Проверка возраста

Функция `validateAge` возвращает сообщение об ошибке, если значение поля не является числом (что определяется вызовом функции `isNaN`) либо введенный возраст меньше 18 или больше 110 лет. У ваших приложений могут быть иные требования к возрастной категории или вообще не быть никаких требований. В случае успешной проверки также будет возвращена пустая строка.

## Проверка адреса электронной почты

И последняя, наиболее сложная проверка — адреса электронной почты — выполняется с помощью функции `validateEmail`. После проверки на существование каких-нибудь введенных данных и возвращения сообщения об ошибке при отсутствии таковых функция дважды вызывает JavaScript-функцию `indexOf`. При первом вызове проверяется наличие точки (`.`), начиная со второго символа, а при втором — присутствие символа `@`, также начиная со второго символа.

Если будут пройдены эти две проверки, вызывается функция `test`, чтобы проверить поле на наличие недопустимых символов. Если любая из этих проверок не будет пройдена, возвращается сообщение об ошибке. Допустимыми в адресе электронной почты считаются буквы в нижнем и верхнем регистрах, символы подчеркивания, тире, точки и символ `@`. Все они перечислены в регулярном выражении, передаваемом методу `test`. Если не будет найдено ни одной ошибки, возвращается пустая строка, свидетельствующая об успешно пройденной проверке. В последней строке примера сценарий и документ закрываются.

На рис. 16.2 показан результат щелчка на кнопке Зарегистрироваться без заполнения полей.

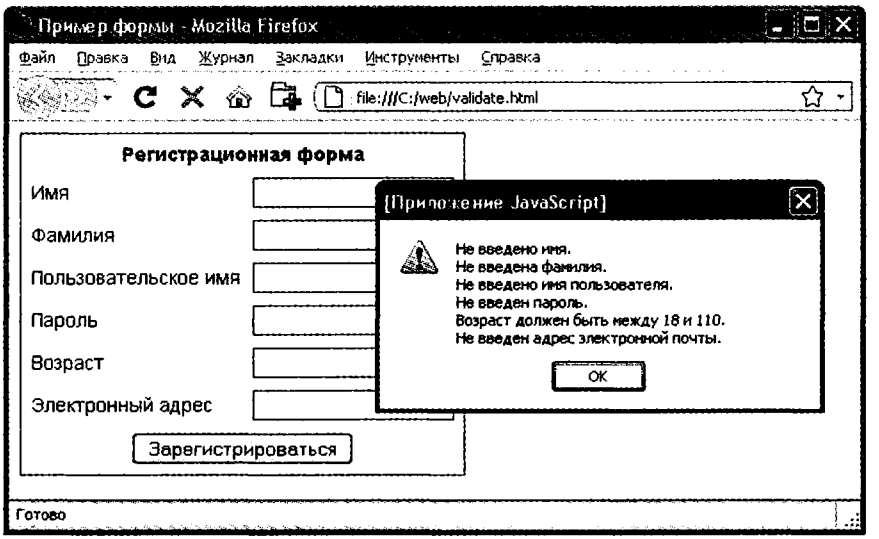



**Рис. 16.2.** Работа JavaScript-формы с проверкой данных

## Использование отдельного файла JavaScript

Конечно, благодаря универсальности своей конструкции и применимости ко многим типам потенциально востребуемых проверок эти шесть функций становятся идеальными кандидатами для выделения их в отдельный файл JavaScript (не забыв при этом удалить все теги <script> или </script>!). Это файл можно назвать `validate_functions.js` и включить сразу же после начального блока сценария в пример 16.1, используя следующую инструкцию:

```
<script src="validate_functions.js"></script>
```

# Регулярные выражения

Давайте более пристально рассмотрим шаблоны соответствия, созданные нами благодаря использованию регулярных выражений, которые поддерживаются как в JavaScript, так и в PHP. Они позволяют выстроить внутри одного выражения более мощные алгоритмы соответствия шаблонам.

## Соответствие, закладываемое в метасимволы

Любое регулярное выражение должно быть заключено в слеши (/). Конкретные символы, находящиеся внутри этих слешей, называются *метасимволами* и имеют специальное предназначение. Например, звездочка (*) имеет аналогичное (но не вполне такое же) значение, как и звездочки, уже встречавшиеся вам в оболочке или

в командной строке Windows. Звездочка означает следующее: «Текст, подвергаемый сравнению, может содержать любое количество указанного перед ней символа или не содержать его вообще».

К примеру, вы ищете имя Le Guin и знаете, что оно может быть написано как с пробелом, так и без него. Из-за не вполне обычной разметки текста (кто-нибудь, например, мог вставить лишние пробелы, чтобы выровнять строки по правому краю) нужно вести поиск в следующей строке:

```
The  difficulty  of  classifying Le     Guin's   works
```

То есть шаблон должен соответствовать строке LeGuin, а также отдельно строкам Le и Guin, разделенным любым количеством пробелов. Решением может стать установка после пробела звездочки:

```
/Le *Guin/
```

В строке, кроме имени Le Guin, присутствует множество других символов, но этот шаблон все равно будет работать. Поскольку регулярное выражение соответствует какой-то части строки, проверочная функция вернет истинное значение. А если нужно узнать, что в строке не содержится ничего другого, кроме Le Guin? Как в этом убедиться, будет показано чуть позже.

Предположим, что известно о непременном наличии хотя бы одного пробела. В таком случае можно воспользоваться знаком плюс (+), поскольку этот метасимвол требует присутствия хотя бы одного из предшествующих ему символов:

```
/Le +Guin/
```

## Нестрогое символьное соответствие

Одним из самых полезных метасимволов является точка ( . ), поскольку она может соответствовать любому символу, за исключением символа новой строки. Предположим, что выполняется поиск HTML-тегов, которые начинаются с символа < и заканчиваются символом >. Проще всего найти тег с помощью следующего регулярного выражения:

```
/<.*>/
```

Точка соответствует любому символу, а звездочка (*) расширяет действие точки до соответствия нулевому или любому другому количеству символов, что означает: «Соответствует всему, что заключено между символами < и >, даже если там ничего нет».

Этот шаблон будет соответствовать строкам <>, , <br /> и т. д. Но если не требуется, чтобы он соответствовал отсутствию символов <>, нужно вместо символа * использовать символ +:

```
/<.+>/
```

Знак «плюс» расширяет действие точки до соответствия одному или нескольким символам, что означает: «Соответствует всему, что находится между символами < и >, пока между ними есть хотя бы один символ».

Этот шаблон будет соответствовать  и , <h1> и </h1>, и тегам с атрибутами, например:

```
<a href="www.mozilla.org">
```

К сожалению, знак «плюс» расширит соответствие вплоть до последнего символа > в строке, поэтому соответствовать шаблону будет и такая строка:

```
<h1><b>Введение</b></h1>
```

А в ней содержится больше одного тега! Чуть позже в этом разделе я покажу более подходящее решение.

Если между угловыми скобками использовать только точку и не ставить за ней знаков + или *, то она будет соответствовать любому одиночному символу, а шаблон будет соответствовать таким тегам, как <b> и <i>, но не будет соответствовать тегам  или <textarea>.

Если нужно, чтобы соответствие относилось к символу точки ( . ) как таковому, его действие нужно отключить, поставив перед ним символ обратного слеша (\), поскольку в противном случае точка будет считаться метасимволом, соответствующим любому символу.

К примеру, если нужен шаблон, соответствующий числу с плавающей точкой 5.0, то в нем можно будет использовать следующее регулярное выражение:

```
/5\.0/
```

Символ обратного слеша может отключить действие любого метасимвола, в том числе и еще одного обратного слеша (если в тексте отыскивается соответствие именно обратному слешу). Но слэш может и запутать ситуацию — чуть позже будет показано, как обратные слеши иногда придают следующим за ними символам специальное предназначение.

Только что мы рассмотрели шаблон соответствия числу с плавающей точкой. Но вам наверняка понадобится проверить соответствие не только строке 5.0, но и строке 5., поскольку обе они содержат значение одного и того же числа с плавающей точкой. Нужно будет также проверить соответствие строкам 5.00, 5.000 и т. д., ведь разрешено использовать любое количество нулей. Это можно сделать добавлением звездочки:

```
/5\.0*/
```

## Группировка с помощью скобок

Предположим, что нужно найти соответствие таким возрастающим степеням, как кило, мега, гига и тера. Иными словами, нужно найти соответствие следующим строкам:

```
1.000
1.000.000
1.000.000.000
1.000.000.000.000
```

Здесь мог бы пригодиться знак «плюс», но нужно сгруппировать строку `.000` так, чтобы действие этого знака распространялось на нее целиком. Для этого служит следующее регулярное выражение:

```
/1(.000)+ /
```

Скобки означают: «При применении какого-нибудь метасимвола наподобие знака "плюс" все это нужно рассматривать как группу». Строки `1.00.000` и `1.000.00` не будут соответствовать шаблону, поскольку в тексте должен быть символ 1, за которым следует одна или несколько групп, состоящих из запятой и трех нулей.

Пробел после знака «плюс» показывает, что соответствие должно закончиться, как только встретится пробел. Без этого пробела строка `1.000.00` будет вычислена соответствующей шаблону, поскольку в расчет будет приниматься только ее первая часть `1.000`, а оставшаяся часть `.00` будет проигнорирована. Пробел нужен после остальных символов шаблона, чтобы обеспечить продолжение поиска соответствия шаблону до конца числа.

## Символьный класс

Иногда требуется установить нестрогое соответствие, но не настолько пространное, чтобы для этого использовать точку. Нестрогость придает регулярным выражениям огромную мощность: она позволяет регулировать строгость и нестрогость в соответствии с вашими желаниями.

Одним из ключевых элементов поддержки нестрогости соответствия является пара квадратных скобок `[]`. Эта пара, как и точка, соответствует всего одному символу, но в эти скобки помещается перечень всех возможных соответствий. При появлении любого из символов этого перечня текст будет соответствовать шаблону. Например, если нужно, чтобы шаблону соответствовали оба написания — американское gray и английское grey, можно задать следующее регулярное выражение:

```
/gr[ae]y/
```

В сравниваемой части текста после `gr` может быть либо `a`, либо `e`. Но должна быть только одна из этих букв: все, что помещается внутри квадратных скобок, соответствует только одному символу. Группа символов внутри скобок называется *символьным классом*.

### Указание диапазона

Для указания диапазона внутри квадратных скобок можно использовать дефис (`-`). Одной из самых распространенных задач является проверка соответствия отдельной цифре, в которой можно использовать диапазон:

```
/[0-9]/
```

Цифры являются настолько распространенным элементом регулярных выражений, что для их представления используется отдельный символ `\d`. Его можно использовать для проверки соответствия цифре вместо регулярного выражения в квадратных скобках:

## Инвертирование

Другим важным свойством квадратных скобок является *инвертирование* символьного класса. За счет помещения знака вставки (^) после открывающей квадратной скобки можно превратить весь символьный класс в его противоположность. После этого он будет означать: «Соответствует любому символу, *за исключением* следующих». Предположим, нужно найти экземпляры строк Yahoo, в которых отсутствует следующий за ними восклицательный знак. (Официальное название компании содержит восклицательный знак!) Для этого можно использовать следующее регулярное выражение:

```
/Yahoo[^!]/
```

Символьный класс состоит из одного символа — восклицательного знака, но он инвертируется стоящим перед ним символом ^. Вообще-то это не самое лучшее решение задачи. Например, это выражение не позволяет найти соответствие, если Yahoo находится в конце строки, поскольку тогда за этим словом не следует *что-нибудь*, а содержимому квадратных скобок должен соответствовать один символ. В более удачном решении используется упреждающее инвертирование (соответствие чему-нибудь, за чем нет ничего другого), но эта тема выходит за рамки данной книги.

# Более сложные примеры

После усвоения понятий символьных классов и инвертирования вы уже готовы к изучению более удачных решений задачи поиска соответствия тегу HTML. Рассматриваемое решение позволяет шаблону не пропустить закрывающую угловую скобку отдельного тега, но по-прежнему соответствовать таким тегам, как  и , а также тегам с атрибутами, таким как:

```
<a href="www.mozilla.org">
```

Один из вариантов такого решения выглядит следующим образом:

```
/<[^>]+>/
```

Это регулярное выражение похоже на результат падения чашки на клавиатуру, после которого она «по-прежнему вполне исправна и работоспособна». Давайте разобьем это выражение на части. На рис. 16.3 показан последовательный анализ всех его элементов.

Вот эти элементы:

- / — открывающий слеш, указывающий на то, что это регулярное выражение;
- < — открывающая угловая скобка тега HTML. Требует точного соответствия, поскольку не является метасимволом;
- [^>] — символьный класс. Сочетание знака вставки и закрывающей угловой скобки ^> означает: «Соответствует всему, кроме закрывающей угловой скобки»;
- + — допускает любое количество символов, соответствующих предыдущему регулярному выражению [^>], если есть хотя бы один соответствующий ему символ;

- `>` — закрывающая угловая скобка тега HTML. Требует точного соответствия;
- `/` — закрывающий слеш, указывающий на конец регулярного выражения.

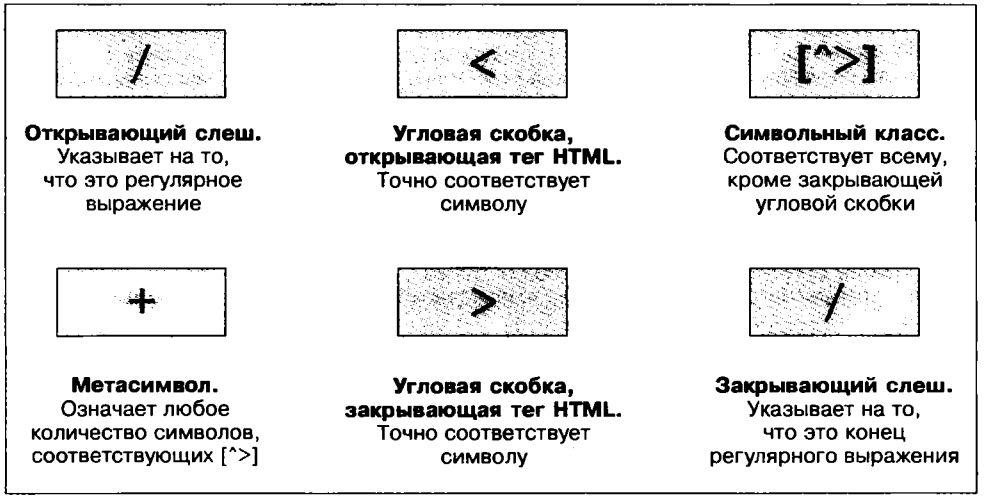



**Рис. 16.3.** Разбор типичного регулярного выражения

Другое решение задачи поиска соответствия тегам HTML связано с использованием так называемых нежадных инструкций. По умолчанию инструкция поиска соответствия шаблону является жадной, возвращающей наиболее длинное из всех возможных соответствий. При нежадном поиске соответствия ищется соответствующая строка, наиболее короткая из возможных. Применение нежадного поиска соответствия выходит за рамки данной книги, но более подробную информацию об этом можно найти по адресу http://tinyurl.com/aboutregex.

Теперь рассмотрим одно из выражений из примера 16.1, которое использовалось в функции `validateUsername`:

```
/[^a-zA-Z0-9_]/
```

На рис. 16.4 показан весь набор элементов этого выражения.

Рассмотрим эти элементы более подробно:

- `/` — открывающий слеш, указывающий на то, что это регулярное выражение;
- `[` — открывающая квадратная скобка, с которой начинается символьный класс;
- `^` — символ инвертирования: инвертирует все, что находится в скобках;
- `a-z` — представляет любую букву в нижнем регистре;
- `A-Z` — представляет любую букву в верхнем регистре;
- `0-9` — представляет любую цифру;
- `_` — символ подчеркивания;
- `]` — квадратная скобка, закрывающая символьный класс;
- `/` — обратный слеш, указывающий на конец регулярного выражения.

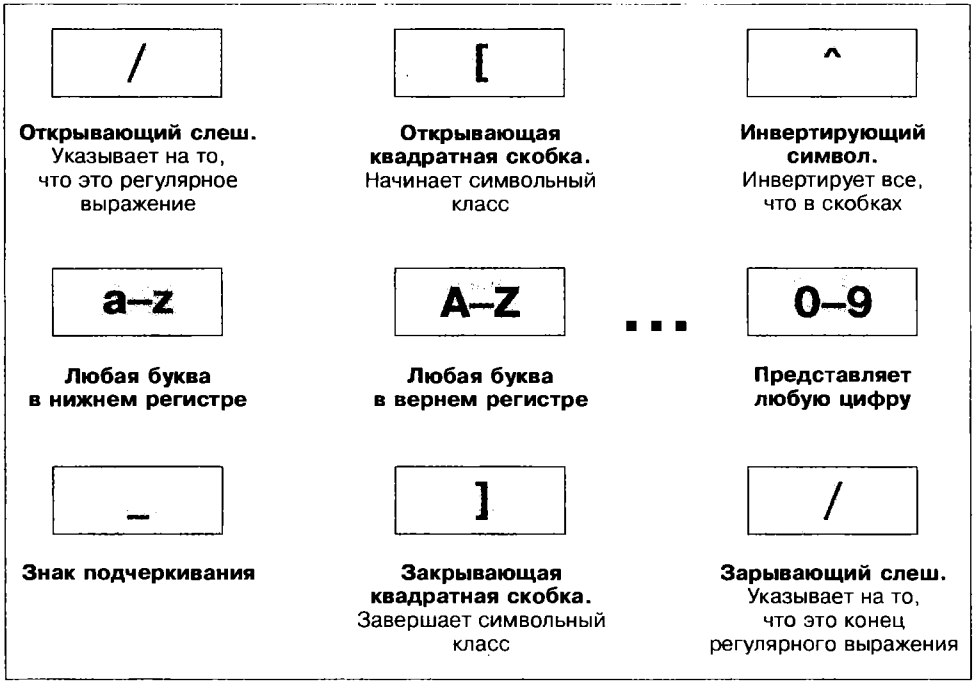



**Рис. 16.4.** Разбор регулярного выражения, используемого в функции validateUsername

Есть еще пара весьма важных метасимволов. Они «закрепляют» регулярное выражение, требуя его применения в определенном месте. Если знак вставки (^) присутствует в начале регулярного выражения, то соответствующее выражению строковое значение должно быть в начале строки текста, иначе оно не будет соответствовать шаблону. По аналогии с этим, если знак доллара ($) ставится в конце регулярного выражения, соответствующее выражению строковое значение должно находиться в конце строки текста.

Знак вставки (^) может запутать ситуацию, поскольку внутри квадратных скобок он означает «инвертировать символьный класс», а в начале регулярного выражения — «соответствовать началу строки». К сожалению, один и тот же символ служит для достижения совершенно разных целей, поэтому при его использовании следует быть особенно внимательными.

Закончим изучение основ регулярных выражений ответом на ранее заданный вопрос: предположим, вам нужно убедиться в том, что в строке нет больше ничего, кроме того, что соответствует регулярному выражению. Что делать в случае, если нужна строка текста, в которой нет ничего, кроме `Le Guin`? Можно усовершенствовать ранее рассмотренное регулярное выражение, закрепив его сразу с двух сторон:

```
/^Le *Guin$/
```

# Сводная таблица метасимволов

В табл. 16.1 показаны метасимволы, используемые в регулярных выражениях.

**Таблица 16.1.** Метасимволы регулярных выражений

| Метасимволы | Описание |
|---|---|
| / | Начало и конец регулярного выражения |
| . | Соответствует любому одному символу, кроме символа новой строки |
| *Элемент** | Соответствует появлению *элемента* от нуля и более раз |
| *Элемент* + | Соответствует появлению *элемента* от одного и более раз |
| *Элемент*? | Соответствует появлению *элемента* от нуля до одного раза |
| [*Символы*] | Соответствует одному из тех *символов*, которые содержатся в квадратных скобках |
| [^*символы*] | Соответствует одному из тех *символов*, которые не содержатся в квадратных скобках |
| (*regex*) | Рассматривает *regex* (сокращение, означающее регулярное выражение) как группу для вычисления или для рассмотрения с одним из следующих метасимволов: *, + или ? |
| *Левое*\|*правое* | Соответствует либо *левому*, либо *правому* |
| [l–r] | Соответствует диапазону символов между l и r |
| ^ | Требует, чтобы соответствие было в начале строки |
| $ | Требует, чтобы соответствие было в конце строки |
| \b | Соответствует границе слова |
| \B | Соответствует при отсутствии границы слова |
| \d | Соответствует одной цифре |
| \D | Соответствует одному символу, не являющемуся цифрой |
| \n | Соответствует символу новой строки |
| \s | Соответствует пробелу |
| \S | Соответствует символу, не являющемуся пробелом |
| \t | Соответствует символу табуляции |
| \w | Соответствует символу, используемому в словах (a–z, A–Z, 0–9 и _) |
| \W | Соответствует символу, не используемому в словах (все, кроме a–z, A–Z, 0–9 и _) |
| *x* | Соответствует *x* (применяется, если *x* является метасимволом, но нужен символ *x* как таковой) |
| {*n*} | Соответствует в точности *n* появлениям |
| {*n*,} | Соответствует *n* и более появлениям |
| {*min*,*max*} | Соответствует как минимум *min* и как максимум *max* появлениям |

После изучения этой таблицы и повторного исследования выражения /[^a-zA-Z0-9_]/ можно понять, что оно легко и просто укорачивается до /[^\w]/, потому что отдельный метасимвол \w (с буквой w в нижнем регистре) указывает на символы a–z, A–Z, 0–9 и _.

Можно проявить еще большую наблюдательность и заметить, что метасимвол \W (с буквой W в верхнем регистре) указывает на все символы, за исключением

a-z, A-Z, 0-9 и _. Это позволяет избавиться также от метасимвола ^ и использовать для выражения только символы /[\W]/.

Чтобы дать вам больше пищи для размышлений о том, что и как работает, в табл. 16.2 показан ряд выражений и описаны ситуации, которым они соответствуют.

**Таблица 16.2.** Примеры регулярных выражений

| Пример | Соответствие |
|---|---|
| r | Первая r в The quick brown |
| rec[ei][ei]ve | Либо receive, либо recieve (но также и receeve или reciive) |
| rec[ei]{2}ve | Либо receive, либо recieve (но также и receeve или reciive) |
| rec(ei\|ie)ve | Либо receive, либо recieve (но не receeve или reciive) |
| cat | Слово cat в I like cats and dogs |
| cat\|dog | Любое из слов cat или dog в I like cats and dogs |
| \ | Символ «.» (Знак «\» необходим, так как «.» является метасимволом) |
| 5\.0* | «5.», «5.0», «5.00», «5.000» и т. д. |
| [a-f] | Любой из символов a, b, c, d, e или f |
| cats$ | Только последнее слово cats в My cats are friendly cats |
| ^my | Только первое my в my cats are my pets |
| \d{2,3} | Любое двух- или трехзначное число (от 00 до 999) |
| 7(,000)+ | «7,000»; «7,000,000»; «7,000,000,000»; «7,000,000,000,000» и т. д. |
| [\w]+ | Любое слово из одного или нескольких символов |
| [\w]{5} | Любое слово из пяти символов |

# Общие модификаторы

В регулярных выражениях можно применять следующие модификаторы:

- /g — допускает «глобальное» соответствие. Этот модификатор применяется с функцией замены, что позволяет выполнить замену во всех соответствующих местах, а не только в месте первого соответствия;
- /i — отключает в регулярном выражении чувствительность к регистру букв. То есть вместо /[a-zA-Z]/ можно указать /[a-z]/i или /[A-Z]/i;
- /m — допускает многострочный режим работы, в котором знак вставки (^) и знак доллара ($) соответствуют позициям перед любыми символами новой строки в сравниваемой строковой переменной и после них. Обычно при поиске соответствия в многострочной строковой переменной знак ^ соответствует только позиции в ее начале, а символ $ — в ее конце.

Например, выражение /cats/g будет соответствовать обоим появлениям слова cats в предложении I like cats and cats like me. Аналогично этому выражение /dogs/gi будет соответствовать обоим появлениям слова dogs (Dogs и dogs) в предложении Dogs like other dogs, поскольку эти модификаторы допускают совместное использование.

## Использование регулярных выражений в JavaScript

В JavaScript регулярные выражения используются в основном в двух методах: test (который вы уже рассматривали) и replace. Метод test просто сообщает, соответствует ли его аргумент регулярному выражению, а метод replace воспринимает второй параметр — строку, которой заменяется текст, соответствующий регулярному выражению. Как и большинство методов, replace генерирует в качестве возвращаемого значения новую строку, входные данные при этом не изменяются.

Если сравнивать эти два метода, то следующая инструкция просто возвращает true, позволяя узнать, что слово cats появляется в строке хотя бы один раз:

```
document.write(/cats/i.test("Cats are fun. I like cats."))
```

А следующая инструкция заменяет оба имеющихся слова cats словом dogs, выводя результат на экран. Поиск должен быть глобальным (/g), чтобы найти все экземпляры этого слова, и нечувствительным к регистру букв (/i), чтобы найти слова, начинающиеся с большой буквы (Cats):

```
document.write("Cats are fun. I like cats.".replace(/cats/gi,"dogs"))
```

Если испытать эту инструкцию в работе, то проявятся ограничения функции замены: поскольку текст заменяется строго той строкой, которую предписано использовать, первое слово Cats заменяется словом dogs, а не словом Dogs.

## Использование регулярных выражений в PHP

В PHP наиболее часто используются следующие функции, в которых применяются регулярные выражения: preg_match, preg_match_all и preg_replace.

Чтобы проверить присутствие слова cats в любом месте строки, в любой комбинации букв в нижнем и верхнем регистрах, можно воспользоваться функцией preg_match:

```
$n = preg_match("/cats/i", "Cats are fun. I like cats.");
```

Поскольку в PHP используется значение 1 для TRUE и значение 0 для FALSE, предыдущая инструкция присвоит переменной $n значение 1. Первым аргументом функции служит регулярное выражение, а вторым — текст, проверяемый на соответствие. Но функция preg_match способна выполнять более сложную задачу, поскольку она воспринимает еще и третий аргумент, который показывает, какой именно текст соответствовал регулярному выражению:

```
$n = preg_match("/cats/i", "Cats are fun. I like cats.", $match);
echo "Количество соответсвий $n: $match[0]";
```

Третий аргумент является массивом (здесь ему присвоено имя $match). Функция помещает текст, соответствующий регулярному выражению, в первый элемент массива, поэтому, если соответствие будет найдено, то соответствующий регулярному выражению текст может быть найден в элементе $match[0]. В данном

примере выводимая на экран информация покажет, что соответствующий текст начинался с прописной буквы:

```
Количество соответствий 1: Cats
```

Если нужно определить все соответствия, используется функция preg_match_all:

```
$n = preg_match_all("/cats/i", "Cats are fun. I like cats.", $match);
echo "Количество соответствий $n: ";
for ($j=0 ; $j < $n ; ++$j) echo $match[0][$j]." ";
```

Как и в предыдущем случае, функции передан массив $match и элементу $match[0] присваиваются найденные соответствия, только теперь они представляют собой подмассив. Для отображения содержимого подмассива в этом примере осуществляется последовательный перебор его элементов с помощью цикла for.

Если нужно заменить часть строки, можно воспользоваться функцией preg_replace. В этом примере все встречающиеся слова cats, независимо от регистра букв, заменяются словами dogs:

```
echo preg_replace("/cats/i", "dogs", "Cats are fun. I like cats.");
```

Тема регулярных выражений слишком обширна, и о ней написана целая книга. Если вам нужна дополнительная информация, я рекомендую статью из «Википедии» по адресу http://tinyurl.com/wikiregex.

# Повторное отображение формы после проверки данных PHP-программой

Вернемся к проверке формы. На данный момент нами создан HTML-документ validate.html, который будет отправлен PHP-программе adduser.php, но это произойдет только в том случае, если поля пройдут проверку средствами JavaScript или если JavaScript отключен или недоступен.

Теперь настало время создать программу, сохраняемую в файле adduser.php. Эта программа получает отправленную форму и проводит собственную проверку, а затем, если проверка не будет пройдена, снова предоставляет форму визитеру. Код, который нужно будет набрать и сохранить, показан в примере 16.3.

**Пример 16.3.** Программа adduser.php

```
<?php // adduser.php

// Сначала следует код PHP

$forename = $surname = $username = $password = $age = $email = "";

if (isset($_POST['forename']))
    $forename = fix_string($_POST['forename']);
if (isset($_POST['surname']))
    $surname = fix_string($_POST['surname']);
if (isset($_POST['username']))
```

```
    $username = fix_string($_POST['username']);
if (isset($_POST['password']))
    $password = fix_string($_POST['password']);
if (isset($_POST['age']))
    $age = fix_string($_POST['age']);
if (isset($_POST['email']))
    $email = fix_string($_POST['email']);

$fail  = validate_forename($forename);
$fail .= validate_surname($surname);
$fail .= validate_username($username);
$fail .= validate_password($password);
$fail .= validate_age($age);
$fail .= validate_email($email);

echo "<html><head><title>Пример формы</title>";

if ($fail == "") {
    echo "</head><body>Проверка формы прошла успешно: $forename,
        $surname, $username, $password, $age, $email.</body></html>";

    // В этом месте отправленные поля будут вводиться в базу данных

    exit;
}

// Теперь выводится HTML и код JavaScript

echo <<<_END

<!-- Раздел HTML -->

<style>.signup { border: 1px solid #999999;
    font: normal 14px helvetica; color:#444444; }</style>
<script type="text/javascript">
function validate(form)
{
    fail  = validateForename(form.forename.value)
    fail += validateSurname(form.surname.value)
    fail += validateUsername(form.username.value)
    fail += validatePassword(form.password.value)
    fail += validateAge(form.age.value)
    fail += validateEmail(form.email.value)
    if (fail == "") return true
    else { alert(fail); return false }
}
</script></head><body>
<table class="signup" border="0" cellpadding="2"
    cellspacing="5" bgcolor="#eeeeee">
<th colspan="2" align="center">Регистрационная форма</th>

<tr><td colspan="2">К сожалению, в вашей форме <br />
```

```
найдены следующие ошибки: <p><font color=red size=1><i>$fail</i></font></p>
</td></tr>

<form method="post" action="adduser.php"
    onSubmit="return validate(this)">
     <tr><td>Имя</td><td><input type="text" maxlength="32"
    name="forename" value="$forename" /></td>
</tr><tr><td>Фамилия</td><td><input type="text" maxlength="32"
    name="surname" value="$surname" /></td>
</tr><tr><td>Пользовательское имя</td><td><input type="text" maxlength="16"
    name="username" value="$username" /></td>
</tr><tr><td>Пароль</td><td><input type="text" maxlength="12"
    name="password" value="$password" /></td>
</tr><tr><td>Возраст</td><td><input type="text" maxlength="3"
    name="age" value="$age" /></td>
</tr><tr><td>Электронный адрес</td><td><input type="text" maxlength="64"
    name="email" value="$email" /></td>
</tr><tr><td colspan="2" align="center">
    <input type="submit" value="Зарегистрироваться" /></td>
</tr></form></table>

<!-- Раздел JavaScript -->

<script type="text/javascript">
function validateForename(field) {
    if (field == "") return "Не введено имя.\\n"
    return ""
}

function validateSurname(field) {
    if (field == "") return "Не введена фамилия.\\n"
    return ""
}

function validateUsername(field) {
    if (field == "") return "Не введено имя пользователя.\\n"
    else if (field.length < 5)
        return "В имени пользователя должно быть не менее 5 символов.\\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))
        return "В имени пользователя разрешены только a-z, A-Z, 0-9, - и _.\\n"
    return ""
}

function validatePassword(field) {
    if (field == "") return "Не введен пароль.\\n"
    else if (field.length < 6)
        return "В пароле должно быть не менее 6 символов.\\n"
    else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) ||
             ! /[0-9]/.test(field))
```

```
        return "Пароль требует 1 символа из каждого набора a-z, A-Z и 0-9.\\n"
    return ""
}

function validateAge(field) {
    if (isNaN(field)) return "Не введен возраст.\\n"
    else if (field < 18 || field > 110)
        return "Возраст должен быть между 18 и 110.\\n"
    return ""
}

function validateEmail(field) {
    if (field == "") return "Не введен адрес электронной почты.\\n"
        else if (!((field.indexOf(".") > 0) &&
                   (field.indexOf("@") > 0)) ||
                  /[^a-zA-Z0-9.@_-]/.test(field))
        return "Электронный адрес имеет неверный формат.\\n"
    return ""
}
</script></body></html>
_END;

// И в завершающей части - PHP-функции

function validate_forename($field) {
    if ($field == "") return "Не введено имя<br />";
    return "";
}

function validate_surname($field) {
    if ($field == "") return "Не введена фамилия<br />";
    return "";
}

function validate_username($field) {
    if ($field == "") return "Не введено имя пользователя<br />";
    else if (strlen($field) < 5)
        return "В имени пользователя должно быть не менее 5 символов<br />";
    else if (preg_match("/[^a-zA-Z0-9_-]/", $field))
        return "В имени пользователя допускаются только буквы, цифры, - и _<br />";
    return "";
}

function validate_password($field) {
    if ($field == "") return "Не введен пароль<br />";
    else if (strlen($field) < 6)
        return "В пароле должно быть не менее 6 символов<br />";
    else if ( !preg_match("/[a-z]/", $field) ||
              !preg_match("/[A-Z]/", $field) ||
```

```
            !preg_match("/[0-9]/", $field))
        return "Пароль требует 1 символа из каждого набора a-z, A-Z и 0-9<br />";
    return "";
}

function validate_age($field) {
    if ($field == "") return "Не введен возраст<br />";
    else if ($field < 18 || $field > 110)
        return "Возраст должен быть между 18 и 110<br />"
    return "";
}

function validate_email($field) {
    if ($field == "") return "Не введен адрес электронной почты<br />";
        else if (!((strpos($field, ".") > 0) &&
                   (strpos($field, "@") > 0)) ||
                    preg_match("/[^a-zA-Z0-9.@_-]/", $field))
        return "Электронный адрес имеет неверный формат<br />";
    return "";
}

function fix_string($string) {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return htmlentities ($string);
}
?>
```

Результат отправки формы при отключенном JavaScript (и двумя неправильно заполненными полями) показан на рис. 16.5.

PHP-раздел этого кода и изменения, внесенные в HTML-раздел, выделены полужирным шрифтом, чтобы сделать заметнее все отличия кода этого примера от кода, который был показан в примерах 16.1 и 16.2.

Если вы внимательно изучили этот пример (или, будем надеяться, набрали или загрузили его с веб-сайта http://lpmj.net), то увидели, что код PHP является практически клоном кода JavaScript. Для проверки каждого из полей в очень похожих функциях используются те же самые регулярные выражения.

Но здесь следует отметить две особенности. Во-первых, для обезвреживания содержимого каждого поля и предотвращения любых попыток внедрения кода используется функция `fix_string`, которая находится в самом конце примера.

Во-вторых, вы должны были заметить, что код HTML из примера 16.1, повторенный в PHP-коде внутри структуры `<<<_END..._END;` отображает форму со значениями, которые посетитель ввел при предыдущей попытке заполнения формы. Это сделано за счет простого добавления еще одного параметра `value` к каждому тегу `<input>` (например, `value="$forename"`). Проявление такой заботы о пользователе всячески приветствуется, поскольку при этом ему не приходится снова и снова заполнять все поля, а остается лишь отредактировать ранее введенные данные.

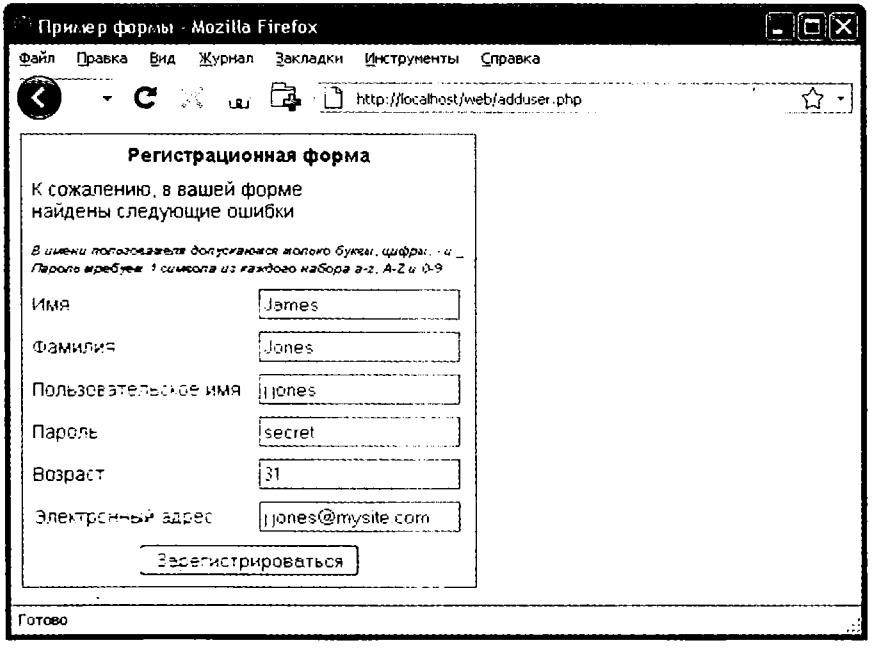



**Рис. 16.5.** Форма, отображаемая после того, как она не прошла проверку средствами PHP

При разработке реального проекта вы вряд ли стали бы сначала создавать HTML-форму вроде той, что показана в примере 16.1. Скорее всего, вместо этого вы сразу перешли бы к созданию PHP-программы, показанной в примере 16.3, в которую включен весь код HTML. И разумеется, вам потребовались бы небольшие доработки для первого вызова этой программы, для того чтобы заблокировать отображение ошибок при еще незаполненных полях. К тому же следовало бы выделить шесть функций JavaScript в отдельный включаемый файл с расширением JS.

После рассмотрения способа объединения кода PHP, HTML и JavaScript в следующей главе будет представлена технология Ajax (Asynchronous JavaScript And XML – асинхронный JavaScript и XML), в которой используются фоновые JavaScript-вызовы, обращенные к серверу с целью получения плавного обновления фрагментов веб-страницы, при котором не требуется повторная отправка всего ее содержимого с веб-сервера.

# Проверьте ваши знания

1. Каким методом JavaScript можно воспользоваться, чтобы послать данные формы на проверку перед их отправкой на сервер?
2. Какой метод JavaScript применяется для проверки соответствия строки регулярному выражению?

3. Используя определения синтаксиса регулярных выражений, напишите такое регулярное выражение, которое будет соответствовать любым символам, не использующимся в словах.
4. Напишите регулярное выражение, которое будет соответствовать как слову fox, так и слову fix.
5. Напишите регулярное выражение, которое будет соответствовать любому отдельному слову, за которым следует любой символ, не использующийся в словах.
6. Используя регулярное выражение, напишите функцию JavaScript, проверяющую наличие слова fox в строке The quick brown fox.
7. Используя регулярное выражение, напишите функцию PHP, заменяющую все экземпляры слова the в строке The cow jumps over the moon словом my.
8. Какое ключевое слово HTML используется для предварительного заполнения полей формы значениями?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 16».

# 17 Использование технологии Ajax

Термин Ajax был придуман в 2005 году. Изначально он означал «асинхронный JavaScript и XML» (Asynchronous JavaScript and XML), хотя это определение несколько устарело. Проще говоря, Ajax является технологией веб-разработки, использующей набор методов, которые встроены в JavaScript, для обмена данными между браузером и сервером в фоновом режиме. Превосходным примером применения этой технологии является Google Maps (рис. 17.1), где новый участок карты загружается с сервера по мере необходимости, для чего не требуется обновление всей страницы.

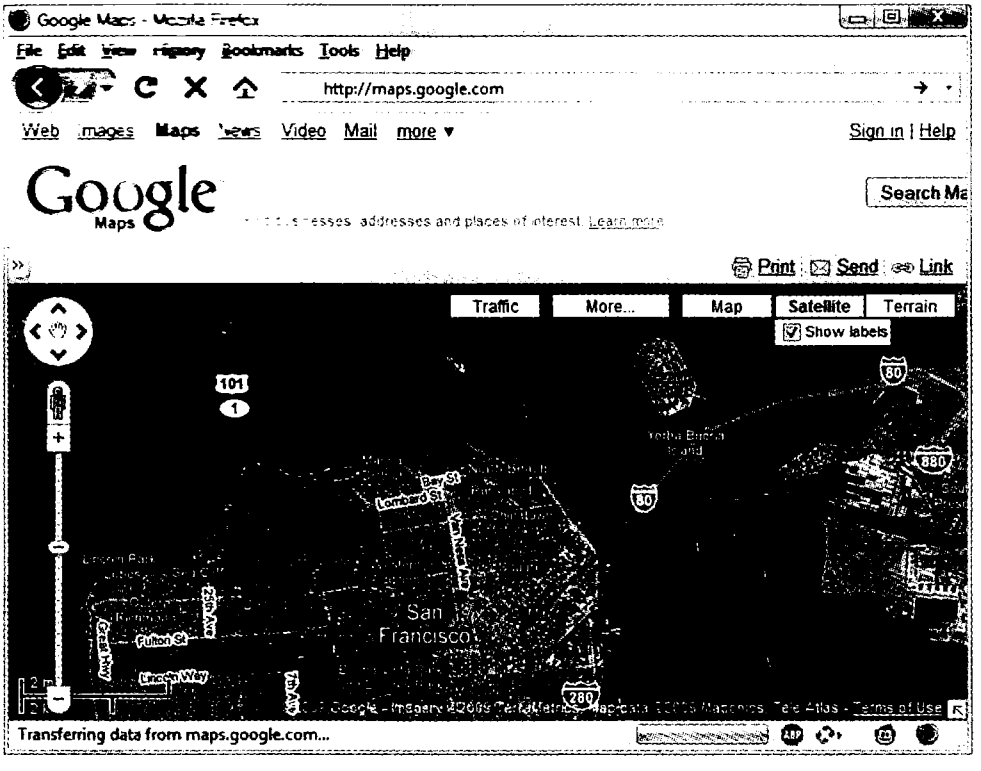



**Рис. 17.1.** Google Maps — превосходный пример использования технологии Ajax

Использование Ajax не только приводит к существенному снижению объема обмена данными, но и обеспечивает плавную динамичность веб-страниц, делая их поведение характерным для самостоятельных приложений. В результате значительно улучшается пользовательский интерфейс и ускоряется реакция на действия пользователя.

# Что такое Ajax?

Современный Ajax начинался в 1999 году с выпуска Internet Explorer 5, где был представлен новый ActiveX-объект `XMLHttpRequest`. Разработанная в корпорации Microsoft технология ActiveX предусматривает использование дополнительных программных модулей, устанавливаемых на ваш компьютер. Позже разработчики других браузеров поддержали этот почин, но вместо применения ActiveX они разработали функциональный модуль, ставший неотъемлемой частью интерпретатора JavaScript.

Но к тому времени уже появилась ранняя форма Ajax, использующая на странице скрытые фреймы, которые взаимодействуют с сервером в фоновом режиме. Самыми первыми потребителями этой технологии были участники форумов, которые задействовали ее для опроса и отображения новых сообщений без перезагрузок страницы.

Давайте посмотрим, как реализовать Ajax, используя JavaScript.

# Использование XMLHttpRequest

Из-за различий в реализации XMLHttpRequest, имеющихся в разных браузерах, возникает необходимость в создании специальной функции, обеспечивающей работу вашего кода во всех основных браузерах. Для этого необходимо разобраться с тремя способами создания объекта `XMLHttpRequest`:

- IE 5: `request = new ActiveXObject("Microsoft.XMLHTTP");`
- IE 6+: `request = new ActiveXObject("Msxml2.XMLHTTP");`
- все остальные браузеры: `request = new XMLHttpRequest();`

Дело в том, что Microsoft с выпуском Internet Explorer 6 решила внести изменения, тогда как все остальные браузеры используют несколько иной метод. Поэтому код, показанный в примере 17.1, будет работать во всех основных браузерах, выпущенных за последние несколько лет.

**Пример 17.1.** Кросс-браузерная Ajax-функция

```
<script>
function ajaxRequest()
{
    try // Браузер не относится к семейству IE?
    {
        var request = new XMLHttpRequest()
    }
    catch(e1)
    {
        try // Это IE 6+?
```

```
        {
            request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
            try // Это IE 5?
            {
                request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch e3     Данный браузер не поддерживает Ajax
            {
                request = false
            }
        }
        return request
}
</script>
```

Вспомним элементарные способы обработки ошибок из предыдущей главы, где применялась конструкция try...catch. Код примера 17.1 является прекрасной иллюстрацией той пользы, которую можно извлечь из применения этой конструкции, поскольку в этом коде ключевое слово try задействуется для выполнения Ajax-команды не в формате IE и в случае успеха — перехода к завершающей инструкции return, возвращающей новый объект. Если все же команда даст сбой, с помощью инструкции catch осуществляется перехват ошибки и выполняется следующая команда. И опять в случае успеха возвращается новый объект, а в случае неудачи предпринимается попытка выполнения последней из трех команд. Если эта попытка окажется неудачной, значит, браузер не поддерживает Ajax и объект request получает значение false, а в случае удачи возвращается полноценный объект. Итак, теперь у вас есть кросс-браузерная функция запроса Ajax, которую можно будет добавить к своей библиотеке полезных функций JavaScript.

Теперь, когда вы располагаете средством для создания объекта XMLHttpRequest, возникает вопрос о том, что можно делать с подобными объектами. Каждый такой объект поступает с набором свойств (переменных) и методов (функций), перечисленных в табл. 17.1 и 17.2 соответственно.

**Таблица 17.1.** Свойства объектов XMLHttpRequest

| Свойство | Описание |
|---|---|
| onreadystatechange | Определяет функцию обработки события, вызываемую в случае изменения имеющегося в объекте свойства readyState |
| readyState | Целочисленное свойство, дающее представление о состоянии запроса. Оно может иметь любое из следующих значений: 0 = неинициализирован, 1 = загружается, 2 = загружен, 3 = в состоянии диалога и 4 = завершен |
| responseText | Данные, возвращенные сервером в текстовом формате |
| responseXML | Данные, возвращенные сервером в формате XML |
| status | Код статуса HTTP, возвращенный сервером |
| statusText | Текст статуса HTTP, возвращенный сервером |

**Таблица 17.2.** Методы объектов XMLHttpRequest

| Метод | Описание |
|---|---|
| abort() | Отмена текущего запроса |
| getAllResponseHeaders() | Возвращение всех заголовков в виде строк |
| getResponseHeader(*параметр*) | Возвращение значения *параметра* в виде строки |
| open('*метод*', '*url*', '*асинхронно*') | Определение используемого HTTP-метода (GET или POST), целевого URL-адреса и обязательности обработки запроса в асинхронном режиме (true или false) |
| send(*данные*) | Отправка *данных* серверу назначения с использованием указанного HTTP-метода |
| setRequestHeader('*параметр*', '*значение*') | Установка в заголовок пары «*параметр — значение*» |

Эти свойства и методы позволяют управлять данными, отправляемыми на сервер и получаемыми в ответ, а также выбирать методы отправки и получения этих данных. Например, можно выбрать формат запрашиваемых данных: текстовый (который может включать HTML и прочие теги) или XML. Можно также решить, какой из методов, POST или GET, следует использовать для отправки данных на сервер.

Рассмотрим сначала метод POST, создав очень простую пару документов: комбинацию из HTML и JavaScript — и PHP-программу для взаимодействия с первым документом через Ajax. Надеюсь, вам понравятся эти примеры, поскольку в них иллюстрируется цель использования Web 2.0 и Ajax. В этих примерах за счет использования всего лишь нескольких строк JavaScript у стороннего веб-сервера запрашивается веб-документ, который затем возвращается браузеру вашим сервером и размещается в определенном разделе текущего документа.

# Реализация Ajax с помощью POST-запросов

Наберите и сохраните в файле `urlpost.html` код примера 17.2, но пока не загружайте его в свой браузер.

**Пример 17.2.** `urlpost.html`

```
<html><head><title>Пример использования Ajax</title>
</head><body><center />
<h1>Загрузка веб страницы в контейнер DIV</h1>
<div id='info'>Это предложение будет заменено</div>
<script>

params = "url=oreilly.com"
request = new ajaxRequest()
request.open("POST", "urlpost.php", true)
request.setRequestHeader("Content-type",
    "application/x-www-form-urlencoded")
request.setRequestHeader("Content-length", params.length)
request.setRequestHeader("Connection", "close")

request.onreadystatechange = function()
```

```
{
    if (this.readyState == 4)
    {
        if (this.status == 200)
        {
            if (this.responseText != null)
            {
                document.getElementById('info').innerHTML =
                    this.responseText
            }
            else alert("Ошибка Ajax: Данные не получены")
        }
        else alert( "Ошибка Ajax: " + this.statusText)
    }
}

request.send(params)

function ajaxRequest()
{
    try
    {
        var request = new XMLHttpRequest()
    }
    catch(e1)
    {
        try
        {
            request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
            try
            {
                request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
                request = false
            }
        }
    }
    return request
}
</script></body></html>
```

Изучим этот документ построчно и посмотрим, что он делает, начиная с первых трех строк, в которых устанавливается, что это HTML-документ, и отображается его заголовок. В следующей строке создается `<div>`-тег с `ID info`, в котором изначально содержится текст: «Это предложение будет заменено». Позже сюда будет вставлен текст, возвращенный с помощью технологии Ajax.

Следующие шесть строк нужны для создания Ajax-запроса типа POST. В первой строке переменной params, которая отправляется на сервер, присваивается значение, состоящее из пары *параметр = значение*. Затем создается Ajax-объект запроса. После этого вызывается метод open, настраивающий объект на создание POST-запроса по адресу urlpost.php в асинхронном режиме. Последние три строки в этой группе настраивают заголовки, необходимые для того, чтобы получающий сервер знал о поступлении POST-запроса.

## Свойство readyState

Теперь мы наконец добрались до самых тонкостей Ajax-вызова, которые целиком базируются на использовании свойства readyState. Наряду с тем, что наша программа настраивает свойство onreadystatechange на то, чтобы при каждом изменении свойства readyState вызывалась выбранная нами функция, «асинхронность» Ajax позволяет браузерам реагировать на пользовательский ввод и изменять содержимое экрана. В данном случае будет использоваться не отдельная функция, имеющая собственное имя, а безымянная (или анонимная) встроенная функция. Она относится к так называемым функциям обратного вызова, поскольку вызывается при каждом изменении свойства readyState.

Синтаксис объявления функции обратного вызова, в котором применяется встроенная безымянная функция, имеет следующий вид:

```
request.onreadystatechange = function()
{
    if (this.readyState == 4)
    {
        // какие-нибудь действия
    }
}
```

Если нужно воспользоваться отдельной функцией, имеющей собственное имя, применяется несколько иной синтаксис:

```
request.onreadystatechange = ajaxCallback

function ajaxCallback()
{
    if (this.readyState == 4)
    {
        // какие-нибудь действия
    }
}
```

При изучении табл. 17.1 можно увидеть, что у свойства readyState может быть пять разных значений. Но только одно из них представляет для нас интерес: значение 4, которое свидетельствует о завершении вызова Ajax. Поэтому при каждом вызове новой функции она возвращает управление без каких-либо действий до тех пор, пока свойство readyState не получит значение 4. Когда наша функция обнаружит это значение, следующим своим действием она проверит статус вызова, чтобы убедиться в том, что он имеет значение 200, означающее, что вызов прошел успешно.

Если этот статус не равен 200, выводится окно предупреждения с сообщением об ошибке, которое содержится в свойстве statusText.

Обратите внимание на то, что на все эти свойства объекта идут ссылки this.readyState, this.status и т. д., без использования текущего имени объекта request, как в ссылках request.readyState или request.status. Это сделано для того, дать вам возможность просто скопировать и вставить код и чтобы он после этого смог работать с любым именем объекта, поскольку ключевое слово this всегда ссылается на текущий объект.

Итак, после того, как установлено, что readyState равен 4, а status равен 200, проверяется наличие значения у свойства responseText. Если значение отсутствует, в окне предупреждения выводится сообщение об ошибке. В противном случае содержимому контейнера  присваивается значение свойства responseText:

```
document.getElementById('info').innerHTML = this.responseText
```

В этой строке с помощью метода getElementByID осуществляется ссылка на элемент info, а затем его свойству innerHTML присваивается значение, возвращенное Ajax-вызовом.

После всех этих настроечных и подготовительных действий Ajax-запрос наконец-то посылается на сервер с помощью следующей команды, которой передаются параметры, заранее определенные в переменной params:

```
request.send(params)
```

Затем весь предыдущий код активизируется при каждом изменении свойства readyState. В конце документа находятся метод ajaxRequest из примера 17.1 и теги, закрывающие сценарий JavaScript и код HTML.

## Серверная половина Ajax-процесса

Теперь мы добрались до PHP-половины этого уравнения, которая показана в примере 17.3. Наберите этот код и сохраните его в файле urlpost.php.

**Пример 17.3.** urlpost.php

```
<?php // urlpost.php
if (isset($_POST['url'])) {
    echo file_get_contents("http://".SanitizeString($_POST['url']));


function SanitizeString($var) {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
}
?>
```

Как видите, этот код невелик по объему и использует неизменно актуальную функцию обезвреживания содержимого строки — SanitizeString, которая должна применяться ко всем отправляемым в адрес сервера данным.

В этой программе для загрузки веб-страницы, которая находится по URL-адресу, представленному в POST-переменной `$_POST['url']`, используется PHP-функция `file_get_contents`. Эта функция обладает достаточной универсальностью, позволяющей ей загружать все содержимое файла или веб-страницы как с локального, так и с удаленного сервера, — она даже учитывает перемещенные страницы и другие перенаправления.

После набора программы можно будет вызвать в веб-браузере файл `urlpost.html`, и через несколько секунд должна появиться первая страница сайта oreilly.com, содержимое которой загружено в `<div>`-контейнер, созданный нами для этих целей. Произойдет это не так быстро, как при непосредственной загрузке веб-страницы, поскольку данные переносятся дважды: сначала на сервер, а потом с сервера на браузер. Результат должен быть похож на тот, что показан на рис. 17.2.

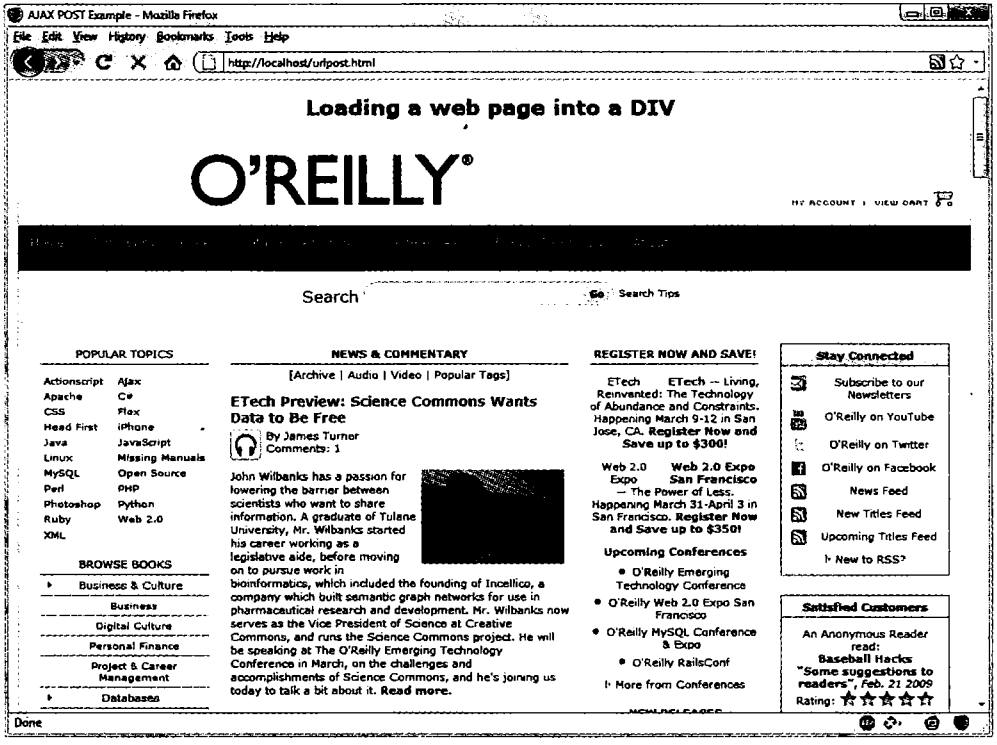



**Рис. 17.2.** Первая страница веб-сайта oreilly.com, загруженная в `<div>`-контейнер

Мы не только добились осуществления Ajax-вызова и получения ответа, возвращенного JavaScript, мы также воспользовались способностью PHP объединять совершенно не связанные друг с другом веб-объекты. Кстати, если бы мы попытались найти способ извлечения веб-страницы oreilly.com непосредственно через Ajax (без обращения к PHP-модулю на стороне сервера), у нас ничего бы не вышло, поскольку существуют блоки безопасности, не допускающие кросс-доменного применения технологии Ajax. Поэтому данный небольшой пример показывает также удобное решение весьма актуальной практической задачи.

# Использование GET вместо POST

При отправке данных любой формы можно выбрать GET-запросы, сэкономив на этом несколько строк кода. Но у таких запросов есть недостаток: некоторые браузеры могут кэшировать GET-запросы, притом что POST-запросы кэшированию никогда не подвергаются. Кэширование запроса нежелательно, потому что браузер просто-напросто заново отобразит то, что он получил в последний раз, и не станет обращаться к серверу за свежими входными данными. Решить эту проблему можно, применив обходной маневр, заключающийся в добавлении к каждому запросу произвольного параметра, обеспечивающего уникальность каждого запрашиваемого URL-адреса.

В коде примера 17.4 показано, как можно добиться такого же результата, который был получен при использовании кода примера 17.2, но на этот раз применяя в Ajax не POST-, а GET-запрос.

**Пример 17.4.** urlget.html

```
<html><head><title>Пример Ajax с GET-запросом</title>
</head><body><center />
<h1>Загрузка веб-страницы в DIV-контейнер</h1>
<div id='info'>Это предложение будет заменено</div>
<script>

nocache = "&nocache=" + Math.random() * 1000000
request = new ajaxRequest()
request.open("GET", "urlget.php?url=oreilly.com" + nocache, true)

request.onreadystatechange = function()
{
    if (this.readyState == 4)
    {
        if (this.status == 200)
        {
            if (this.responseText != null)
            {
                document.getElementById('info').innerHTML =
                    this.responseText
            }
            else alert("Ошибка Ajax: Данные не получены ")
        }
        else alert( "Ошибка Ajax: " + this.statusText)
    }


request.send(null)

function ajaxRequest()

    try
    {
        var request = new XMLHttpRequest()
```

```
    }
    catch(e1)
    {
        try
        {
            request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
            try
            {
                request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
                request = false
            }
        }
    }
    return request
}
</script></body></html>
```

Различия между этими двумя документами, на которые следует обратить внимание, выделены полужирным шрифтом и состоят в следующем.

- Для GET-запроса не требуется отправка заголовков.
- Метод open вызывается с использованием GET-запроса с предоставлением URL-адреса, строка которого содержит символ ?, за которым следует пара «параметр — значение» — url=oreilly.com.
- Вторая пара «параметр — значение» начинается с использования символа &, за которым для параметра nocache устанавливается случайное значение из диапазона от 0 до 1 000 000. Такой прием обеспечивает разное содержимое каждого запрашиваемого URL-адреса, что препятствует обслуживанию запросов из кэша.
- Вызов метода send теперь содержит только параметр null, поскольку здесь отсутствуют параметры, которые передаются при POST-запросе. Учтите, что опускать этот параметр нельзя, поскольку это вызовет ошибку.

Для сопровождения нового документа необходимо изменить PHP-программу так, чтобы она отвечала на GET-запрос. Файл urlget.php, в котором содержится код программы, показан в примере 17.5.

**Пример 17.5.** urlget.php

```
<?php
if (isset($_GET['url'])) {
    echo file_get_contents("http://".sanitizeString($_GET['url']));
}

function sanitizeString($var) {
    $var = strip_tags($var);
```

```
    $var = htmlentities($var);
    return stripslashes($var);
}
?>
```

Разница между этим кодом и кодом примера 17.3 заключаются в том, что ссылка на массив $_POST заменена ссылкой на массив $_GET. Конечный результат вызова urlget.html в вашем браузере будет идентичен результату вызова urlpost.html.

# Отправка XML-запросов

Хотя создаваемые нами объекты называются объектами XMLHttpRequest, пока мы обходились без использования XML. В рассмотренных случаях применения данной технологии использование термина Ajax было в какой-то степени неоправданным, поскольку фактически технология позволяет запрашивать текстовые данные любого вида, а не только те, которые относятся к формату XML. Вы смогли убедиться в том, что мы запрашивали с помощью Ajax весь HTML-документ, но могли бы с таким же успехом запросить текстовую страницу, числовую строку или даже данные электронной таблицы.

Внесем изменения в приведенный ранее пример документа и PHP-программы и настроим их на извлечение данных в формате XML. Для этого рассмотрим сначала PHP-программу xmlget.php, показанную в примере 17.6.

**Пример 17.6.** xmlget.php

```
<?php
if (isset($_GET['url'])) {
    header('Content-Type: text/xml');
    echo file_get_contents("http://".sanitizeString($_GET['url']));

function sanitizeString($var) {
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
```

Эта программа по сравнению с предыдущей подверглась небольшому изменению (которое выделено полужирным шрифтом), чтобы перед возвращением извлеченного документа выводился правильный XML-заголовок. Здесь не выполняются никакие проверки, поскольку предполагается, что Ajax-вызов запросит настоящий XML-документ.

Теперь рассмотрим HTML-документ xmlget.html, показанный в примере 17.7.

**Пример 17.7.** xmlget.html

```
<html><head><title>Пример извлечения XML с помощью Ajax</title>
</head><body>
<h2>Загрузка XML содержимого в DIV-контейнер</h2>
```

```
<div id='info'>Это предложение будет заменено</div>
<script>

nocache = "&nocache=" + Math.random() * 1000000
url = "rss.news.yahoo.com/rss/topstories"
request = new ajaxRequest()
request.open("GET", "xmlget.php?url=" + url + nocache, true)
out = "";

request.onreadystatechange = function()
{
    if (this.readyState == 4)
    {
        if (this.status == 200)
        {
            if (this.responseXML != null)
            {
                titles = this.responseXML.getElementsByTagName('title')

                for (j = 0 ; j < titles.length ; ++j)
                {
                    out += titles[j].childNodes[0].nodeValue + '<br />'
                }
                document.getElementById('info').innerHTML = out
            }
            else alert("Ошибка Ajax: Данные не получены")
        }
        else alert( "Ошибка Ajax: " + this.statusText)
    }
}

request.send(null)

function ajaxRequest() {
    try
    {
        var request = new XMLHttpRequest()
    }
    catch(e1)
    {
        try
        {
            request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
            try
            {
                request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
```

```
            {
                request = false
            }
        }
    }
    return request
}
</script></body></html>
```

В этом коде все различия также выделены полужирным шрифтом, чтобы вы могли увидеть, что он очень похож на предыдущие версии, за исключением того, что теперь запрашивается URL-адрес rss.news.yahoo.com/rss/topstories, по которому находится XML-документ, содержащий поток последних новостей — Yahoo! News Top Stories feed.

Другое существенное изменение касается использования свойства `responseXML`, которым заменено свойство `responseText`. Когда сервер возвращает XML-данные, свойство `responseText` возвращает значение `null`, а свойство `responseXML` будет содержать возвращенные XML-данные.

Но `responseXML` не просто содержит строку XML-текста — на самом деле в нем содержится полноценный объект XML-документа, который может быть проанализирован с использованием методов и свойств DOM-дерева. Это, к примеру, означает, что к нему можно применить JavaScript-метод `getElementsByTagName`.

## Несколько слов о XML

Документ XML, как правило, имеет форму RSS-потока, показанного в примере 17.8. Но красота XML заключается в том, что этот тип структуры может быть сохранен внутри DOM-дерева (рис. 17.3), что дает возможность выполнять в нем быстрый поиск.

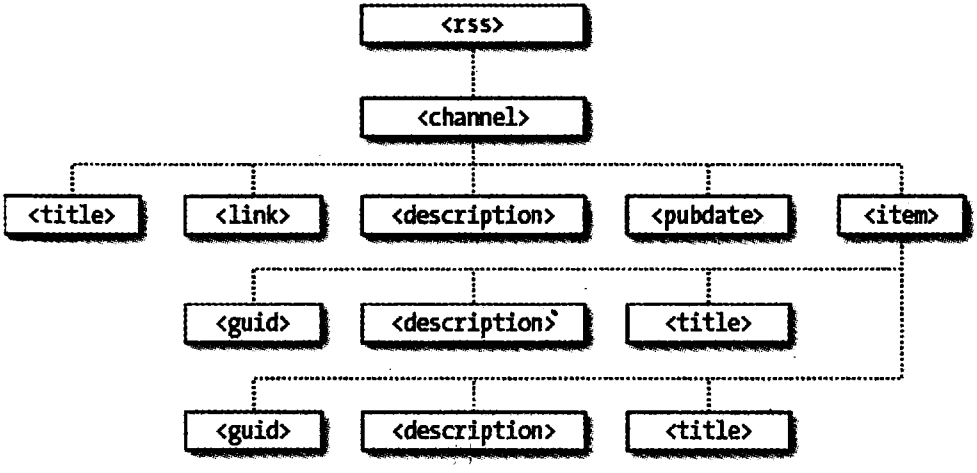



**Рис. 17.3.** DOM-дерево примера 17.8

**Пример 17.8.** Документ XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
    <channel>
        <title>RSS-поток</title>
        <link>http://website.com</link>
        <description>RSS-поток website.com </description>
        <pubDate>Понедельник, 16 мая 2011 года, 00:00:00 GMT</pubDate>
        <item>
            <title>Заголовок</title>
            <guid>http://website.com/headline</guid>
            <description>Это заголовок</description>
        </item>
        <item>
            <title>Заголовок 2</title>
            <guid>http://website.com/headline2</guid>
            <description>Второй заголовок</description>
        </item>
    </channel>
</rss>
```

Следовательно, используя метод `getElementsByTagName`, можно быстро извлечь значения, связанные с различными тегами, не занимаясь громоздким строчным поиском. Именно это и делается в коде примера 17.7, где выдается следующая команда:

```
titles = this.responseXML.getElementsByTagName('title')
```

За счет выполнения только этой одной команды все значения элементов `title` помещаются в массив `titles`. После этого остается лишь извлечь их с помощью следующего выражения (где `j` — порядковый номер заголовка, к которому осуществляется доступ):

```
titles[j].childNodes[0].nodeValue
```

Затем все заголовки добавляются к строковой переменной `out`, и поскольку все они уже прошли обработку, результат вставляется в пустой `<div>`-контейнер в начале документа. При вызове в браузере файла `xmlget.html` будет получен результат, похожий на тот, что показан на рис. 17.4.

При запросе XML-данных, равно как и при работе со всеми другими формами данных, можно воспользоваться либо методом POST, либо методом GET — ваш выбор на результат не повлияет.

## А зачем вообще использовать XML?

Может возникнуть вопрос, а для чего еще можно применять XML, кроме как для извлечения XML-документов в виде RSS-потоков? Проще всего ответить, что пользоваться им вас никто не заставляет, но если вам нужно возвращать структурированные данные своим Ajax-приложениям, то отправка простых, неорганизо-

ванных фрагментов текста может превратиться в настоящую проблему, для решения которой потребуется довольно сложная обработка в JavaScript.

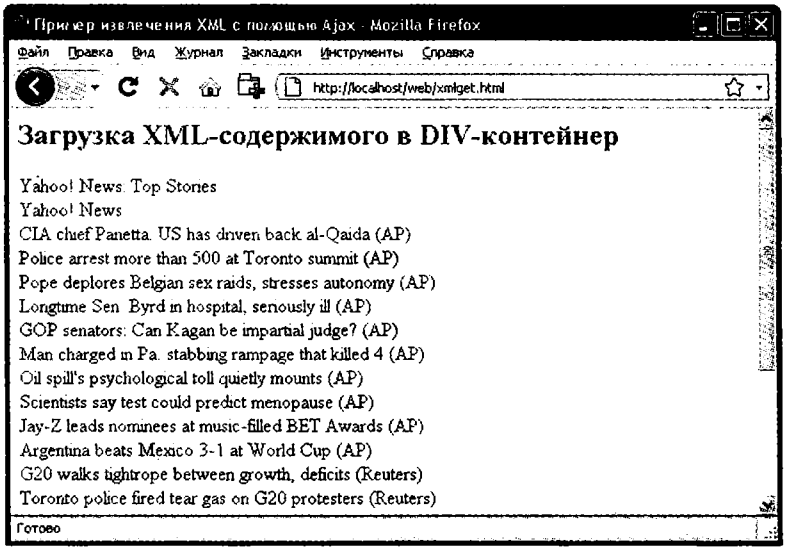



**Рис. 17.4.** Извлечение с помощью Ajax новостного XML-потока Yahoo!

Вместо этого можно создать XML-документ и вернуть его Ajax-функции, которая автоматически поместит его в DOM-дерево в виде уже знакомого вам легкодоступного HTML DOM-объекта.

# Использование для Ajax специальной среды

Теперь, когда вы узнали о том, как создавать собственные Ajax-процедуры, можно будет исследовать некоторые из свободно распространяемых программных продуктов, которые представляют собой среду, способную упростить работу с применением этой технологии и предлагающую множество более совершенных функциональных возможностей. В частности, я советую обратить внимание на библиотеку jQuery, которая, наверное, является наиболее востребованной средой для работы с применением Ajax.

Ее можно загрузить (и получить полную документацию) с веб-сайта http://jquery.com, но знайте, что вас с самого начала ждет крутой поворот в ее изучении, поскольку вам придется познакомиться с предоставляемой этой средой функцией $, которая широко используется для обращения к функциональным возможностям jQuery. Тем не менее, разобравшись с работой jQuery, вы поймете, что эта среда может существенно облегчить и ускорить веб-разработку благодаря большому количеству предлагаемых ею готовых функций.

Теперь, изучив азы работы технологии Ajax, в следующей главе мы рассмотрим, как можно в наш набор технологий веб-разработки ввести каскадные таблицы стилей — CSS (Cascading Style Sheets).

# Проверьте ваши знания

1. Зачем нужна функция для создания новых XMLHttpRequest-объектов?
2. Для чего предназначена конструкция try...catch?
3. Сколько свойств и методов имеется у объекта XMLHttpRequest?
4. Как можно определить завершение Ajax-вызова?
5. Как узнать об успешном завершении Ajax-вызова?
6. В каком свойстве объекта XMLHttpRequest содержится текстовый ответ, возвращенный Ajax-вызовом?
7. В каком свойстве объекта XMLHttpRequest содержится XML-ответ, возвращенный Ajax-вызовом?
8. Как указать функцию обратного вызова, предназначенную для обработки ответов Ajax-вызова?
9. Какой метод объекта XMLHttpRequest используется для инициирования Ajax-запроса?
10. В чем состоит основное различие между Ajax GET- и POST-запросом?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 17».

# 18 Введение в CSS

Используя каскадные таблицы стилей — Cascading Style Sheets (CSS), — вы можете применить стиль к своим веб-страницам, чтобы придать им желаемый внешний вид. Работа CSS основана на их подключении к объектной модели документа — Document Object Model (DOM), которая была рассмотрена в главе 13.

Используя CSS, можно быстро и просто изменить стиль любого элемента. Например, если не нравится исходный вид заголовков, определяемых тегами <h1>, <h2> и т. д., можно назначить новый стиль, отменяющий исходные настройки, касающиеся используемого семейства шрифтов и размера, применения полужирного шрифта или курсива, а также многих других свойств.

Один из способов добавления стилей к веб-странице заключается во вставке требуемых для этого инструкций в заголовок страницы между тегами <head> и </head>. Поэтому для изменения стиля, применяемого к содержимому тега <h1>, нужно воспользоваться следующим кодом (синтаксис которого будет рассмотрен чуть позже):

```
<style>
    h1 { color:red; font-size:3em; font-family:Arial; }
</style>
```

Внутри HTML-страницы этот код может иметь вид, показанный в примере 18.1, в котором, подобно всем остальным примерам, используемым в данной главе, применяется стандартное HTML5-объявление DOCTYPE. Результат выполнения этого кода показан на рис. 18.1.

**Пример 18.1.** Простая HTML-страница

```
<!DOCTYPE html>
<html>
    <head>
        <title>Здравствуй, мир! </title>
        <style>
            h1 { color:red; font-size:3em; font-family:Arial; }
        </style>
    </head>
    <body>
        <h1>Всем привет</h1>
    </body>
</html>
```

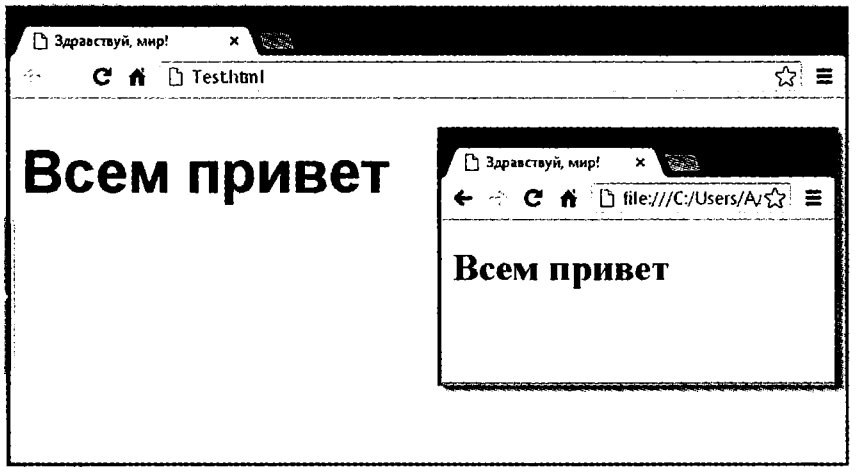



**Рис. 18.1.** Стилизация тега, оригинальный стиль которого показан во вставке

# Импортирование таблицы стилей

Когда стиль нужно применить не к одной странице, а ко всему сайту, лучше управлять стилями путем перемещения их из веб-страниц в отдельные файлы, называемые *таблицами стилей*, и затем импортировать стиль, который вам нужен. Тем самым достигается минимизация того объема кода, который дублируется на ваших страницах, улучшаются показатели обслуживаемости и предоставляется возможность применения разных таблиц стилей к разным форматам подачи информации (например, в варианте просматриваемой веб-страницы и в варианте вывода на печать) без изменения HTML. Такое отделение содержимого от разметки является фундаментальным принципом дизайна.

Достичь этого можно двумя различными способами, первый из которых заключается в использовании CSS-директивы `@import`:

```
<style>
    @import url('styles.css');
</style>
```

Эта инструкция заставляет браузер извлечь таблицу стилей с именем `styles.css`. Гибкость, присущая команде `@import`, позволяет создавать таблицы стилей, которые сами импортируют другие таблицы стилей, а те, в свою очередь, могут импортировать другие таблицы и т. д. А вот теги `<style>` и `</style>` при вызове внешних таблиц стилей из других таблиц не нужны, и их присутствие сделает инструкцию неработоспособной.

**Импортирование таблицы стилей из HTML-кода.** Включить таблицу стилей можно также с помощью HTML-тега `<link>`:

```
<link rel='stylesheet' type='text/css' href='styles.css' />
```

Результат будет точно таким же, как и при использовании директивы `@import`, но `<link>` является тегом, применяемым только в HTML, и не относится к стилевым

директивам, поэтому он не может задействоваться в одной таблице стилей для импорта другой такой таблицы. Он также не может помещаться внутри пары тегов <style>...</style>.

Точно так же, как в CSS, можно использовать несколько директив @import для включения в состав таблицы стилей нескольких внешних таблиц, в коде HTML можно применять любое нужное количество элементов, задействующих теги <link>.

# Встроенные настройки стиля

Можно также выполнять индивидуальные настройки или заменять конкретные стили, вставляя объявления стилей непосредственно в код HTML следующим образом (в данном случае внутри тегов задается курсивный текст синего цвета):

```
<div style='font-style:italic; color:blue;'>Всем привет</div>
```

Но подобные настройки стоит отложить до тех пор, пока не сложатся самые крайние обстоятельства, поскольку они нарушают принцип отделения содержимого от разметки.

## Использование идентификаторов (ID)

Более удачным решением для настроек стиля отдельно взятого элемента является назначение формирующему его коду HTML идентификатора

```
<div id='iblue'>Всем привет</div>
```

Тем самым устанавливается, что содержимое -контейнера с идентификатором, имеющим значение iblue, должно иметь применяемый к нему стиль, который определен в стилевых настройках iblue. Соответствующая CSS-инструкция для этого может иметь следующий вид:

```
#iblue { font-style:italic; color:blue; }
```

Обратите внимание на использование символа решетки (#), который указывает на то, что эта инструкция задает стилевые настройки только для идентификатора по имени iblue.

## Использование классов

Если нужно применить один и тот же стиль ко многим элементам, не следует давать каждому из них особый ID, поскольку можно указать класс для управления всеми этими элементами:

```
<div class='iblue'>Привет</div>
```

Тем самым утверждается, что стиль, определенный в классе iblue, должен применяться к содержимому данного элемента (и любых других элементов, относящихся к этому классу). При использовании класса можно либо в заголовке страницы, либо во внешней таблице стилей задействовать для настройки стилей класса следующее правило:

```
.iblue { font-style:italic; color:blue; }
```

Вместо использования символа решетки (#), который закреплен за идентификаторами (ID), инструкции, относящиеся к классу предваряются символом точки (.).

# Правила CSS

*Правило* CSS является инструкцией или последовательностью инструкций, предписывающих браузеру способ отображения конкретного элемента или элементов на странице. Каждая инструкция в CSS-правиле начинается с *селектора*, являющегося элементом, к которому будет применяться правило. Например, в следующем назначении h1 является селектором, для которого задается размер шрифта на 240 % больше чем у того, который используется по умолчанию:

```
h1 { font-size:240%; }
```

Все имеющиеся в правилах изменяемые свойства должны быть расположены внутри символов { и } и следовать за селектором. Часть, которая находится перед двоеточием (в данном случае это font-size) является изменяемым свойством, а часть, которая располагается после двоеточия, является применяемым к этому свойству значением (в данном случае это 240%). Задавая для принадлежащего селектору h1 свойства font-size значение 240%, мы гарантируем, что содержимое тегов <h1>...</h1> будет отображено с размером шрифта, превосходящим на 240 % исходный размер.

И наконец, следует точка с запятой (;), завершающая инструкцию. В данном примере, поскольку font-size является последним свойством правила, точка с запятой не требуется (но она должна присутствовать, если за этим свойством будет задано значение еще одного свойства).

## Использование точек с запятой

В CSS точки с запятой применяются в качестве разделителей нескольких инструкций CSS, расположенных в одной и той же строке. Но при наличии в правиле только одной инструкции (или при встраивании настройки стиля в HTML-теге) точку с запятой можно опустить, и то же самое можно сделать в отношении последней инструкции в группе.

Но чтобы при использовании CSS избавиться от ошибок, которые трудно будет распознать, можно взять за правило использование точки с запятой после каждой настройки CSS. Затем их можно копировать и вставлять или же изменять свойства, не заботясь об удалении точек с запятой там, где они, в принципе, не нужны, или о добавлении их туда, где они необходимы.

## Множественные задания стиля

Задать несколько стилевых настроек можно двумя разными способами. Можно объединить их в одной строке:

```
h1 { font-size:240%; color:blue; }
```

Здесь добавлено второе задание стиля, изменяющее цвет всех заголовков, задаваемых тегом <h1>, на синий. Можно также расположить задания построчно:

```
h1 { font-size:240%;
color:blue; }
```

Или же можно разнести задания еще дальше, расположив столбцами по двоеточиям:

```
h1
{
 font-size :240%;
 color     :blue;
}
```

Тогда будет проще заметить, где начинается каждый новый набор правил, поскольку селектор всегда находится в первом столбце, а следующие за ним задания аккуратно выстраиваются благодаря одинаковому горизонтальному смещению всех значений свойств.

В предыдущих примерах замыкающие точки с запятой не нужны, но если придется объединять какие-нибудь подобные группы инструкций в одну строку, то при наличии всех точек с запятой это можно будет сделать довольно быстро.

Один и тот же селектор можно указывать произвольное количество раз, и CSS будет объединять все свойства. То есть предыдущему примеру можно также придать следующий вид:

```
h1 { font-size: 240%; }
h1 { color    : blue; }
```

Каких-либо правильных или неправильных способов раскладки кода CSS не существует, но я рекомендую вам по крайней мере стараться соблюдать единообразие в построении каждого блока CSS, чтобы в нем можно было разобраться с первого взгляда.

А что произойдет, если задать одно и то же свойство для одного и того же селектора дважды?

```
h1 { color      red; }
h1 { color      blue; }
```

Будет применено последнее заданное свойство, в данном случае то, которое имеет значение `blue`. Повторять в одном файле одно и то же свойство для одного и того же селектора было бы бессмысленно, но такие повторения часто бывают при реальном использовании веб-страниц, когда для них применяются сразу несколько стилей. Это и есть одно из ценных свойств CSS, которое называется *каскадированием*.

## Использование комментариев

CSS-правила желательно прокомментировать, пусть даже не все или не основную их часть, а только главную группу инструкций. Это можно сделать двумя способами. Можно, например, разместить комментарии внутри пары следующих тегов:

```
/* Это комментарий CSS */
```

Или же можно развернуть комментарий на несколько строк:

```
/*
    Много-
    строчный
    комментарий
*/
```

При использовании многострочного комментария нужно иметь в виду, что в них нельзя вкладывать однострочные (или любые другие) комментарии. Это может привести к непредсказуемым ошибкам.

# Типы стилей

Существует несколько разных типов стилей, начиная с исходных стилей, установленных в вашем браузере (и любых пользовательских стилей, которые вы можете применить в своем браузере, чтобы переопределить исходные значения), продолжая вложенными или встроенными стилями и заканчивая внешними таблицами стилей. Для стилей, которые были определены, действует иерархическая последовательность выполнения, направленная снизу вверх.

## Исходные стили

В веб-браузере применяется задание исходных стилей, имеющих самый низкий уровень приоритета. Этот набор стилей создается на тот случай, когда у веб-страницы нет определений каких-нибудь других стилей. Он предназначен для использования в качестве общего набора стилей, достаточно корректно отображаемого в большинстве случаев.

До создания CSS это были только стили, применяемые к документу, и только небольшая часть этих стилей могла быть изменена веб-страницей (например, внешний вид шрифта, его цвет и размер плюс несколько аргументов, относящихся к размеру элементов).

## Пользовательские стили

Далее по уровню возрастания приоритета следуют стили, определенные пользователем. Они поддерживаются большинством современных браузеров, но в каждом из них реализованы по-своему. Если вы хотите узнать, как создавать собственные исходные настройки стилей для просмотра веб-страниц, воспользуйтесь какой-нибудь поисковой системой и введите в нее название вашего браузера и далее слова «пользовательские стили» (user styles) Например, «Firefox пользовательские стили» или «Internet Explorer пользовательские стили». На рис. 18.2 показано окно выбора таблицы пользовательских стилей Microsoft Internet Explorer.

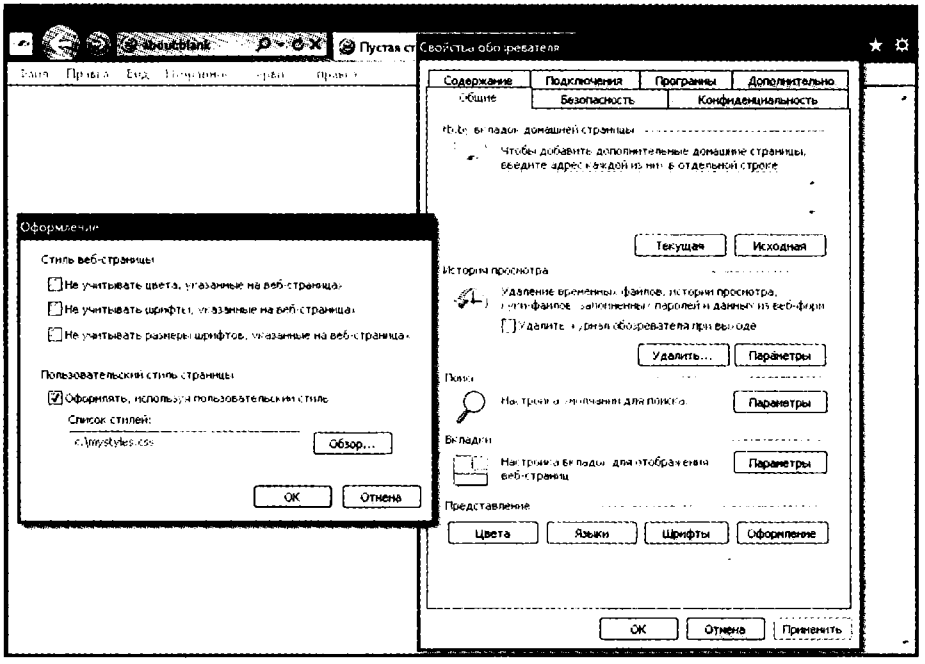



**Рис. 18.2.** Применение пользовательской таблицы стилей для Internet Explorer

Если задан пользовательский стиль, который уже был определен в качестве исходного стиля браузера, то пользовательский стиль заменит исходный стиль браузера. Любые стили, не определенные в пользовательской таблице стилей сохранят свои исходные значения, установленные в браузере.

## Внешние таблицы стилей

Стили, заданные во внешней таблице стилей, заменят любые стили, заданные как пользователем, так и браузером. Внешние таблицы стилей являются рекомендуемым способом создания ваших стилей, поскольку вы можете создавать разные таблицы стилей для разных целей, например для общего использования в Интернете, для просмотра страниц на браузерах мобильных устройств с небольшими экранами, для получения распечатки и т. д. Нужно будет просто применить при создании веб-страницы один нужный набор стилей для каждого типа носителя информации.

## Внутренние стили

Внутренние стили, создаваемые внутри тегов <style> ... </style>, имеют более высокий уровень приоритета над всеми предыдущими типами стилей и поэтому могут использоваться для замены стилевого оформления в любых внешних таблицах стилей, загруженных в то же самое время. Но с этого места начинается отделение стилевого оформления от содержимого.

## Внедренные стили

И наконец, рассмотрим внедренные стили, представляющие собой назначение свойства непосредственно элементу. Они также имеют наивысший уровень приоритета над любым типом стилей, и их использование имеет следующий вид:

```
<a href="http://google.com" style="color:green;">Посетите Google</a>
```

В этом примере определяемая ссылка будет отображена зеленым цветом, независимо от любых исходных или других цветовых настроек, применяемых браузером или любой другой таблицей стилей либо непосредственно к этой ссылке, либо общим порядком ко всем ссылкам.

При использовании этого типа образования стилей вы нарушаете отделение разметки от содержимого, и поэтому применять подобные решения рекомендуется только в случае крайней необходимости.

# Селекторы CSS

Средства доступа к одному или нескольким элементам страницы называются *селекцией*, и, как уже ранее показывалось, та часть правила CSS, которая этим занимается, известна как селектор. Как вы уже, наверное, догадались, существует множество разнообразных селекторов.

## Селектор типа

Селектор типа определяет HTML-элемент, к которому применяется стиль, например  или <i>. Следующее правило, к примеру, обеспечивает полное выравнивание всего текста, находящегося между тегами ... :

```
p { text-align:justify; }
```

## Селектор потомков

Селекторы потомков позволяют применять стили к элементам, содержащимся внутри других элементов. Например, следующее правило настраивает вывод всего текста внутри тегов ... красным цветом, но только если эти теги окажутся внутри тегов ... (как в этом случае: Hello there ):

```
p b { color:red; }
```

Вложенность селекторов потомков может продолжаться до бесконечности, поэтому следующее правило является вполне приемлемым для того, чтобы полужирный текст внутри элемента маркированного списка выводился синим цветом:

```
ul li b { color:blue; }
```

В качестве практического примера представим себе, что нужно использовать другую систему нумерации, отличающуюся от исходной для пронумерованного списка вложенного в маркированный список. Этого можно достичь следующим способом, заменяющим исходную нумерацию (начинающуюся с 1) буквами в нижнем регистре (начинающимися с a):

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            ul ol { list-style-type:lower-alpha; }
        </style>
    </head>
    <body>
        <ol>
            <li>Один</li>
            <li>Два</li>
            <li>Три</li>
        </ol>
        <ul>
            <ol>
                <li>Один</li>
                <li>Два</li>
                <li>Три</li>
            </ol>
        </ul>
    </body>
</html>
```

Загрузка этого кода HTML в веб-браузер даст следующий результат — как видите, <li>-элементы внутри второго <ol>-блока отображаются по-иному, нежели в первом пронумерованном списке:

1. Один
2. Два
3. Три
   a. Один
   b. Два
   c. Три

## Селектор дочерних элементов

Селектор дочерних элементов похож на селектор потомков, но он еще больше конкретизирует область применения стиля, выбирая только те элементы, которые являются непосредственными дочерними элементами другого элемента. Например, следующий код использует селектор потомков, который изменит цвет любого текста, выделенного полужирным шрифтом внутри абзаца на красный, даже если сам текст, выделенный полужирным шрифтом находится внутри выделения курсивом (подобно следующему коду: Привет всем ):

```
p b { color:red; }
```

В данном случае слово Привет отображается красным цветом. Но когда требуется более конкретное поведение, чтобы сделать область применения селектора еще у́же, может использоваться селектор дочерних элементов. Например, следующий селектор дочерних элементов установит красный цвет для текста, выделенного полужирным шрифтом только в том случае, если элемент будет непосредственным дочерним элементом абзаца и внутри не содержится другого элемента:

```
p > b { color:red: }
```

Теперь слово Привет в предыдущем примере HTML-кода не изменит свой цвет, потому что элемент <b> не является непосредственным дочерним элементом того элемента, который задается тегом <p>.

В качестве практического примера представим себе, что нужно применить стиль только к тем <li>-элементам, которые являются непосредственными дочерними элементами <ol>-элементов. Добиться этого можно с помощью следующего кода, где на <li>-элементы, являющиеся непосредственными дочерними элементами <ul>-элементов, стиль применяться не будет:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            ol > li { font-weight:bold: }
        </style>
    </head>
    <body>
        <ol>
            <li>Один</li>
            <li>два</li>
            <li>Три</li>
            <ul>
                <li>Один</li>
                <li>два</li>
                <li>Три</li>
            </ul>
        </ol>
    </body>
</html>
```

Результат загрузки этого HTML-кода в браузер будет иметь следующий вид:

1. **Один**
2. **два**
3. **Три**
   - Один
   - два
   - Три

## Селектор смежных элементов

Селектор смежных элементов похож на селектор дочерних элементов, за исключением того, что вместо применения к родительским и дочерним элементам он применя-

ется к элементам, находящимся на одном и том же уровне и следующим непосредственно друг за другом без каких-либо других элементов, которые располагаются между ними (хотя текст между ними допускается).

Селектор смежных элементов состоит из двух и более селекторов со знаком «плюс» (+) между каждым из них:

```
i + b {color: red: }
```

Этот селектор сделает красным любой текст, выделенный полужирным шрифтом, но только там, где он следует сразу же за элементом, выделенным курсивом. Например, текст между тегами  и  в следующем фрагменте кода будет выведен красным цветом:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            i + b {color: red: }
        </style>
    </head>
    <body>
        <div>Это текст в div-контейнере.
            <i>Это текст, выделенный курсивом.</i>
            А это возвращение к основному тексту.
            <b>А теперь это текст, выделенный полужирным шрифтом,
               который будет отображен красным цветом.</b>
        </div>
    </body>
</html>
```

## Селектор элементов, имеющих идентификатор

Если у элемента есть идентификатор (наподобие следующего: `<div id='mydiv'>`), к нему можно обратиться из CSS напрямую следующим способом, выделяющим весь текст в названном элементе курсивом:

```
#mydiv { font-style:italic: }
```

### Повторное использование идентификаторов

Идентификаторы могут использоваться в документе только один раз, поэтому только первое найденное появление идентификатора приведет к применению нового значения того или иного свойства, заданного правилом CSS. Но в CSS можно непосредственно ссылаться на любые идентификаторы, имеющие одинаковые имена, если они появляются в элементах разного типа:

```
<div id='myid'>Привет</div> <span id='myid'>Привет</span>
```

Поскольку идентификаторы обычно применяются только к уникальным элементам, следующее правило будет применять подчеркивание только к первому появлению `myid`:

```
#myid { text-decoration:underline: }
```

Но можно добиться того, чтобы правило в CSS применялось к обоим появлениям данного идентификатора:

```
span#myid { text-decoration:underline; }
div#myid  { text-decoration:underline; }
```

Или в сокращенной записи (см. далее раздел, посвященный группировке):

```
span#myid.#myid { text-decoration:underline; }
```

Я не рекомендую использовать такую форму селекции, поскольку любой код JavaScript, который также должен обращаться к этим элементам, не сможет с этим справиться, потому что широко применяемая функция getElementById вернет только первое появление элемента с таким идентификатором. Для ссылки на любые другие экземпляры программе придется перебрать весь список элементов в документе, что является куда более сложной задачей. Лучше всегда выбирать для идентификаторов только уникальные имена.

## Селектор класса

Когда на странице имеются элементы, для которых нужно применить один и тот же стиль, всем этим элементам можно задать одно и то же имя класса (например: `<span class='myclass'>`), а затем создать единое правило для одновременного изменения всех этих элементов, как в следующем правиле, создающем смещение левого края на 10 пикселов для всех элементов, которые используют данный класс:

```
.myclass { margin-left:10px; }
```

В современных браузерах HTML-элементы могут использовать более одного класса, если имена классов разделить пробелами, например: `<span class='thisclass thatclass otherclass'>`. Но следует запомнить, что некоторые очень старые браузеры допускают применение в аргументе `class` только одного имени.

### Ограничение области действия класса

Вы можете ограничить область действия класса, указав тип элементов, к которым должно применяться правило. Например, следующее правило применяет настройки только к абзацам, использующим класс `main`:

```
p.main { text-indent:30px; }
```

В данном примере только те абзацы, которые используют класс `main` (как этот: `<p class="main">`), получат новое значение свойства. На любые другие типы элементов, которые могут применять этот класс (такие как `<div class="main">`), это правило распространяться не будет.

## Селектор атрибутов

Многие HTML-теги поддерживают атрибуты, и использование селектора данного типа может избавить вас от применения идентификаторов и классов для ссылок на элементы, задаваемые этими тегами. Например, можно непосредственно со-

слаться на атрибуты следующим образом, установив для всех элементов, задействующих атрибут type="submit", ширину, равную 100 пикселам:

```
[type="submit"] { width:100px; }
```

Если нужно ограничить область действия селектора до, к примеру, элементов ввода, принадлежащих форме и имеющих это значение атрибута типа, можно вместо предыдущего воспользоваться следующим правилом:

```
form input[type="submit"] { width:100px; }
```

Селекторы атрибутов также работают применительно к идентификаторам и классам, например, селектор [class="classname"] работает точно так же, как и селектор .classname (за исключением того, что у последнего из них более высокий уровень приоритета). Точно таким же образом селектор [id="idname"] может использоваться вместо селектора идентификатора #idname. Селекторы классов и идентификаторов, предваряемые символами решетки (#) и точки (.) могут рассматриваться в качестве краткой формы селекторов атрибутов, имеющей при этом более высокий уровень приоритета.

## Универсальный селектор

Групповой символ *, или универсальный селектор соответствует любому элементу, поэтому следующее правило приведет к полному беспорядку в документе, установив зеленое обрамление для всех его элементов:

```
* { border:1px solid green; }
```

Скорее всего, универсальный селектор будет использоваться не сам по себе, а как часть какого-нибудь составного правила, где он будет весьма эффективен. Например, следующее правило будет применять тот же самый стиль, что предыдущее, но только ко всем абзацам, которые являются подчиненными элементами того элемента, у которого имеется идентификатор со значением boxout, и только в том случае, если они не являются непосредственными дочерними элементами:

```
#boxout * p {border:1px solid green; }
```

Разберемся в том, что здесь происходит. Первым селектором, следующим за #boxout является символ звездочки (*), стало быть, он ссылается на любой элемент внутри объекта boxout. Затем следующий селектор p ограничивает фокус селекции, направляя его только на абзацы (что и определяется символом p), являющиеся подчиненными элементами, возвращаемыми селектором *. Поэтому данное CSS-правило приводит к выполнению следующих действий (в которых для ссылки на одни и те же вещи я использую взаимозаменяемые понятия «объект» и «элемент»).

1. Поиск объекта с идентификатором, имеющим значение boxout.
2. Поиск всех подчиненных элементов объекта, возвращенного при выполнении действия 1.
3. Поиск всех подчиненных p-элементов тех объектов, которые были возвращены при выполнении действия 2, и, поскольку это последний селектор в группе, поиск также всех подчиненных p-элементов, подчиняющихся этим подчиненным

элементам (и т. д.) того объекта, который был возвращен при выполнении действия 2.

4. Применение стилей, заданных внутри символов { и } к объектам, возвращенным при выполнении действия 3.

В результате зеленое обрамление применяется только к абзацам, являющимся внучатыми (или правнучатыми и т. д.) элементами основного элемента.

## Групповая селекция

При использовании CSS имеется возможность одновременного применения правила более чем к одному элементу, классу или любому другому типу селектора путем разделения селекторов запятыми. Например, следующее правило поместит пунктирную оранжевую линию под всеми абзацами, элементом с идентификатором idname и всеми элементами, использующими класс со значением classname:

```
p, #idname, .classname { border-bottom:1px dotted orange; }
```

На рис. 18.3 показан результат применения разных селекторов, а рядом показаны применяемые к ним правила.

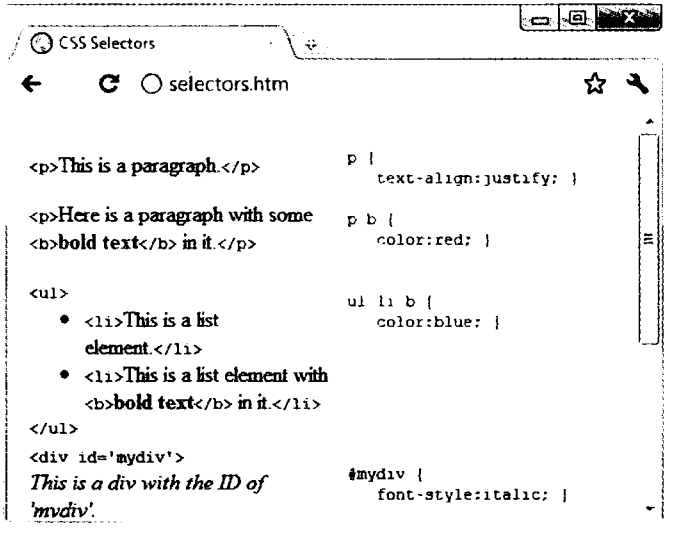



**Рис. 18.3.** Фрагменты кода HTML и применяемые в отношении этих фрагментов правила CSS

# Каскадность CSS

Одной из основополагающих особенностей свойств CSS является их каскадность, благодаря которой технология и называется каскадными таблицами стилей (Cascading Style Sheets). Но что это означает?

Каскадирование является методом, используемым для решения потенциальных конфликтов между различными типами стилей, поддерживаемых браузером, и применения их в порядке приоритетности в зависимости от создателя стилей, от метода, который использован для создания стилей и от типов выбранных свойств.

## Создатель таблиц стилей

Все современные браузеры поддерживают три основных типа таблиц стилей. В порядке приоритетности сверху вниз они располагаются следующим образом.

1. Созданные автором документа.
2. Созданные пользователем.
3. Созданные браузером.

Эти три набора таблиц стилей обрабатываются в обратном порядке. Сначала к документу применяются исходные настройки веб-браузера. Без них веб-страницы, не использующие таблицы стилей, выглядели бы ужасно. Они включают внешний вид, размер и цвет шрифта, интервалы между элементами, обрамления и отступы в таблицах и все остальные разумные стандарты, ожидаемые пользователем.

Затем, если пользователь создал какие-нибудь стили, которые предпочитает использовать в качестве стандартных, эти стили заменяют исходные стили браузера, с которыми они могут конфликтовать.

И наконец, применяются любые стили, созданные автором текущего документа, заменяя любые стили, либо созданные в качестве исходных стилей браузера, либо созданные пользователем.

## Методы создания таблиц стилей

Таблицы стилей могут создаваться с помощью трех различных методов. Если расположить их в порядке приоритетности сверху вниз, получится следующий список.

1. Внедренные стили.
2. Встроенная таблица стилей.
3. Внешняя таблица стилей.

Эти методы создания таблиц стилей также применяются в порядке, обратном порядку их приоритетности. Поэтому сначала обрабатываются все внешние таблицы стилей, и к документу применяются их стили.

Затем обрабатываются любые встроенные стили (которые находятся внутри тегов `<style>...</style>`). Все, что конфликтует с внешними правилами, получает приоритет и заменяет эти правила.

И наконец, наивысший приоритет получают любые стили, применяемые непосредственно к элементу в качестве внедренного стиля (такие как `<div style="...">...</div>`), которые заменяют все предыдущие заданные свойства.

# Селекторы таблиц стилей

Существует три разных способа выбора стилизуемых элементов. В порядке убывания приоритетности их список имеет следующий вид.

1. Обращение по индивидуальному идентификатору или селектор атрибутов.
2. Обращение в группах по классу.
3. Обращение по тегам элементов.

Селекторы обрабатываются согласно количеству и типам элементов, подпадающих под правило, что несколько отличается от предыдущих двух правил разрешения конфликтов. Причина состоит в том, что правила не должны сразу применяться только к одному типу селектора и могут иметь отношение к разным селекторам.

Таким образом, метод, необходимый для определения уровня приоритета правил, может содержать любую комбинацию селекторов. Это делается с помощью вычисления специфики каждого правила путем выстраивания их в порядке убывания области действия.

## Вычисление специфики

Специфика правила вычисляется путем создания трехкомпонентных чисел на основе типов селекторов в показанном выше пронумерованном списке. Эти составные числа сначале выглядят как [0,0,0]. При обработке правила каждый селектор, который ссылается на идентификатор, увеличивает первое число на единицу, и составное число приобретает вид [1,0,0].

Посмотрим на следующее правило. У него имеется три ID-ссылки (#heading, #main и #menu), поэтому составное число приобретает вид [3,0,0]:

```
#heading, #main, #menu,
.text, .quote, .boxout, .news, .comments,
p, blockquote {
    font-family: 'Times New Roman';
    font-size  : 14pt; }
```

Затем во вторую часть составного числа помещается количество селекторов, ссылающихся на класс. В этом примере таких селекторов пять (.text, .quote, .boxout, .news и .comments), поэтому число приобретает вид [3,5,0].

И наконец, вычисляется количество селекторов, ссылающихся на теги элементов, и результат помещается в последнюю часть составного числа. В нашем примере таких селекторов два (p и blockquote), поэтому составное число приобретает следующий окончательный вид: [3,5,2], чего вполне достаточно для сравнения специфики этого правила с другими спецификами.

Когда в составном числе набирается девять или меньше селекторов каждого типа, его можно преобразовать непосредственно в десятичное число, в нашем случае это 352. Правила с меньшим числом, чем это, будут иметь меньший приоритет, а правила с более высоким числом будут иметь больший приоритет. Когда у двух правил будет одно и то же значение, выигрывает последнее из применявшихся.

## [illegible]ование другой системы счисления

[illegible] числе набирается больше девяти типов селекторов, нужно перехо[illegible] старшую систему счисления. Например, составное число [11,7,19] не [illegible] образованию в десятичное простым объединением трех частей. Вместо [illegible] можно преобразовать в число с более высоким основанием системы счисления, [illegible] нованием 20 (или выше, если будет больше 19 селекторов любого типа). [illegible] нужно умножить все три части и сложить результаты, как показано [illegible] с самого правого числа и переходя влево:

```
[illegible] Ч 19 = 380
[illegible] Ч  7 = 2800
[illegible] Ч 11 = 88000
[illegible]ом виде = 91180
```

[illegible] нужно использовать более высокое основание, замените значения 20 сле[illegible] значениями применяемого основания. Затем, когда все составные числа набора [illegible] прошли преобразование из этого основания в десятичное, будет просто [illegible] специфику, а стало быть, и уровень приоритета каждого.

К счастью, процессор CSS делает все это за вас, но понимание принципов данной [illegible] поможет правильно создавать правила и разбираться в тех уровнях прио[illegible] которые у них будут.

Если все предыдущие вычисления кажутся слишком сложными, вам будет приятно узнать, что в большинстве случаев вы можете обойтись простым практическим правилом: чем меньше изменяемых элементов и чем конкретнее они указаны, тем выше будет приоритет, присваиваемый правилу.

## Одни правила бывают равнее других

Когда два или более правила задания стилей имеют абсолютно одинаковый уровень приоритета, то по умолчанию будет применяться последнее обработанное правило. Но вы можете придать правилу более высокий уровень приоритета по сравнению с другими равными ему правилами, используя объявление !important:

```
color:#ff0000 !important; }
```

При этом все предыдущие равные настройки заменяются (даже те, в которых используется объявление !important), и любые равные правила, обрабатываемые позже, игнорируются. Например, второе из двух следующих правил обычно имело бы приоритет, но из-за применения объявления !important в ранее заданном правиле второе правило игнорируется:

```
color:#ff0000 !important; }
color:#ffff00 }
```

Пользовательские таблицы стилей могут создаваться для определения исходных стилей браузера, и в них может применяться объявление !important. В этом случае пользовательская настройка стиля будет иметь преимущество над аналогичными свойствами, указанными на текущей веб-странице. Но на очень старых браузерах, использующих CSS1, эта особенность не поддерживается.

# Разница между  и 

Оба элемента  и  относятся к контейнерам, но некоторые качества у них отличаются. По умолчанию -элемент имеет бесконечную ширину (как минимум до края браузерного экрана), которую можно увидеть, если применить к контейнеру единичное обрамление:

```
<div style="border:1px solid green;">Привет</div>
```

А -элемент не шире того текста, который в нем содержится. Поэтому следующий код HTML создаст обрамление только вокруг слова Привет, и оно не будет расширяться до правого края браузерного экрана:

```
<span style="border:1px solid green;">Привет</span>
```

Кроме того, -элемент сопровождает текст или другие объекты и при их переносе на следующие строки, и поэтому может иметь довольно сложное обрамление. Например, в примере 18.2, код CSS использован для создания желтого фона для всех -элементов, для создания голубого фона для всех -элементов и для добавления обрамления и к тем и к другим, перед тем как создать несколько примеров - и -блоков.

**Пример 18.2.** Пример контейнеров  и 

```
<!DOCTYPE html>
<html>
    <head>
        <title>Пример div и span</title>
        <style>
            div, span { border:1px solid black; }
            div { background-color:yellow; }
            span { background-color:cyan; }
        </style>
    </head>
    <body>
        <div>Этот текст находится внутри тега div</div>
        А этот нет. <div>А этот снова внутри тега div.</div><br />

        <span> Этот текст находится внутри тега span.</span>
        А этот нет. <span> А этот снова внутри тега span.</span><br /><br />

        <div>Это более объемный текст в теге div, который переносится
        на следующую строку браузера </div><br />

        <span> Это более объемный текст в теге span, который переносится
        на следующую строку браузера </span>
    </body>
</html>
```

Как выполнение кода этого примера выглядит в окне браузера, показано на рис. 18.4. Хотя в печатной версии книги все изображается в серых тонах, на рисунке четко показано, как -элементы расширяются до правого края браузерного

экрана, и заставляют следующее за ними содержимое появляться с начала первой доступной позиции ниже себя.

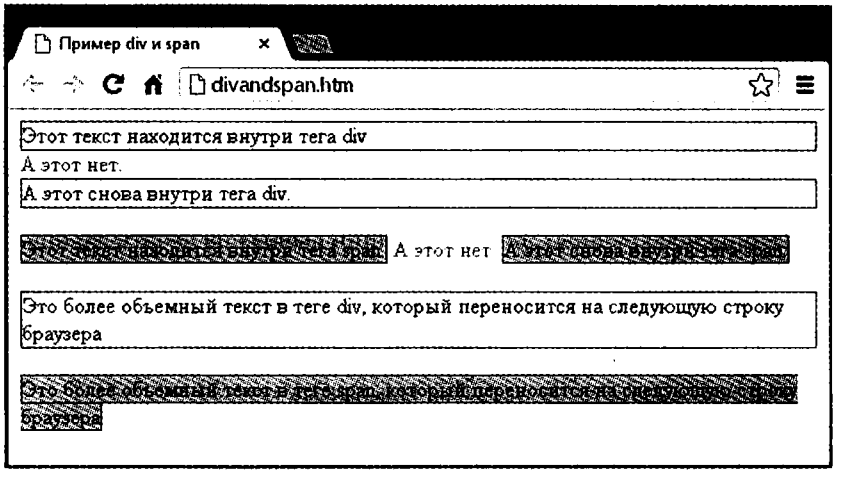



**Рис. 18.4.** Разнообразные элементы, имеющие разную ширину

На рисунке также отображено, как ведут себя -элементы, занимая только то пространство, которое требуется для размещения их содержимого, не заставляя при этом последующее содержимое страницы появляться под ними.

Например, в двух самых нижних примерах, показанных на рисунке, можно увидеть, что когда -элементы при достижении края экрана переносятся на новую строку, они сохраняют прямоугольную форму, в то время как -элементы просто следуют за потоком содержащегося в них текста (или другого содержимого).

Поскольку -теги могут создавать только прямоугольные контейнеры, они лучше подходят для содержания таких объектов, как изображения, блоки, цитаты и т. д., а контейнеры, создаваемые -тегами, больше подходят для содержания текста или других атрибутов, размещаемых один за другим на одной линии, и которые должны располагаться слева направо (или в некоторых языках справа налево).

# Измерения

CSS поддерживает впечатляющий диапазон различных единиц измерения, позволяя очень точно выкраивать веб-страницы для конкретных значений или относительных размеров. Я обычно пользуюсь следующими единицами измерений (и считаю, что вам они также будут наиболее полезными) — это пикселы, пункты, эмы и проценты. Рассмотрим подробнее эти, а также другие единицы измерения.

- *Пиксел (pixel)* — размер пиксела варьируется в соответствии с размерами и глубиной пиксела на пользовательском мониторе. Один пиксел равен ширине и высоте

отдельной точки на экране, поэтому данную единицу измерения лучше всего использовать для мониторов. Например:

```
.classname { margin:5px; }
```

- *Пункт (point)* — равен по размеру 1/72 дюйма. Эта единица измерения пришла из полиграфии и лучше всего подходит для этой среды, но также широко используется и для мониторов. Например:

```
.classname { font-size:14pt; }
```

- *Дюйм (inch)* — равен 72 пунктам, и он также относится к типу единиц измерения, наиболее приспособленных для организации вывода на печать. Например:

```
.classname { width:3in; }
```

- *Сантиметр (centimeter)* — являются еще одной единицей измерения, которая наиболее пригодна для организации вывода на печать. Один сантиметр немного превышает по размеру 28 пунктов. Например:

```
.classname { height:2cm; }
```

- *Миллиметр (millimeter)* — это 1/10 сантиметра (или почти 3 пункта). Миллиметры являются еще одной единицей измерения, наиболее подходящей для организации вывода на печать. Например:

```
.classname { font-size:5mm; }
```

- *Пика (pica)* — является еще одной типографской единицей измерения, равной 12 пунктам. Например:

```
.classname { font-size:1pc; }
```

- *Эм (em)* — равен текущему размеру шрифта (ширине латинской буквы `m`). Является одной из наиболее полезных единиц измерения для CSS, поскольку используется для описания относительных размеров. Например:

```
.classname { font-size:2em; }
```

- *Экс (ex)* — также относится к текущему размеру шрифта. Он равен высоте буквы `x` нижнего регистра. Это менее популярная единица измерения, которая наиболее часто используется в качестве хорошего приблизительного значения, помогающего установить ширину прямоугольного блока, который будет содержать некий текст. Например:

```
.classname { width:20ex; }
```

- *Процент (percent)* — эта единица сродни эму (em), поскольку это относительная, а не абсолютная единица измерения. Когда используется шрифт, как в случае с `1em`, 100 % равны текущему размеру шрифта. Когда эта единица не относится к шрифту, она относится к размеру того контейнера, к которому применяется данное свойство. Например:

```
.classname { height:120%; }
```

На рис. 18.5 каждый из этих типов измерений показа по очереди применительно к отображаемому тексту почти одинаковых размеров.

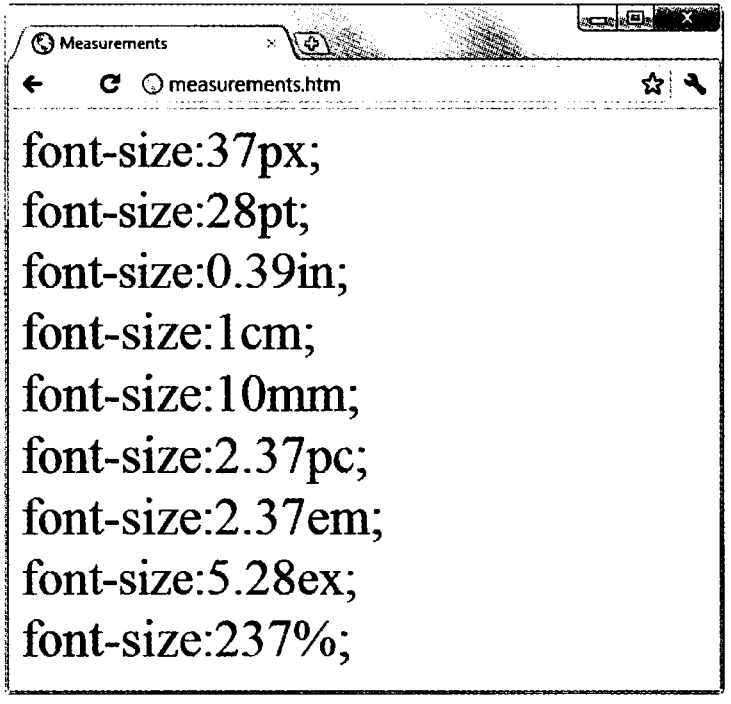



**Рис. 18.5.** Различные измерения, приводящие примерно к одинаковому результату

# Шрифты и оформление

С помощью CSS можно настроить четыре основных свойства шрифта: семейство — `family`, стиль — `style`, размер — `size` и насыщенность — `weight`. Пользуясь этими свойствами, можно точно настроить способ отображения текста на ваших веб-страницах и (или) вывода его на печать и т. д.

## font-family

Это свойство назначает используемый шрифт. Оно также поддерживает перечисление множества шрифтов в порядке предпочтения слева направо, чтобы стилевое оформление при отсутствии у пользователя установленного предпочитаемого шрифта постепенно снижалось в сторону менее предпочитаемых шрифтов. Например, для установки шрифта по умолчанию для абзацев можно воспользоваться следующим CSS-правилом:

```
p { font-family:Verdana, Arial, Helvetica, sans-serif; }
```

Если название шрифта состоит из двух и более слов, его нужно заключить в кавычки:

```
p { font-family:"Times New Roman", Georgia, serif; }
```

Для использования на веб-страницах больше всего подходит такое семейство шрифтов, как Arial, Helvetica, Times New Roman, Times, Courier New и Courier, поскольку все эти шрифты доступны практически на всех веб-браузерах и операционных системах. Шрифты Verdana, Georgia, Comic Sans MS, Trebuchet MS, Arial Black и Impact можно смело применять на Mac и PC, но они могут быть не установлены в других операционных системах, таких как Linux. Другими распространенными, но менее надежными шрифтами являются Palatino, Garamond, Bookman и Avant Garde. Если используется один из менее надежных шрифтов, нужно убедиться что в ваших настройках CSS предложены один или несколько менее предпочтительных шрифтов, чтобы веб-страницы в отсутствие на браузерах предпочитаемых вами шрифтов шли в их использовании по нисходящей.

На рис. 18.6 показано применение этих двух наборов CSS-правил.


**Рис. 18.6.** Выбор семейства шрифтов

# font-style

С помощью этого свойства можно выбрать вывод шрифта в обычном — `normal`, курсивном — `italic` или наклонном — `oblique` виде (последний похож на курсив и обычно используется для гарнитуры sans-serif). Следующие правила создают три класса (`normal`, `italic` и `oblique`), которые могут применяться к элементам для создания соответствующих эффектов:

```
.normal  { font-style:normal;  }
.italic  { font-style:italic;  }
.oblique { font-style:oblique; }
```

# font-size

В предыдущем разделе, касающемся единиц измерения, было рассмотрено множество способов изменения размера шрифта, но все они сводятся к двум основным типам:

фиксированным и относительным. Фиксированная настройка похожа на следующее правило, которым для абзацев устанавливается размер шрифта, равный 14 пунктам:

```
p { font size:14pt: }
```

В качестве альтернативы можно предпочесть работу с текущим размером шрифта по умолчанию, используя его для стилевого решения таких разновидностей текста, как заголовки. В следующих правилах определены относительные размеры некоторых заголовков, где тег <h4> начинает с прибавки в 20 % к размеру по умолчанию, и каждое возрастание размера задается больше предыдущего на 40 %:

```
h1 { font-size:240%: }
h2 { font-size:200%: }
h3 { font-size:160%: }
h4 { font-size:120%: }
```

На рис. 18.7 показана подборка размеров шрифтов в действии.

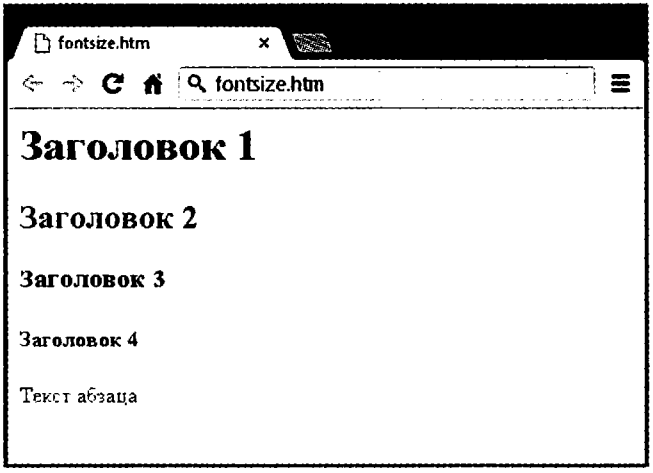



**Рис. 18.7.** Настройка четырех размеров заголовков и размер абзаца, используемый по умолчанию

## font-weight

Используя это свойство, можно задать насыщенность или степень жирности шрифта. Оно поддерживает несколько значений, но в основном востребованы `normal` и `bold`:

```
.bold { font-weight:bold: }
```

# Управление стилями текста

Независимо от используемого шрифта в способ вывода текста можно внести дополнительные изменения, меняя его оформление — `decoration`, разрядку — `spacing` и выравнивание — `alignment`. Но свойства текста и шрифта перекликаются, в том

смысле что курсивный и полужирный текст можно получить, используя свойства font-style и font-weight, в то время как другой текст, например подчеркнутый, требует использования свойства text-decoration.

## Оформление

Используя свойство text-decoration, можно применить к тексту такие эффекты, как подчеркивание — underline, перечеркивание — line-through, верхнее подчеркивание — overline и мигание — blink. Следующее правило создает новый класс по имени over, который применяет верхнее подчеркивание к тексту (насыщенность линий, используемых для подчеркивания снизу и сверху и для перечеркивания, будет соответствовать насыщенности шрифта):

```
.over { text-decoration:overline; }
```

На рис. 18.8 можно увидеть подборку стилей, насыщенностей и оформлений шрифтов.

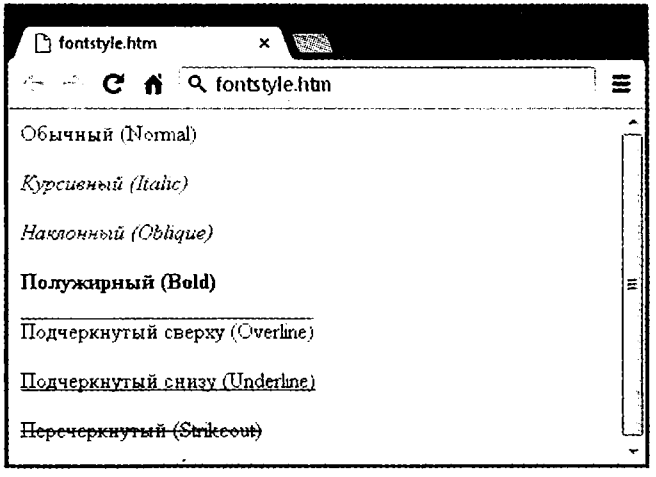



**Рис. 18.8.** Примеры доступных правил стилевых решений и оформления

## Разрядка

Существуют свойства, позволяющие изменить разрядку строк, слов и букв. Например, следующие правила настраивают разрядку строк для абзацев путем изменения свойства line-height, делая его на 25 % больше, устанавливают свойство word-spacing равным 30 пикселам и разрядку букв 3 пиксела:

```
p {
    line-height   :125%;
    word-spacing  :30px;
    letter-spacing:3px; }
```

## Выравнивание

В CSS доступны четыре типа выравнивания текста: по левому краю – left, по правому краю – right, по центру – center и по ширине содержимого – justify. В следующем правиле текст абзаца настроен на полное выравнивание по ширине:

```
p { text-align:justify; }
```

## Преобразование

Для преобразования текста доступны четыре свойства: отсутствие преобразования – none, преобразование первых букв слов в заглавные – capitalize, преобразование всех букв в заглавные – uppercase и преобразование всех букв в строчные – lowercase. Следующее правило создает класс по имени upper, гарантирующий при его применении вывод всего текста в верхнем регистре:

```
.upper { text transform:uppercase; }
```

## Отступы

С помощью свойства text-indent можно создать отступ первой строки блока текста на указанную величину. Следующее правило создает отступ первой строки каждого абзаца на 20 пикселов, но могут быть применены и другие единицы измерений или процентное увеличение:

```
p { text-indent:20px; }
```

На рис. 18.9 к блоку текста было применено следующее правило:

```
p {          line-height  :150%;
             word-spacing  :10px;
             letter-spacing:1px;       }
.justify   { text-align     :justify;  }
.uppercase { text transform:uppercase; }
.indent    { text indent    :20px;     }
```

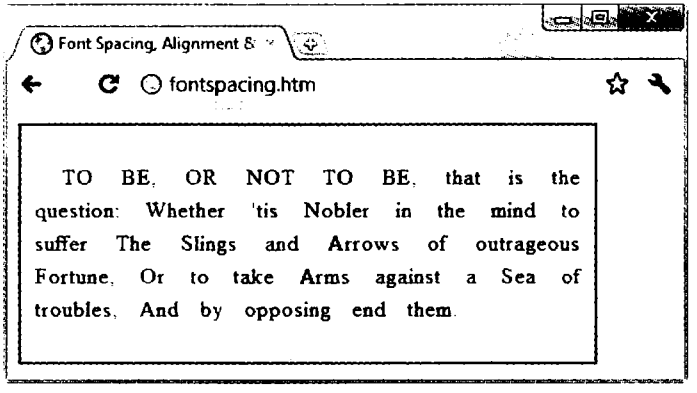



**Рис. 18.9.** Примененные правила отступа, преобразования в верхний регистр и разрядки

# Цвета CSS

Цвета могут применяться к первому плану, а также к фону текста и объектов, путем использования свойства цвета — color и фонового цвета — background-color (или путем предоставления единственного аргумента свойству фона — background). Указанный цвет может быть одним из именованных цветов (например, red или blue), цветом, составленным из трех шестнадцатеричных чисел RGB (например, #ff0000 или #0000ff), или цветом, составленным с использованием CSS-функции rgb.

Названия стандартных 16 цветов, определенных организацией по стандартам W3C (http://www.w3.org), следующие: аквамарин — aqua, черный — black, синий — blue, яркий пурпурно-красный, или фуксия, — fuchsia, серый — gray, зеленый — green, яркий светло-зеленый — lime, красно-коричневый — maroon, темно-синий — navy, оливковый — olive, фиолетовый — purple, красный — red, серебристый — silver, зеленовато-голубой — teal, белый — white и желтый — yellow. Следующее правило использует одно из этих названий для установки фонового цвета для объекта с ID, имеющим значение object:

```
#object { background-color:silver; }
```

В показанном ниже правиле фоновый цвет текста во всех -элементах установлен желтым (поскольку на компьютерном дисплее шестнадцатеричные уровни ff красного, плюс ff зеленого, плюс 00 синего составляют желтый цвет):

```
div { color:#ffff00; }
```

Или, если не хочется работать с шестнадцатеричными числами, можно указать свои три цветовые составляющие с помощью функции rgb, как в следующем правиле, которое изменяет фоновый цвет текущего документа на аквамарин:

```
body { background-color:rgb(0, 255, 255); }
```

Если вы не хотите работать в диапазоне 256 уровней для каждого основного цвета, можете вместо них в функции rgb использовать процентный показатель со значениями в диапазоне от самого низкого (0) количества основного цвета до самого высокого (100), например: rgb(58%, 95%, 74%). Можно также для более тонкого управления цветом применять числа с плавающей точкой, например: rgb(23.4%, 67.6%, 15.5%).

# Сокращенные цветовые строки

Есть также короткая форма строки шестнадцатеричных чисел, в которой для каждого цвета используется только первое число из каждой двухбайтовой пары. Например, вместо назначения цвета #fe4692 можно применять #f49, где опущены вторые шестнадцатеричные цифры из каждой пары, что равно цветовому значению #ff4499.

Получается почти такой же цвет. Это можно применить там, где точный цвет не нужен. Разница между строками из шести и из трех цифр в том, что первые поддерживают шестнадцать миллионов различных цветов, а вторые — всего четыре тысячи.

Там, где используется такой цвет, как #883366, полным эквивалентом ему будет #836 (поскольку повторяющиеся цифры подразумеваются в сокращенной версии), и для создания одного и того же цвета можно применять любую строку.

## Градиенты

Вместо использования сплошного цветового фона можно применить градиент, который будет автоматически переходить от выбранного исходного цвета до выбранного конечного цвета. Градиент лучше использовать в связке с простым цветовым правилом, чтобы браузеры, не поддерживающие градиенты, отображали как минимум сплошной цвет.

В примере 18.3 задействуется правило отображения оранжевого градиента (или просто обычного оранжевого цвета на браузерах, не поддерживающих градиенты), как показано в средней части рис. 18.10.

**Пример 18.3.** Создание линейного градиента

```
<!DOCTYPE html>
<html>
    <head>
        <title>Создание линейного градиента</title>
        <style>
            .orangegrad {
                background:orange;
                background:linear-gradient(top, #fb0, #f50);
                background:-moz-linear-gradient(top, #fb0, #f50);
                background:-webkit-linear-gradient(top, #fb0, #f50);
                background:-o-linear-gradient(top, #fb0, #f50);
                background:-ms-linear-gradient(top, #fb0, #f50); }
        </style>
    </head>
    <body>
        <div class='orangegrad'>Черный текст <br />
        на оранжевом <br />линейном градиенте</div>
    </body>
</html>
```

Как показано в предыдущем примере, многие CSS-правила требуют префиксы, предназначенные для того или иного браузера, например -moz-, -webkit-, -o- и -ms- (соответственно для браузеров на основе движка Mozilla, таких как Firefox; для браузеров на основе движка WebKit, таких как Apple Safari, Google Chrome, и браузеров iOS Android, а также для браузеров Opera и Microsoft). Основные CSS-правила и атрибуты, а также указания на то, где требуются версии, подстроенные под тот или иной браузер, перечислены на веб-сайте http://caniuse.com.

Для создания градиента нужно выбрать, где он будет начинаться, вверху (`top`), внизу (`bottom`), слева (`left`) справа (`right`) или по центру (`center`), или в любых составных местах, например в левом верхнем углу (`top left`) или от центра вправо (`center right`), затем ввести нужные начальный и конечный цвета и применить правило либо линейного (`linear-gradient`), либо радиального (`radial-gradient`) градиента, обеспечив правила для всех браузеров, на которые вы нацелились.

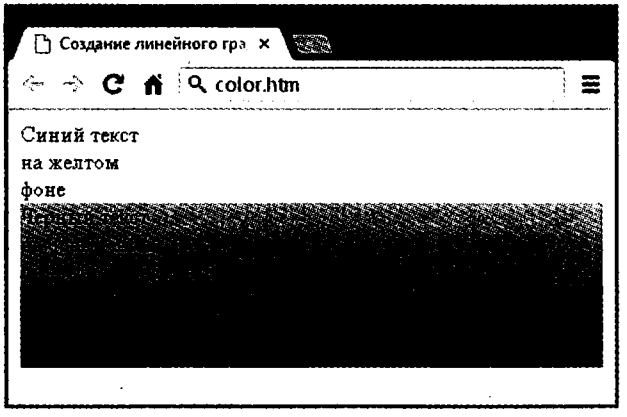



**Рис. 18.10.** Сплошной цвет фона, а также линейный и радиальный градиенты

Вы также можете использовать не только начальный и конечный цвета, а предоставлять между ними в качестве дополнительных аргументов составляющие конечные цвета. Например, если предоставлены пять аргументов, каждый из них будет управлять изменением цвета одной пятой области (в соответствии с его местом в списке аргументов).

# Позиционирование элементов

Элементы попадают на веб-страницу туда, где они находятся в документе, но могут перемещаться путем изменения свойства позиции элемента от исходной статической до абсолютной, относительной или фиксированной.

## Абсолютное позиционирование

Элемент с абсолютным позиционированием удаляется из документа, и любые другие элементы, которые в состоянии это сделать, займут освободившееся пространство. Затем вы можете позиционировать объект в любое нужное место в документе, используя свойства верх `top`, право `right`, низ `bottom` и лево `left`. Он остается над (или под) другими элементами.

Например, для перемещения объекта с ID, имеющим значение `object`, в абсолютное место, находящееся на 100 пикселов ниже начала документа и на 200 пикселов от левого края, к нему нужно применить следующие правила (вы также можете использовать любые другие единицы измерения, поддерживаемые CSS):

```
#object {
    position:absolute;
    top      100px;
    left     200px; }
```

## Относительное позиционирование

Подобным образом можно переместить объект относительно того места, которое он бы занимал при обычном ходе формирования документа. Так, например, для перемещения объекта на 10 пикселов вниз и 10 пикселов вправо от его обычного положения нужно воспользоваться следующими правилами:

```
=object {
    position:relative;
    top     :10px;
    left    :10px; }
```

## Фиксированное позиционирование

Заключительные настройки свойства позиционирования позволяют переместить объект в абсолютное положение, но только внутри окна просмотра текущего браузера. Затем при прокрутке документа объект остается именно там, куда он был помещен, а основной документ будет прокручиваться под ним — это неплохой способ создания док-панелей и других подобных устройств. Для фиксирования объекта в левом верхнем углу браузера нужно воспользоваться следующими правилами:

```
#object {
    position:fixed;
    top     :0px;
    left    :0px; }
```

## Сравнение типов позиционирования

В примере 18.4 показано, как могут использоваться эти разные значения позиционирования.

**Пример 18.4.** Применение разных значений позиционирования

```
<!DOCTYPE html>
<html>
    <head>
        <title>Позиционирование</title>
        <style>
            #object1 {
                position  :absolute;
                background:pink;
                width     :100px;
                height    :100px;
                top       :100px;
                left      :0px; }
            #object2 {
                position  :relative;
                background:lightgreen;
                width     :100px;
                height    :100px;
                top       :-8px;
```

```
            left      :110px; }
        #object3 {
            position  :fixed;
            background:yellow;
            width     :100px;
            height    :100px;
            top       :100px;
            left      :236px; }
    </style>
</head>
<body>
    <br /><br /><br /><br /><br />
    <div id='object1'>Абсолютное позиционирование</div>
    <div id='object2'>Относительное позиционирование</div>
    <div id='object3'>Фиксированное позиционирование</div>
</body>
</html>
```

На рис. 18.11 код примера 18.4 был загружен в браузер, и окно браузера было уменьшено по ширине и высоте, чтобы появилась необходимость в прокрутке вниз, чтобы увидеть всю веб-страницу. На рисунке элемент с фиксированным позиционированием, который изначально находился на одной линии с другими двумя элементами, остался на месте, в то время как другие элементы были прокручены вверх по странице, и теперь этот элемент оказался смещенным ниже двух других элементов.

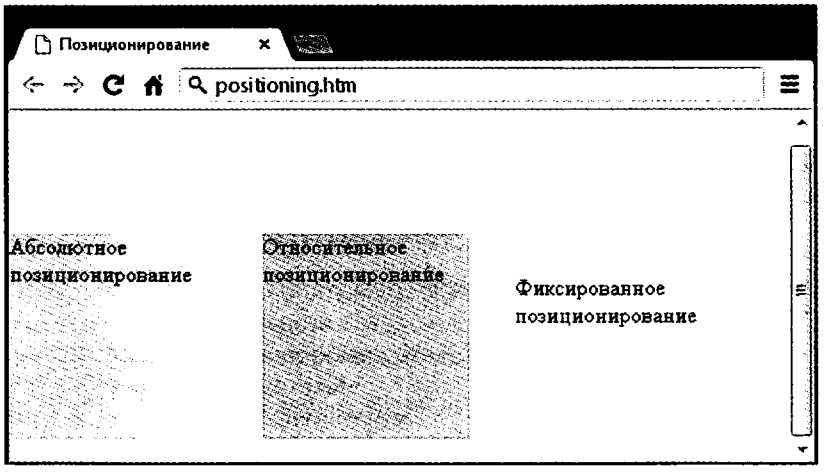



**Рис. 18.11.** Использование разных значений позиционирования

Если вы наберете код этого примера (или загрузите его с сопровождающего веб-сайта) и загрузите этот код в свой браузер, то увидите, что элемент с фиксированным позиционированием остается на месте даже при прокрутке страницы. Вы также можете заметить, что элемент с абсолютным позиционированием расположен точно на 100 пикселов ниже с нулевым горизонтальным смещением, а элемент

с относительным позиционированием фактически переместился вверх на 8 пикселов, а затем сместился от левого края на 110 пикселов, чтобы выстроиться в линию рядом с первым элементом.

# Псевдоклассы

Существуют селекторы и классы, используемые только внутри таблиц стилей и не имеющие каких-либо соответствующих тегов или атрибутов в HTML. Их задача заключается в том, чтобы классифицировать элементы, используя характеристики, отличные от их имен, атрибутов или содержимого, то есть характеристики, которые не могут быть прослежены по дереву документа. К их числу относятся такие псевдоклассы, как `first-line`, `first-child` и `first-letter`.

Псевдоклассы отделяются от элементов с помощью символа двоеточия (:). Например, для создания класса по имени `bigfirst` для выделения первой буквы элемента можно воспользоваться следующим правилом:

```
.bigfirst:first-letter {
    font-size:400%;
    float    :left; }
```

Когда класс `bigfirst` применяется к текстовым элементам, первая буква будет отображаться сильно увеличенной, а остальной текст будет показан в обычном размере, аккуратно ее обтекая (благодаря свойству `float`), как будто первая буква является изображением или другим объектом. В число других псевдоклассов входят `hover`, `link`, `active` и `visited`. Все они наиболее полезны применительно к anchor-элементам, как показано в следующих правилах, которые устанавливают для ссылок в качестве исходного синий цвет, а для посещенных ссылок — светло-синий цвет:

```
a:link    { color:blue;      }
a:visited { color:lightblue; }
```

Следующая пара правил интересна тем, что в них используется псевдокласс `hover`, поэтому они применяются только при помещении указателя мыши над элементом. В этом примере они изменяют в ссылке цвет текста на белый на красном фоне, предоставляя динамический эффект, который можно было бы ожидать только от использования кода JavaScript:

```
a:hover {
    color      :white;
    background:red; }
```

Здесь вместо более длинного свойства цвета фона — `background-color` я использовал свойство фона — `background` с единственным аргументом.

Псевдокласс `active` также имеет динамический характер, выражающийся в том, что он влияет на изменение ссылки в промежутке времени между щелчком кнопкой мыши и освобождением этой кнопки, как в следующем правиле, изменяющем цвет ссылки на темно-синий:

```
a:active { color:darkblue; }
```

Еще одним интересным динамическим псевдоклассом является focus, который применяется, только когда элемент получает фокус путем его выбора пользователем посредством клавиатуры или мыши. Следующее правило применяет универсальный селектор, чтобы всегда помещать светло-серую пунктирную границу толщиной 2 пиксела вокруг объекта, имеющего фокус:

```
*:focus { border:2px dotted #888888; }
```

Как показано на рис. 18.12, код примера 18.5 выводит две ссылки и поле ввода. Первая ссылка показана серым цветом, поскольку она уже посещалась в этом браузере, а вторая ссылка еще не посещалась и показана синим цветом. Была нажата клавиша Tab, и фокусом ввода теперь служит поле ввода, поэтому цвет его фона поменялся на желтый. Когда будет щелчок кнопкой мыши на любой из ссылок, она отобразится фиолетовым цветом, а когда над ней будет проходить указатель мыши, она будет показана красным цветом.

**Пример 18.5.** Псевдоклассы link и focus

```
<!DOCTYPE html>
<html>
    <head>
        <title>Псевдоклассы</title>
        <style>
            a:link    { color:blue;       }
            a:visited { color:gray;       }
            a:hover   { color:red;        }
            a:active  { color:purple;     }
            *:focus   { background:yellow; }
        </style>
    </head>
    <body>
        <p>Для перемещения фокуса по элементам нажимайте
           клавишу Tab (и Shift+Tab)</p>  <br /> <br />
        <a href='http://google.com'>Ссылка на Google'</a><br />
        <a href='nowhere'>Ссылка вникуда'</a><br />
        <input type='text' />
    </body>
</html>
```

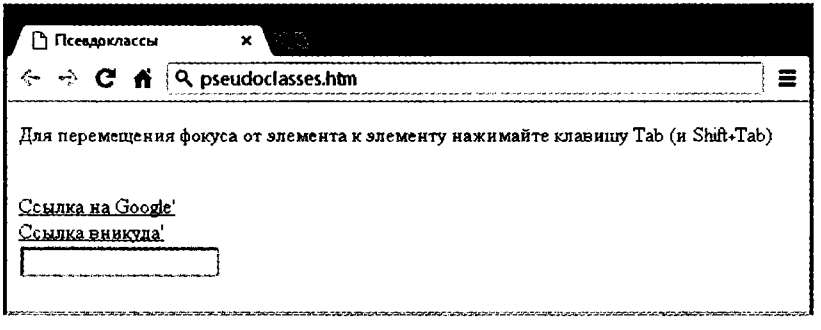



**Рис. 18.12.** Псевдоклассы, примененные к подборке элементов

Доступны также и другие псевдоклассы, дополнительную информацию о них можно получить на следующем веб-сайте: http://tinyurl.com/pseudoclasses.

Применять псевдокласс focus к универсальному селектору *, как показано в этом примере, нужно с большой осторожностью — Internet Explorer воспринимает документ, не имеющий фокуса, как уже имеющий фокус, применяемый ко всей веб-странице, и (в этом случае) будет желтой вся страница, пока не будет нажата клавиша Tab, или же фокус не перейдет на один из элементов страницы.

# Псевдоэлементы

Псевдоэлементы являются средствами добавления к элементу не стиля, а содержимого. Они включаются путем установки двоеточия после спецификатора типа элемента, за которым следует псевдоэлемент. Например, для помещения какого-нибудь текста перед элементом, использующим класс `offer`, можно применить следующее правило:

```
offer:before { content='Special Offer! '; }
```

Теперь элемент, использующий класс `offer`, будет иметь строку, которая предоставлена свойству `content` и отображается перед этим элементом. Точно так же можно использовать псевдоэлемент `:after` для помещения чего-либо после всех ссылок (например), как в следующем коде, благодаря которому за всеми ссылками на странице будет следовать изображение `link.gif`:

```
a:after { url(link.gif); }
```

# Сокращенная запись правил

Для экономии пространства группы родственных CSS-свойств могут объединяться в простое сокращенное назначение. Например, я уже несколько раз использовал сокращение для создания границы, как в правиле `focus` в предыдущем разделе:

```
*:focus { border:2px dotted #888888; }
```

На самом деле это сокращенное объединение следующего набора правил:

```
*:focus {
    border-width:2px;
    border-style:dotted;
    border-color:#ff8800; }
```

При использовании сокращенной записи правила нужно только применить свойства к тому пункту, у которого следует изменить значения. Для установки только ширины и стиля границы без изменения ее цвета можно также использовать следующее правило:

```
* focus { border:2px dotted; }
```

Порядок размещения свойств в сокращенной записи правила может играть важную роль, и их неправильная расстановка приводит обычно к неожиданным результатам. Поскольку изложить многочисленные подробности в данной главе не представляется возможным, при необходимости воспользоваться сокращенной записью CSS вы сможете найти описание свойств, используемых по умолчанию, и порядок их применения в руководстве по CSS или в какой-нибудь поисковой системе. Для начала могу порекомендовать зайти на следующий веб-сайт: http://tinyurl.com/shcss.

# Модель блока и макет страницы

Свойства CSS, влияющие на макет страницы, основаны на модели блока (более подробно этот вопрос рассмотрен в главе 19), вложенном наборе свойств, окружающем элемент. Фактически такие свойства есть (или могут быть) у всех элементов, включая тело документа, чье поле вы можете (к примеру) удалить с помощью следующего правила:

```
body { margin:0px; }
```

Модель блока объекта начинается снаружи, с поля объекта. Внутри него находится граница, затем следует отступ содержимого от границы. И наконец, следует содержимое объекта.

Если приобрести навыки работы с моделью блока, то можно существенно продвинуться на пути создания профессионально спланированных страниц, поскольку даже только эти свойства во многом определяют стилевое оформление страницы.

## Установка полей

Поле является самым крайним уровнем модели блока. Оно отделяет элементы друг от друга и требует разумного использования. Предположим, к примеру, что вы решили выбрать по умолчанию поле в 10 пикселов вокруг каждого из нескольких элементов. Если расположить их друг над другом, то в результате сложения полей между ними получится разрыв в 20 пикселов.

Чтобы устранить эту потенциальную проблему, когда два элемента с полями позиционируются непосредственно один над другим, используется только самое большое из двух полей для отделения их друг от друга. Если оба поля имеют одинаковую ширину, используется только одна ширина. С помощью этого вы, скорее всего, добьетесь желаемого результата. Но при этом следует иметь в виду, что поля элементов с заданным абсолютным позиционированием или встраиваемых элементов не подвергаются поглощению другими полями.

Поля элемента могут быть изменены целиком с помощью свойства `margin` или отдельно друг от друга с помощью свойств `margin-left`, `margin-top`, `margin-right` и `margin-bottom`. При установке свойства `margin` можно предоставить один, два, три или четыре аргумента, в результате чего получится эффект, прокомментированный в следующих правилах:

```
 * Установка всех полей шириной 1 пиксел */
margin:1px;

 * Установка верхнего и нижнего полей шириной 1 пиксел,
   а левого и правого - 2 пиксела */
margin:1px 2px;

 * Установка верхнего поля шириной 1 пиксел, левого и правого - 2 пиксела
   и нижнего - 3 пиксела */
margin:1px 2px 3px;

 * Установка верхнего поля шириной 1 пиксел, правого - 2, нижнего - 3
   и левого - 4 пиксела */
margin:1px 2px 3px 4px;
```

На рис. 18.13 показан код примера 18.6, загруженный в браузер, где правило свойства `margin` (выделенное в коде полужирным шрифтом) применяется к прямоугольным элементам, помещенным внутри элемента `<table>`. Размер таблицы не задан, поэтому она будет просто охватывать как можно плотнее внутренний `<div>`-элемент. Вследствие этого сверху будет поле шириной 10 пикселов, справа — поле шириной 20 пикселов, снизу — поле шириной 30 пикселов и слева — поле шириной 40 пикселов.

**Пример 18.6.** Порядок применения полей

```
<!DOCTYPE html>
<html>
    <head>
        <title>Поля</title>
        <style>
            #object1 {
                background  :lightgreen;
                border-style:solid;
                border-width:1px;
                font-family :Courier New;
                font-size   :9px;
                width       :100px;
                height      :100px;
                padding     :5px;
                margin      :10px 20px 30px 40px; }
        </style>
    </head>
    <body>
        <table border='1' cellpadding='0' cellspacing='0' bgcolor='cyan'>
            <tr>
                <td>
                    <div id='object1'>margin:<br />10px 20px 30px 40px;</div>
                </td>
            </tr>
        </table>
    </body>
</html>
```

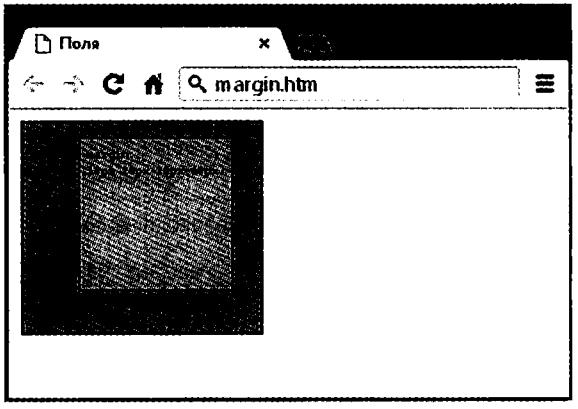



**Рис. 18.13.** Охватывающая таблица расширяется в соответствии с шириной полей

## Применение границ

Уровень границ модели блока похож на уровень полей, за исключением того, что здесь отсутствует поглощение. Это следующий уровень по мере продвижения к центру модели блока. Основными свойствами, используемыми для изменения границ, являются `border`, `border-left`, `border-top`, `border-right` и `border-bottom`. Каждое из них может иметь другие подсвойства, добавляемые в виде суффиксов, например `-color`, `-style` и `-width`.

Четыре способа установки отдельных свойств, которые использовались для свойства `margin`, применимы также и для свойства ширины границы `border-width`, поэтому все следующие правила составлены верно:

```
/* Все границы */
border-width:1px;

/* Верхняя/нижняя и левая/правая */
border-width:1px 5px;

/* Верхняя, левая/правая и нижняя */
border-width:1px 5px 10px;

/* Верхняя, правая, нижняя и левая */
border-width:1px 5px 10px 15px;
```

На рис. 18.14 показано каждое из этих правил, примененное по очереди к группе квадратных элементов. Если смотреть на первый из них, сразу становится понятно, что ширина всех границ равна 1 пикселу. А вот у второго элемента верхняя и нижняя границы имеют ширину 1 пиксел, а его боковые границы имеют ширину по 5 пикселов. У третьего элемента верхняя граница шириной 1 пиксел, его боковые границы — по 5 пикселов, а его нижняя — 10 пикселов. Четвертый элемент изображен с верхней границей шириной 1 пиксел, с правой границей — 5 пикселов, нижней — шириной 10 пикселов и левой — шириной 15 пикселов.

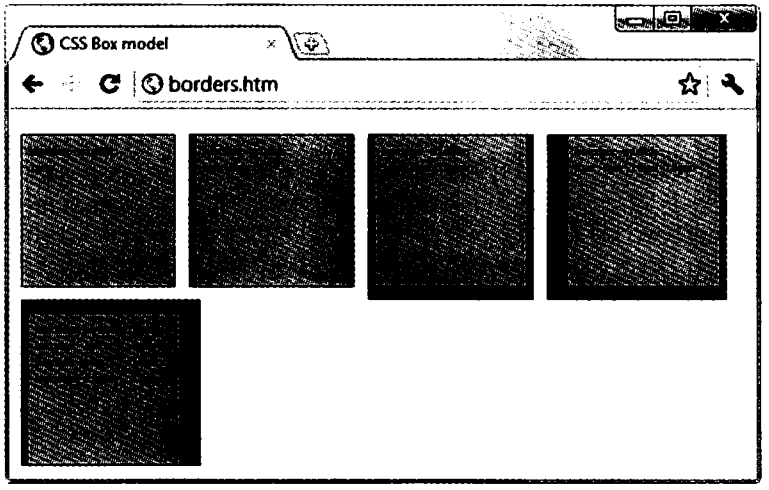



**Рис. 18.14.** Применение правил задания границ в полной и сокращенной записи

Для последнего элемента, расположенного под предыдущими элементами, сокращенная запись правил не использовалась. Вместо этого каждая из его границ задавалась отдельно. Как видите, для получения аналогичного результата потребовалось набрать значительно больше символов.

## Настройка отступов

Самыми глубокими уровнями модели блока (отличающимися от содержимого элемента) являются отступы, применяемые внутри любых границ и (или) полей. Основными свойствами, используемыми для изменения отступов, являются `padding`, `padding-left`, `padding-top`, `padding-right` и `padding-bottom`.

Те четыре способа установки отдельных свойств, которые задействовались для свойств `margin` и `border`, применимы также и для свойства отступа — `padding`, поэтому все следующие правила составлены верно:

```
 * Все отступы */
padding:1px;

 * Верхний/нижний и левый/правый */
padding:1px 2px;

 * Верхний, левый/правый и нижний */
padding:1px 2px 3px;

 * Верхний, правый, нижний и левый */
padding:1px 2px 3px 4px;
```

На рис. 18.15 показаны правила отступов, выделенные в примере 18.7 полужирным шрифтом и примененные к тексту в ячейке таблицы (как определено с помощью правила `display:table-cell;`, которое задает охват `<div>`-элемента

наподобие ячейки таблицы). Размеры ячейки не заданы, поэтому она максимально плотно охватывает текст. Вследствие этого получается отступ, равный 10 пикселам, над внутренним элементом, 20 пикселам — справа, 30 пикселам — снизу и 40 пикселам — слева.

**Пример 18.7.** Применение отступов

```
<!DOCTYPE html>
<html>
    <head>
        <title>Отступы</title>
        <style>
            #object1 {
                border-style:solid;
                border-width:1px;
                background  :orange;
                color       :darkred;
                font-face   :Arial;
                font-size   :12px;
                text-align  :justify;
                display     :table-cell;
                width       :148px;
                padding     :10px 20px 30px 40px; }
        </style>
    </head>
    <body>
        <div id='object1'>To be, or not to be that is
        the question: Whether 'tis Nobler in the mind
        to suffer The Slings and Arrows of outrageous
        Fortune, Or to take Arms against a Sea of
        troubles, And by opposing end them.</div>
    </body>
</html>
```

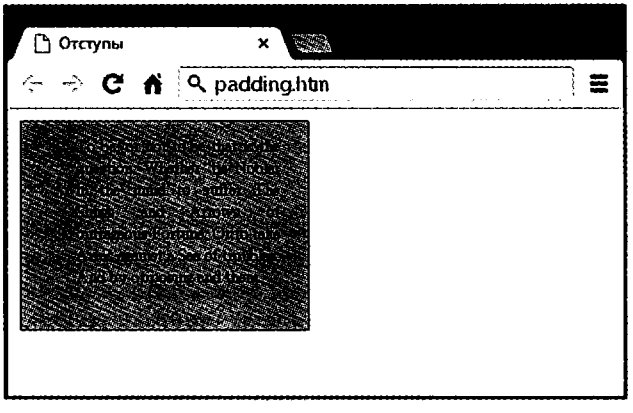



**Рис. 18.15.** Применение к объекту разных значений отступов

## Содержимое объекта

В глубине модели блока, в его центре, находится элемент, стиль которого может быть задан всеми способами, рассмотренными ранее в этой главе, и который может содержать (а зачастую и содержит) еще и подчиненные элементы, у которых, в свою очередь, могут быть свои подчиненные элементы и т. д., у каждого из которых могут быть свои настройки стиля и модели блока.

Теперь, после изучения основ, в следующей главе мы рассмотрим усовершенствованные CSS-таблицы, включая и способы применения таких переходных эффектов, как перемещение и вращение, а также других новых интересных свойств, появившихся в CSS3.

# Проверьте ваши знания

1. Какая инструкция используется для импорта одной таблицы стилей в другую (или в блок `<style>` кода HTML)?
2. Каким HTML-тегом можно воспользоваться для импорта таблицы стилей в документ?
3. Какой атрибут HTML-тега применяется для непосредственной вставки стиля в элемент?
4. В чем разница между идентификатором CSS и классом CSS?
5. Какие символы используются в качестве префиксов в CSS-правилах: а) идентификаторы и б) классы?
6. Каково назначение точки с запятой в CSS-правилах?
7. Как в таблице стилей добавляется комментарий?
8. Какой символ используется CSS для представления «любого элемента»?
9. Как в CSS можно выбрать группу разных элементов и (или) типов элементов?
10. Как можно задать преимущество одного из двух CSS-правил, имеющих одинаковый уровень приоритета?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 18».

# 19 Расширение CSS с помощью CSS3

Первая реализация CSS была разработана в 1996 году, выпущена в 1999 году и к 2001 году была поддержана всеми выпусками браузеров. Стандарт для этой версии, CSS1, был еще раз пересмотрен в 2008 году. Со второй спецификацией, CSS2, разработчики начали работать в 1998 году; ее стандарт был в конечном итоге завершен в 2007 году, а затем еще раз пересмотрен в 2009 году.

В 2001 году началась разработка спецификации CSS3, а ее характеристики были предложены сравнительно недавно, в 2009 году. Процесс разработки, вероятно, еще продолжится до поступления завершающих рекомендаций по CSS3. И даже несмотря на то, что разработка спецификации CSS3 еще не завершена, уже начали поступать предложения на будущее для спецификации CSS4.

В этой главе будут описаны характеристики CSS3, которые уже в целом приняты всеми основными браузерами. Некоторые из этих характеристик предоставляют такие функциональные возможности, которые до этого могли быть предоставлены только с помощью JavaScript.

Я рекомендую использовать CSS3 для реализации динамических свойств везде, где только можно, вместо JavaScript, поскольку весьма вероятно, что поддержка CSS приобретет очень высокий уровень оптимизации (и в силу этого также и очень высокое быстродействие) и, что для вас не менее важно, при этом упростится поддержка ваших продуктов в новых браузерах и их версиях.

## Селекторы атрибутов

В предыдущей главе были подробно рассмотрены различные селекторы атрибутов, применяемые в CSS, которые мы сейчас вкратце повторим. Селекторы используются в CSS для сопоставления с HTML-элементами. Существует десять разных типов селекторов, показанных в табл. 19.1.

**Таблица 19.1.** CSS-селекторы, псевдоклассы и псевдоэлементы

| Тип селектора | Пример |
|---|---|
| Универсальный селектор | * { color:#555; } |
| Селекторы типов | b { color:red; } |
| Селекторы классов | .classname { color:blue; } |

| Тип селектора | Пример |
|---|---|
| Селекторы идентификаторов | #idname { background:cyan; } |
| Селекторы потомков | span em { color:green; } |
| Селекторы дочерних элементов | div > em { background:lime; } |
| Селекторы смежных элементов | i + b { color:gray; } |
| Селекторы атрибутов | a[href='info.htm'] { color:red; } |
| Псевдоклассы | a:hover { font-weight:bold; } |
| Псевдоэлементы | p::first-letter { font-size:300%; } |

Разработчики CSS3 решили, что большинство из этих селекторов работают достаточно хорошо и в представленном на данный момент виде, но три усовершенствования, направленные на упрощение поиска соответствия элементам на основе содержимого их атрибутов, они все же внесли.

# Соответствующие части строк

В CSS2 для поиска соответствия строке `'info.htm'`, находящейся в `href`-атрибуте, можно было использовать такой селектор, как `a[href='info.htm']`, но поиска соответствия только части строки не существовало. Но в CSS3 пошли дальше и определили три новых оператора: `^`, `$` и `*`. Если один из них непосредственно предшествует символу равенства (=), то с помощью этих символов в том порядке, в котором они перечислены, можно искать соответствие в начале, в конце или в любой части строки.

## Оператор ^

Например, следующему селектору будет соответствовать любой `href`-атрибут, чье значение начинается со строки `'http://website'`:

```
a[href^='http://website']
```

Таким образом, ему будет соответствовать следующий элемент:

```
<a href='http://website.com'>
```

А этот элемент соответствовать не будет:

```
<a href='http://mywebsite.com'>
```

## Оператор $

Для поиска соответствия только в конце строки можно использовать следующий селектор, которому будет соответствовать любой `img`-тег, чей `src`-атрибут заканчивается на `'.png'`:

```
img[src$='.png']
```

Например, ему будет соответствовать следующий тег:

```
<img src='photo.png' />
```

А этот тег соответствовать не будет:

```
<img src='snapshot.jpg' />
```

### Оператор *

Для поиска соответствия подстроки, находящейся в любом месте атрибута, можно воспользоваться следующим селектором. Он найдет любые ссылки на странице, имеющие строку 'google' в любом месте ссылки:

```
a[href*='google']
```

Например, ему будет соответствовать эта часть кода HTML:

```
<a href='http://google.com'>
```

а эта часть соответствовать не будет:

```
<a href='http://gmail.com'>
```

## Свойство box-sizing

В модели блока W3C определено, что ширина и высота объекта должна относиться только к размерам содержимого элемента, игнорируя любые отступы или границы. Но некоторые веб-дизайнеры выразили желание указывать размеры, относящиеся ко всему элементу, включая любые отступы и границы.

Чтобы предоставить такое свойство, CSS3 позволяет вам выбрать желаемую модель блока со свойством задания размеров блока — box-sizing. Например, для использования общей ширины и высоты объекта, включая отступы и границы, нужно применять следующее объявление:

```
box-sizing:border-box;
```

Или, чтобы ширина и высота объекта относилась только к содержимому, нужно воспользоваться следующим объявлением (применяемым по умолчанию):

```
box-sizing:content-box;
```

Браузеры на движках WebKit и Mozilla (такие как Safari и Firefox соответственно) требуют для этого объявления использования собственных префиксов (-webkit- и -moz-), о чем подробно рассказывается на веб-сайте http://caniuse.com.

## Создание фона в CSS3

Спецификация CSS3 предоставляет два новых свойства, background-clip и background-origin, которые могут использоваться для указания, где должен фон начинаться внутри элемента и как усекать фон так, чтобы он не появлялся в тех частях модели блока, где он нежелателен.

Для выполнения этих задач оба свойства поддерживают следующие значения:

- border-box — относится к внешнему краю границы;
- padding-box — относится к внешнему краю области отступа;
- content-box — относится к внешнему краю области содержимого.

## Свойство background-clip

Это свойство определяет, должен ли фон игнорироваться (усекаться) если он появляется либо внутри границы, либо в области отступов элемента. Например, следующее объявление определяет, что фон может отображаться во всех частях элемента, вплоть до внешнего края границы:

```
background-clip:border-box;
```

Если не нужно, чтобы фон появлялся в области границы элемента, его можно ограничить только той частью элемента, которая находится внутри и заканчивается внешним краем его области отступов, например:

```
background-clip:padding-box;
```

Или же можно ограничить фон, чтобы он отображался только внутри области содержимого элемента, воспользовавшись следующим объявлением:

```
background-clip:content-box;
```

На рис. 19.1 показаны три ряда элементов, отображаемых в веб-браузере Safari: в первом ряду для свойства `background-clip` используется значение `border-box`, во втором применяется значение `padding-box`, а в третьем используется значение `content-box`.

В первом ряду внутреннему блоку (файлу изображения, загруженному в левую верхнюю часть элемента с отключенным повторением) разрешается отображаться в элементе везде. Можно также совершенно отчетливо видеть, что он отображается в области границы первого блока, поскольку стиль границы указан пунктирным.

Во втором ряду в области границы не отображаются ни фоновое изображение, ни фоновое затенение, поскольку они были усечены по области отступов с помощью установки для свойства `background-clip` значения `padding-box`.

И наконец, в третьем ряду и фоновое затенение, и фоновое изображение были усечены для отображения только внутри области содержимого каждого элемента (показанного внутри светлого, ограниченного пунктирной линией блока), путем использования для свойства `background-clip` значения `content-box`.

## Свойство background-origin

С помощью этого свойства можно также указать, где должно располагаться фоновое изображение, указав для этого, где должен начинаться левый верхний угол данного изображения. Например, следующее объявления указывает, что начало фонового изображения должно быть в левом верхнем углу внешнего края границы:

```
background-origin:border-box;
```

Чтобы установить начало изображения в левый верхний внешний угол области отступов, нужно воспользоваться следующим объявлением:

```
background-origin:padding-box;
```

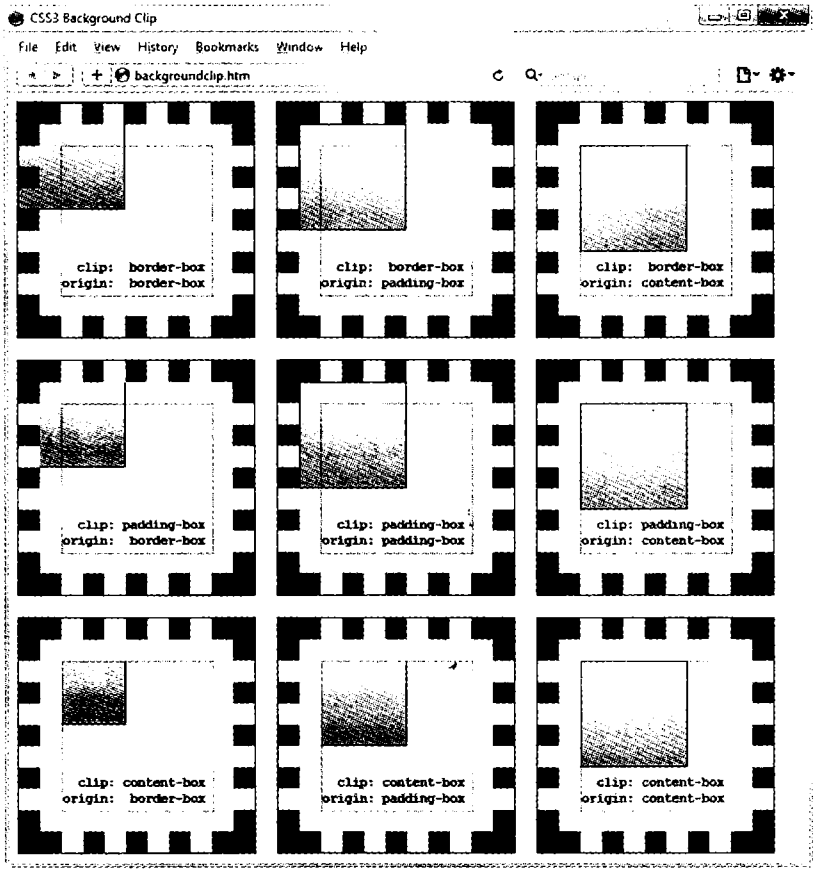



**Рис. 19.1.** Разные способы сочетания свойств фона CSS3

И чтобы установить начало изображения в левый верхний угол области внутреннего содержимого элемента, нужно воспользоваться следующим объявлением:

```
background-origin:content-box;
```

Посмотрите еще раз на рис. 19.1, в каждом ряду для первого блока используется свойство `background-origin` со значением `border-box`, для второго блока это же свойство применяется со значением `padding-box`, а для третьего — со значением `content-box`. Следовательно, в каждом ряду меньший по размеру внутренний блок отображается для первого блока в левом верхнем углу границы, для второго он отображается в левом верхнем углу области отступов, а для третьего — в левом верхнем углу содержимого.

Единственное отличие рядов, которое стоит отметить в отношении начала внутреннего блока на рис. 19.1 состоит в том, что во втором и третьем рядах внутренний блок усекается соответственно областями отступов и содержимого, и поэтому та часть блока, которая находится за пределами этих областей, не отображается.

## Свойство background-size

Точно так же, как это делалось для указания ширины и высоты изображения при использовании тега <img />, в последних версиях всех браузеров можно сделать то же самое для изображений фона.

Это свойство можно применить следующим образом (здесь *ww* — ширина, а *hh* высота):

```
background-size:wwpx hhpx;
```

При необходимости можно использовать только один аргумент, и для обоих размеров будет установлено указанное значение. Кроме того, если применить данное свойство к блочному элементу, например к  (но не к такому встроенному элементу, как ), можно указать ширину и (или) высоту в процентном отношении, а не в виде фиксированного значения.

### Использование значения auto

Если нужно масштабировать только один размер фонового изображения, чтобы при этом автоматически масштабировался и другой его размер для соблюдения прежних пропорций, для другого размера можно воспользоваться значением auto:

```
background-size:100px auto;
```

Этим объявлением устанавливается ширина, равная 100 пикселам, и высота, равная значению, пропорциональному увеличению или уменьшению ширины.

Разные браузеры могут требовать различных версий имен свойства background, поэтому при их использовании обратитесь к веб-сайту http://caniuse.com, чтобы убедиться, что вы применяете все версии, которые требуются тем браузерам, на работу с которыми вы рассчитываете.

## Использование нескольких фонов

Теперь, используя CSS3, вы можете прикрепить к элементу несколько фонов, каждый из который может применять ранее рассмотренные свойства фона CSS3. Соответствующий пример показан на рис. 19.2. На этом рисунке в качестве фона были назначены восемь изображений, которые используются для создания четырех углов и четырех кромок границы сертификата.

Для вывода нескольких фоновых изображений с помощью одного CSS-объявления нужно их разделить занятыми. В примере 19.1 показан код HTML и CSS, использованный для создания фона рис. 19.2.

**Пример 19.1.** Использование в фоне сразу нескольких изображений

```
<!DOCTYPE html>
<html>
    <head>
        <title>CSS3: пример нескольких фоновых изображений</title>
        <style>
            .border {
                font-family:'Times New Roman'
```

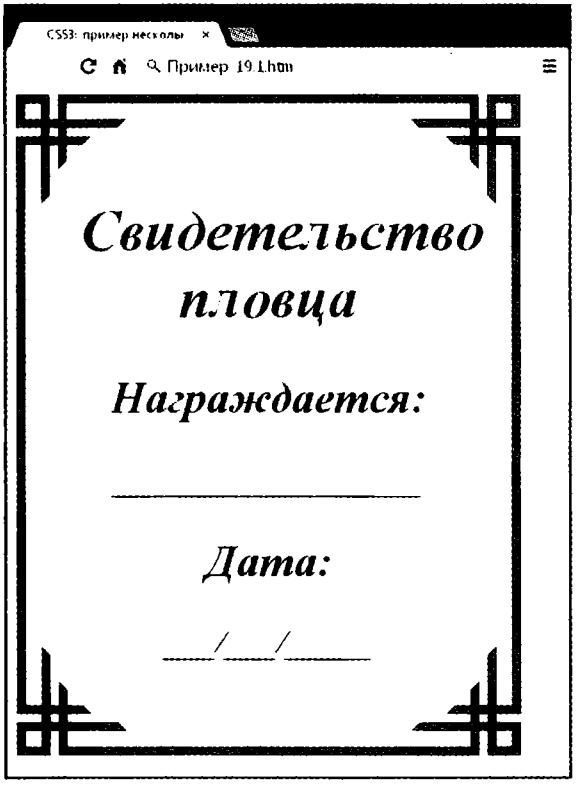



**Рис. 19.2.** Фон, созданный с помощью нескольких изображений

```
            font-style :italic;
            font-size  :170%;
            text-align :center;
            padding    :60px;
            width      :350px;
            height     :500px;
            background :url('b1.gif') top    left  no-repeat,
                        url('b2.gif') top    right no-repeat,
                        url('b3.gif') bottom left  no-repeat,
                        url('b4.gif') bottom right no-repeat,
                        url('ba.gif') top          repeat-x,
                        url('bb.gif') left         repeat-y,
                        url('bc.gif') right        repeat-y,
                        url('bd.gif') bottom       repeat-x }
    </style>
</head>
<body>
    <div class='border'>
        <h1>Свидетельство пловца</h1>
        <h2>Награждается:</h2>
        <h3>_____</h3>
```

```
            <h2>Дата:</h2>
            <h3>___/___/_____</h3>
        </div>
    </body>
</html>
```

Первые четыре строки объявления фона в блоке CSS расставляют угловые изображения по четырем углам элемента, а последние четыре строки помещают изображения кромок, которые обрабатываются в последнюю очередь, потому что порядок приоритетности для фоновых изображений имеет направление сверху вниз. Иными словами, когда они накладываются друг на друга, дополнительные фоновые изображения будут появляться позади уже размещенных изображений. Если бы GIF-изображения были перечислены в обратном порядке, то повторяющиеся изображения кромок отображались бы поверх углов, что было бы неправильно.

Используя этот код CSS, можно изменять размеры содержащего фон элемента по любым направлениям, и граница будет всегда правильно изменяться в размерах, чтобы поместиться в элементе, что намного проще, чем применение таблиц или нескольких элементов для получения такого же эффекта.

# Границы CSS3

CSS3 так же придает намного больше гибкости способам возможного представления границ, разрешая независимо менять цвета всех четырех кромок, отображать изображения для кромок и углов, предоставлять значения радиусов для придания границам закругленных углов и помещать прямоугольные тени под элементами.

## Свойство border-color

Применять цвета к границе можно двумя способами. Начнем с того, что свойству можно передать всего один цвет:

```
border-color:#888;
```

Это объявление устанавливает светло-серый цвет для всех границ элемента. Можно также установить цвета границ по отдельности (здесь цвета границы устанавливаются в различные градации серого):

```
border-top-color   :#000;
border-left-color  :#444;
border-right-color :#888;
border-bottom-color:#ccc;
```

Кроме того, можно назначить все цвета по отдельности в одном объявлении:

```
border-color:#f00 #0f0 #880 #00f;
```

Это объявление устанавливает цвет верхней границы в #f00, правой границы в #0f0, нижней границы в #880 и левой границы в #00f (красный, зеленый,

оранжевый и синий соответственно). Можно также использовать в качестве аргументов названия цветов, что уже рассматривалось в предыдущей главе.

## Свойство border-radius

До появления CSS3 способные веб-разработчики придумали множество различных настроек с целью получения закругленных границ, используя, как правило, теги <table> или .

Но теперь добавление закругленных границ к элементу дается по-настоящему просто, и, как показано на рис. 19.3, работает в последних версиях всех основных браузеров. На этом рисунке граница толщиной 10 пикселов выведена различными способами. Код HTML для получения такого результата показан в примере 19.2.

**Пример 19.2.** Свойство border-radius

```
<DOCTYPE html>
<html>
  <head>
    <title>CSS3: примеры радиусов границ</title>
    <style>
      .box {
        margin-bottom:10px;
        font-family  :'Courier New', monospace;
        font-size    :12pt;
        text-align   :center;
        padding      :10px;
        width        :380px;
        height       :75px;
        border       :10px solid #006; }
      .b1 {
        -moz-border-radius   :40px;
        -webkit-border-radius:40px;
        border-radius        :40px; }
      .b2 {
        -moz-border-radius   :40px 40px 20px 20px;
        -webkit-border-radius:40px 40px 20px 20px;
        border-radius        :40px 40px 20px 20px; }
      .b3 {
        -moz-border-radius-topleft        :20px;
        -moz-border-radius-topright       :40px;
        -moz-border-radius-bottomleft     :60px;
        -moz-border-radius-bottomright    :80px;
        -webkit-border-top-left-radius    :20px;
        -webkit-border-top-right-radius   :40px;
        -webkit-border-bottom-left-radius :60px;
        -webkit-border-bottom-right-radius:80px;
        border-top-left-radius            :20px;
        border-top-right-radius           :40px;
        border-bottom-left-radius         :60px;
        border-bottom-right-radius        :80px; }
      .b4 {
```

```
            -moz-border-radius-topleft        :40px 20px;
            -moz-border-radius-topright       :40px 20px;
            -moz-border-radius-bottomleft     :20px 40px;
            -moz-border-radius-bottomright    :20px 40px;
            -webkit-border-top-left-radius    :40px 20px;
            -webkit-border-top-right-radius   :40px 20px;
            -webkit-border-bottom-left-radius :20px 40px;
            -webkit-border-bottom-right-radius:20px 40px;
            border-top-left-radius            :40px 20px;
            border-top-right-radius           :40px 20px;
            border-bottom-left-radius         :20px 40px;
            border-bottom-right-radius        :20px 40px; }
        </style>
    </head>
    <body>
        <div class='box b1'>
            border-radius:40px;
        </div>

        <div class='box b2'>
            border-radius:40px 40px 20px 20px;
        </div>

        <div class='box b3'>
            border-top-left-radius    :20px;<br/>
            border-top-right-radius   :40px;<br />
            border-bottom-left-radius :60px;<br />
            border-bottom-right-radius:80px;
        </div>

        <div class='box b4'>
            border-top-left-radius    :40px 20px;<br />
            border-top-right-radius   :40px 20px;<br />
            border-bottom-left-radius :20px 40px;<br />
            border-bottom-right-radius:20px 40px;
        </div>
    </body>
</html>
```

Так, например, для создания закругленной границы радиусом 20 пикселов можно просто воспользоваться следующим объявлением:

```
border-radius:20px;
```

Хотя многие браузеры (включая Internet Explorer) со свойствами радиусов границы будут работать хорошо, некоторые текущие (и многие более старые) версии основных браузеров используют разные имена свойств, и если нужна поддержка всех этих браузеров, придется также включить для них соответствующие префиксы, характерные для тех или иных браузеров, такие как -moz- и -webkit-. Чтобы обеспечить работу предыдущего примера во всех браузерах, я добавил в код все требуемые префиксы.

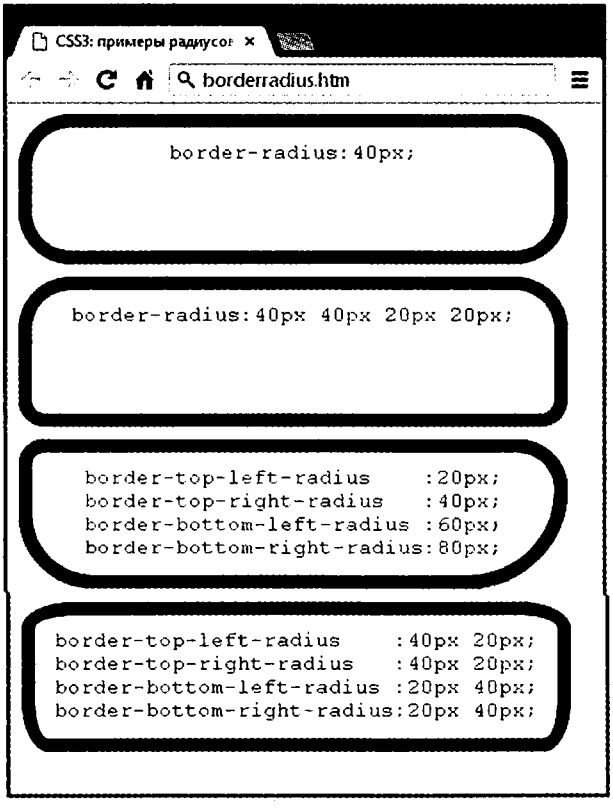



**Рис. 19.3.** Смешивания и сопоставления различных свойств радиусов границы

Можно указать отдельные радиусы для каждого из четырех углов (по часовой стрелке, начиная с левого верхнего угла):

```
border-radius:10px 20px 30px 40px;
```

При необходимости можете также указать радиус отдельно для каждого угла элемента:

```
border-top-left-radius    :20px;
border-top-right-radius   :40px;
border-bottom-left-radius :60px;
border-bottom-right-radius:80px;
```

И при ссылке на отдельные углы можно предоставить два аргумента, выбирая тем самым разные вертикальные и горизонтальные радиусы (в результате чего получаются более интересные и тонко настраиваемые границы):

```
border-top-left-radius    :40px 20px;
border-top-right-radius   :40px 20px;
border-bottom-left-radius :20px 40px;
border-bottom-right-radius:20px 40px;
```

Первым аргументом задается горизонтальный, а вторым вертикальный радиус.

# Прямоугольные тени

Для применения прямоугольной тени нужно указать горизонтальное и вертикальное смещение от объекта и величину размытости, добавляемой к тени, а также используемый для тени цвет:

```
box-shadow:15px 15px 10px #888;
```

Два значения по 15px задают (по порядку) горизонтальное вертикальное смещение от элемента, и эти значения могут быть отрицательными, нулевыми или положительными. Значение 10px указывает величину размытости (примерно четверть сантиметра на обычном дисплее), где меньшие значения приводят к меньшей размытости, а #888 — это цвет тени, который может быть любым допустимым цветом (см. раздел «Цвета CSS» главы 18). Результат этого объявления можно увидеть на рис. 19.4.

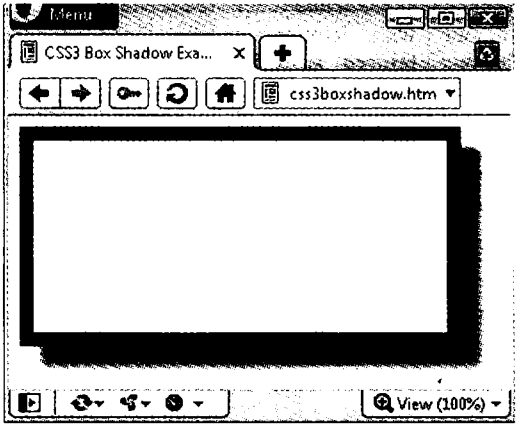



**Рис. 19.4.** Прямоугольная тень, отображенная под элементом element

При использовании этого свойства в браузерах, основанных на движках WebKit и Mozilla, для него нужно применять префиксы -webkit- и -moz-.

# Выход элемента за пределы размеров

В CSS2 можно определить, что делать, когда один элемент слишком велик, чтобы полностью поместиться в другом, родительском по отношению к нему элементу, путем указания для свойства `overflow` значения `hidden`, `visible`, `scroll` или `auto`. Но теперь в CSS3 можно также отдельно применить эти значения к горизонтальному или вертикальному направлению, как в следующих примерах объявлений:

```
overflow-x:hidden;
overflow-x:visible;
overflow-y:auto;
overflow-y:scroll;
```

# Разметка с использованием нескольких колонок

Использование нескольких колонок уже давно является у веб-разработчиков наиболее востребованным свойством, и в CSS3 оно наконец-то было реализовано, а Internet Explorer 10 стал последним из основных браузеров, принявшим это свойство.

Теперь перетекание текста по нескольким колонкам задать не сложнее, чем указать количество колонок, а затем (дополнительно) выбрать разрядку между ними и тип разделительной линии (если она нужна). На рис. 19.5 показан результат выполнения кода примера 19.3.

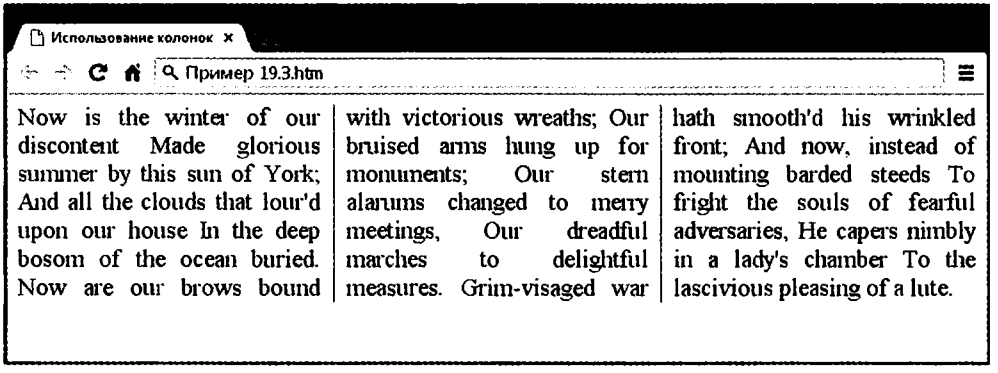



**Рис. 19.5.** Перетекание текста по нескольким колонкам

**Пример 19.3.** Использование CSS для создания нескольких колонок

```
<DOCTYPE html>
<html>
    <head>
        <title>Использование колонок</title>
        <style>
            .columns {
                text-align          :justify;
                font-size           :16pt;
                -moz-column-count   :3;
                -moz-column-gap     :1em;
                -moz-column-rule    :1px solid black;
                -webkit-column-count:3;
                -webkit-column-gap  :1em;
                -webkit-column-rule :1px solid black;
                column-count        :3;
                column-gap          :1em;
                column-rule         :1px solid black; }
        </style>
    </head>
    <body>
        <div class='columns'>
```

```
            Now is the winter of our discontent
            Made glorious summer by this sun of York;
            And all the clouds that lour'd upon our house
            In the deep bosom of the ocean buried.
            Now are our brows bound with victorious wreaths;
            Our bruised arms hung up for monuments;
            Our stern alarums changed to merry meetings.
            Our dreadful marches to delightful measures.
            Grim-visaged war hath smooth'd his wrinkled front;
            And now, instead of mounting barded steeds
            To fright the souls of fearful adversaries,
            He capers nimbly in a lady's chamber
            To the lascivious pleasing of a lute.
        </div>
    </body>
</html>
```

Внутри класса `.columns` первые две строки просто предписывают браузеру выровнять текст по правому краю и установить для него размер шрифта `16pt`. Эти объявления для нескольких колонок не нужны, но они улучшают отображение текста. В остальных строках элемент настраивается таким образом, чтобы внутри него текст перетекал по трем колонкам с разрывом между колонками, равным `1em`, и с границей в 1 пиксел, проходящей по середине каждого разрыва.

В этом примере браузеры на основе движков Mozilla и WebKit требуют для объявлений соответствующих этим браузерам префиксов.

# Цвета и непрозрачность

Способы определения цветов в CSS3 существенно расширились: теперь вы можете также использовать CSS-функции для применения цветов в широко распространенных форматах RGB (красный, зеленый, синий), RGBA (красный, зеленый, синий, альфа), HSL (тон, насыщенность, яркость) и HSLA (тон, насыщенность, яркость, альфа). Значение *альфа* определяет прозрачность цвета, позволяющую увидеть элементы, расположенные ниже.

## Цвета HSL

Для определения цвета с помощью функции `hsl` сначала нужно выбрать из цветового круга значение для тона в диапазоне от 0 до 359. Любой более высокий номер цвета просто возвращается по кругу к началу, таким образом значение 0 соответствует красному цвету, точно так же, как и значения 360 и 720.

В цветовом круге основные цвета — красный, зеленый и синий — поделены на 120 градусов, поэтому чистый красный цвет соответствует значению 0, зеленый — значению 120, а синий — значению 240. Числа между этими значениями представляют собой оттенки, содержащие различные пропорции основных цветов с обеих сторон.

Затем нужен уровень насыщенности, значение которого лежит в диапазоне от 0 до 100 %. Он определяет то, насколько сильно цвет будет размыт или ярок. Значения насыщенности начинаются в центре колеса со светло-серого цвета (насыщенность равна 0 %), а затем по направлению к краю (где насыщенность равна 100 %) становится все более отчетливой.

Вам остается только решить, насколько ярким нужен цвет, для этого нужно выбрать значение яркости в диапазоне от 0 до 100 %. Значение 50 % для яркости дает наполненный, яркий цвет, а уменьшение значения (вниз, вплоть до минимума в 0 %) делает его темнее, до тех пор пока цвет не станет черным. Увеличение значения (вверх, вплоть до максимума в 100 %) делает цвет светлее, до тех пор пока он не станет белым. Вы можете визуально представить это подмешиванием в цвет либо черного, либо белого цвета.

Так, например, для выбора полностью насыщенного желтого цвета со стандартной яркостью нужно воспользоваться следующим объявлением:

```
color:hsl(60, 100%, 50%);
```

Или для выбора темно-синего цвета можно воспользоваться следующим объявлением:

```
color:hsl(240, 100%, 40%);
```

Этим также можно воспользоваться (как и всеми остальными CSS-функциями, связанными с заданием цвета) с любым свойством, ожидающим применения цветовых настроек, например с `background-color` и т. д.

## Цвета HSLA

Для предоставления еще большего контроля над способом цветообразования, можно воспользоваться функцией `hsla`, предоставив ей четвертый (или альфа) уровень настройки цвета, значение которого задается числом с плавающей точкой в диапазоне от 0 до 1. Значение 0 определяет, что цвет является полностью прозрачным, а число 1 задает полную непрозрачность цвета.

Выбрать желтый цвет с полной насыщенностью, стандартной яркостью и непрозрачностью 30 % можно с помощью следующего объявления:

```
color:hsla(60, 100%, 50%, 0.3);
```

Или же для выбора полностью насыщенного, но чуть более светлого синего цвета с 82%-ной непрозрачностью можно воспользоваться следующим объявлением:

```
color:hsla(240, 100%, 60%, 0.82);
```

## Цвета RGB

Наверное, вам более знакома система выбора цвета RGB, поскольку она похожа на использование форматов цвета *#nnnnnn* и *#nnn*. Например, для задания желтого

цвета можно воспользоваться любым из следующих объявлений (первое из них поддерживает шестнадцать миллионов цветов, а второе — четыре тысячи):

```
color:#ffff00;
color:#ff0;
```

Для получения такого же результата можно также воспользоваться CSS-функцией rgb, но при этом нужно применять не шестнадцатеричные, а десятичные числа (где десятичное число 255 соответствует шестнадцатеричному числу ff):

```
color:rgb(255, 255, 0);
```

Но еще лучше вам будет даже не задумываться больше о том, чему соответствуют значения до 256, поскольку можно указать процентные значения:

```
color:rgb(100%, 100%, 0);
```

Фактически теперь вы можете с большой точностью определить настройки для нужного цвета, просто думая о его основных цветовых составляющих. Например, сочетание зеленого и синего дает фиолетовый цвет, поэтому для задания цвета, близкого к фиолетовому, но с синей составляющей, преобладающей над зеленой, можно составить первое предположение, что для него нужно определить 0 % красного, 40 % зеленого и 60 % синего цвета и попробовать воспользоваться следующим объявлением:

```
color:rgb(0%, 60%, 40%);
```

## Цвета RGBA

Как и функция hsla, функция rgba поддерживает четвертый (альфа) аргумент, позволяющий, к примеру, с помощью следующего объявления применить к прежнему фиолетовому цвету 40%-ную непрозрачность:

```
color:rgba(0%, 60%, 40%, 0.4);
```

## Свойство opacity

Свойство opacity предоставляет такое же альфа-управление, что и функции hsla и rgba, но позволяет изменять непрозрачность объекта (или прозрачность, если это вам больше нравится) отдельно от его цвета.

Для использования этого цвета нужно применить к элементу следующее объявление (которое в данном примере устанавливает непрозрачность, равную 25 %, или прозрачность, равную 75 %):

```
opacity:0.25;
```

Для браузеров на основе движков WebKit и Mozilla для этого свойства требуются соответствующие этим браузерам префиксы. Кроме того, для обратной совместимости с выпусками Internet Explorer, предшествующими версии 9, нужно добавить следующее объявление (в котором значение непрозрачности умножено на 100):

```
filter:alpha(opacity='25');
```

# Эффекты, применяемые к тексту

Теперь с помощью CSS3 к тексту могут применяться новые эффекты, включая тени текста, наложение, применяемое к тексту, и перенос слов.

## Свойство text-shadow

Это свойство аналогично свойству `box-shadow` и получает такой же набор аргументов: горизонтальное и вертикальное смещение, величину размытости и используемый цвет. Например, следующее объявление задает смещение тени на 3 пиксела как в горизонтальном, так и в вертикальном направлении и отображает тень темно-серым цветом с размытостью 4 пиксела:

```
text-shadow:3px 3px 4px #444;
```

Результат применения этого объявления выглядит, как показано на рис. 19.6. Это объявление работает в последних версиях всех основных браузеров (кроме Internet Explorer 9 или ниже).


**Рис. 19.6.** Применение тени к тексту

## Свойство text-overflow

При использовании любого из CSS-свойств `overflow` со значением, равным `hidden`, можно также воспользоваться свойством `text-overflow` для помещения многоточия сразу же после некоторого текста, подвергшегося усечению:

```
text-overflow:ellipsis;
```

Если это свойство не использовать, то когда текст «To be, or not to be. That is the question.» усекается, результат выглядит, как показано на рис. 19.7, но с применением объявления результат выглядит, как изображено на рис. 19.8.

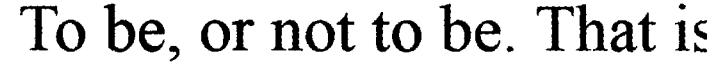



**Рис. 19.7.** Текст автоматически усекается

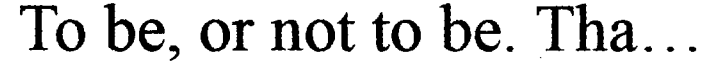



**Рис. 19.8.** Вместо простого усечения, текст завершается многоточием

Чтобы это работало, требуется выполнить три условия.

- У элемента должно быть свойство `overflow`, настроенное на невидимость, например `overflow:hidden`.
- Элемент должен иметь свойство `white-space:nowrap`, настраивающее на ограничение текста.
- Ширина элемента должна быть меньше, чем усекаемый текст.

## Свойство word-wrap

Когда используется по-настоящему длинное слово, шире того элемента, в котором оно содержится, оно либо выйдет за пределы, либо будет усечено. Но в качестве альтернативного варианта свойству `text-overflow` и усечению текста можно воспользоваться свойством `word-wrap` со значением `break-word` для переноса длинных строк:

```
word-wrap:break-word;
```

Например, на рис. 19.9 показано, что слово Honorificabilitudinitatibus шире, чем содержащее его поле (чей правый край представлен в виде сплошной вертикальной черты между буквами t и a), и, поскольку свойство `overflow` применено не было, слово выходит за границу своего контейнера.


**Рис. 19.9.** Слово имеет слишком большую ширину для своего контейнера и поэтому выходит за его границу

Но на рис. 19.10 свойству `word-wrap` элемента было присвоено значение `break-word` и поэтому слово аккуратно перенесено на следующую строку.

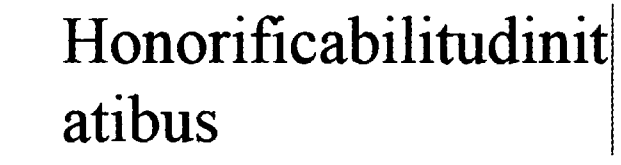



**Рис. 19.10.** Теперь слово переносится по достижении правого края

# Веб-шрифты

Применение веб-шрифтов CSS3 существенно повысило возможности оформления текста, доступные веб-дизайнерам, позволяя загружать шрифты из Интернета, а не только со своего пользовательского компьютера, и отображать их по всей Всемирной сети. Для достижения этого результата нужно объявить веб-шрифт, используя свойство `@font-face`:

```
@font-face
{
  font-family:FontName;
  src:url('FontName.otf');
}
```

Функция `url` требует значение, содержащее путь или URL-адрес шрифта. На большинстве браузеров можно использовать либо шрифты TrueType (TTF), либо шрифты OpenType (OTF), но Internet Explorer ограничивает вас применением шрифтов TrueType, преобразованных в шрифты EOT (EOT).

Чтобы сообщить браузеру тип шрифта, можно воспользоваться функцией `format`, как в следующем примере (для шрифтов OpenType):

```
@font-face
{
  font-family:FontName;
  src:url('FontName.otf') format('opentype');
}
```

или в этом примере (для шрифтов TrueType):

```
@font-face
{
  font-family:FontName;
  src:url('FontName.ttf') format('truetype');
}
```

Но поскольку Internet Explorer принимает только EOT-шрифты, он игнорирует объявления `@font-face`, содержащие функцию `format`.

## Веб-шрифты Google

Одним из лучших способов использования веб-шрифтов является их бесплатная загрузка с серверов Google. Дополнительную информацию по данному вопросу можно найти на сайте веб-шрифтов Google (http://google.com/webfonts; рис. 19.11), где можно получить доступ более чем к 500 семействам шрифтов.

Чтобы показать, насколько легко можно использовать один из этих шрифтов, в следующей строке для загрузки такого шрифта применяется HTML-тег `<link>`:

```
<link href='http://fonts.googleapis.com/css?family=Lobster' />
```

Затем, чтобы воспользоваться таким шрифтом, его нужно просто применить в CSS-объявлении:

```
h1 { font-family:'Lobster', arial, serif; }
```

# Трансформации

Используя трансформации, можно наклонять, вращать, растягивать и сжимать элементы в любом из трех измерений (да, 3D поддерживается, но пока только в браузерах, работающих на движке WebKit). Это упрощает создание впечатляющих эффек-

тов путем выхода за пределы однообразных макетов на основе -контейнеров и других элементов, поскольку теперь они могут быть показаны под различными углами и в различных формах.

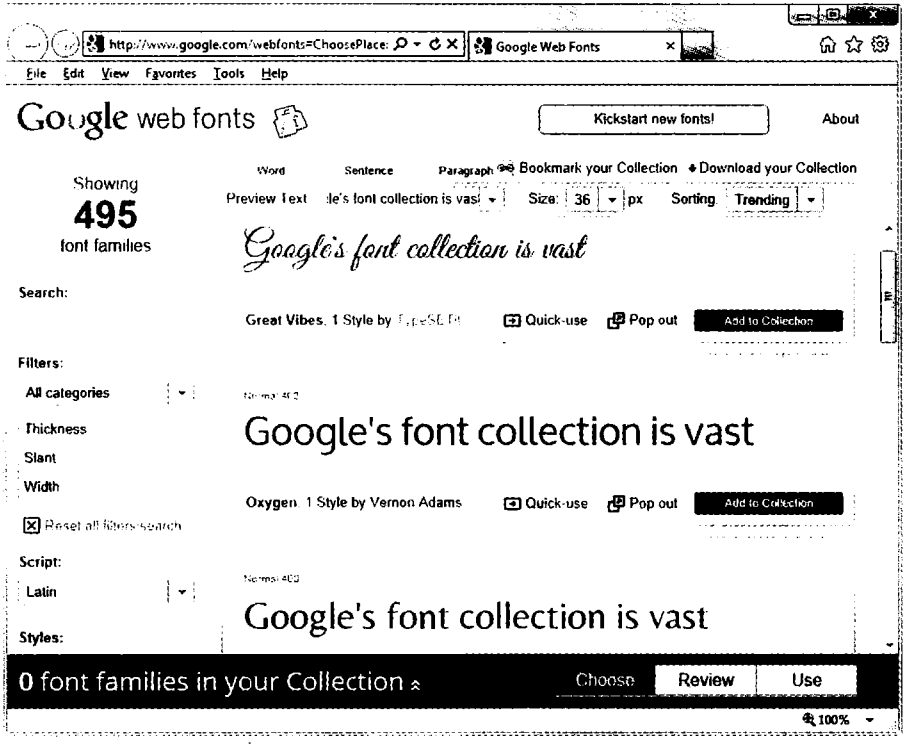



**Рис. 19.11.** Включить веб-шрифты Google не составляет труда

Для выполнения трансформаций нужно воспользоваться свойством `transform` (у которого, к сожалению, должны быть соответствующие префиксы для использования в браузерах Mozilla, WebKit, Opera и Microsoft; по этому вопросу снова следует обратиться на веб-сайт http://caniuse.com).

К свойству `transform` можно применять множество значений, начиная со значения `none`, которое переключает объект в состояние, не допускающее трансформаций:

```
transform:none;
```

Свойство `transform` можно дополнить одной или несколькими из следующих разнообразных функций:

- `matrix` – трансформирует объект, применяя к нему матрицу значений;
- `translate` – перемещает исходную точку элемента;
- `scale` – масштабирует объект;
- `rotate` – вращает объект;
- `skew` – наклоняет объект.

Существуют также отдельные версии многих из этих функций, например `translateX`, `scaleY` и т. д.

Браузеры на основе движка WebKit (например, Apple Safari, iOS, Google Chrome и Android) поддерживают также 3D-трансформации. Но я не буду их рассматривать, потому что Internet Explorer, Opera и браузеры на основе движка Mozilla пока их не поддерживают, но есть надежда, что в ближайшее время подтянутся и эти браузеры.

Так, например, чтобы повернуть элемент по часовой стрелке на 45°, можно применить к нему следующее объявление:

```
transform:rotate(45deg);
```

В то же время вы можете увеличить объект, как это делается с помощью следующего объявления, приводящего к увеличению его ширины в полтора, а высоты в два раза с последующим поворотом:

```
transform:scale(1.5, 2) rotate(45deg);
```

На рис. 19.12 показан объект до и после применения трансформации.

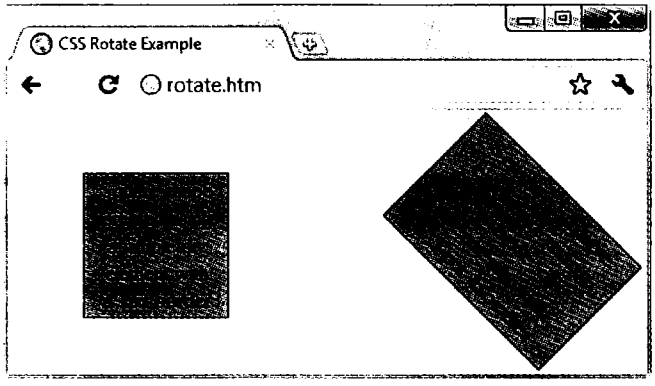



**Рис. 19.12.** Объект до и после трансформации

# Переходы

В последних версиях основных браузеров (включая Internet Explorer 10, не ниже) появилось новое динамическое свойство, называемое *переходами*. Переходы определяют эффект анимации, который нужно применить при трансформации элемента, и браузер автоматически позаботится за вас обо всех промежуточных кадрах.

Для настройки перехода можно предоставить четыре свойства:

```
transition-property       :свойство;
transition-duration       :время;
transition-delay          :время;
transition-timing-function:тип;
```

Свойства нужно предварять соответствующими префиксами браузеров, работающих на движках Mozilla, WebKit, Opera и Microsoft.

## Свойства, применяемые к переходам

У переходов есть такие свойства, как `height`, `border-color` и т. д. При указании свойств преследуется цель изменения CSS-свойства по имени `transition-property` (здесь слово property, «свойства», используемое разными инструментами, имеет разные значения). Можно включить в объявление сразу несколько свойств, разделяя их запятыми:

```
transition-property:width, height, opacity;
```

Или, если вам нужно абсолютно все, относящееся к элементу, подвергаемому переходу (включая цвета), используется значение `all`:

```
transition-property:all;
```

## Продолжительность перехода

Свойство `transition-duration` требует значения от нуля и более секунд. Следующее объявление задает завершение перехода через 1,25 секунды:

```
transition-duration:1.25s;
```

## Задержка перехода

Если свойству `transition-delay` дается значение более нуля секунд (то есть больше значения по умолчанию), происходит задержка между исходным отображением элемента и началом его перехода. Следующее объявление задает начало перехода после задержки в 0,1 секунды:

```
transition-delay:0.1s;
```

Если свойству `transition-delay` дается значение меньше нуля секунд (иными словами, отрицательное значение), переход будет выполнен в момент изменения свойства, но проявится таким образом, будто оно началось с указанным смещением по времени, то есть на каком-то своем промежуточном цикле.

## Задание скорости перехода

Свойству `transition-timing-function` требуется присвоить одно из следующих значений.

- `ease` — медленное начало, ускорение и медленное завершение.
- `linear` — переход с постоянной скоростью.
- `ease-in` — медленное начало, а затем быстрый переход до самого завершения.
- `ease-out` — быстрое начало, сохранение высокой скорости почти до завершения и медленное завершение.
- `ease-in-out` — медленное начало, быстрый переход, затем медленное завершение.

Использование любого из этих значений со словом ease обеспечивает исключительную плавность и естественность перехода в отличие от линейного (linear) перехода, который выглядит как-то более механическим. И если этих изменений вам не вполне достаточно, вы можете также создать свой собственный переход, используя функцию cubic-bezier.

Например, следующие объявления применялись для создания пяти предыдущих типов переходов, и они показывают, как просто можно создавать свои собственные переходы:

```
transition-timing-function:cubic-bezier(0.25, 0.1, 0.25, 1);
transition-timing-function:cubic-bezier(0,    0,   1,    1);
transition-timing-function:cubic-bezier(0.42, 0,   1,    1);
transition-timing-function:cubic-bezier(0,    0,   0.58, 1);
transition-timing-function:cubic-bezier(0.42, 0,   0.58, 1);
```

## Сокращенный синтаксис

Возможно, проще будет воспользоваться сокращенной версией этого свойства и включить все значения в одно объявление, такое как показано далее, которым задается переход всех свойств в линейном режиме за период 0,3 секунды, после начальной (необязательной) задержки в 0,2 секунды:

```
transition:all .3s linear .2s;
```

Это избавит вас от хлопот, связанных с вводом многих очень похожих друг на друга объявлений, особенно если вы поддерживаете префиксы всех основных браузеров.

В примере 19.4 показано, как можно сразу воспользоваться и переходом, и трансформацией. С помощью CSS создается квадратный, оранжевый элемент, с неким текстом внутри, а псевдокласс hover указывает на то, что при проходе над этим объектом указателя мыши он должен повернуться на 180° и изменить свой цвет с оранжевого на желтый (рис. 19.13).

**Пример 19.4.** Эффект перемещения, связанный с применением псевдокласса hover

```
<DOCTYPE html>
<html>
    <head>
        <title>Перемещение при проходе мыши </title>
        <style>
            #square {
                position          :absolute;
                top               :50px;
                left              :50px;
                width             :100px;
                height            :100px;
                padding           :2px;
                text-align        :center;
                border-width      :1px;
                border-style      :solid;
```

```
            background          :orange;
            transition          :all .8s ease-in-out;
            -moz-transition     :all .8s ease-in-out;
            -webkit-transition:all .8s ease-in-out;
            -o-transition       :all .8s ease-in-out;
            -ms-transition      :all .8s ease-in-out; }
        #square:hover {
            background          :yellow;
            -moz-transform      :rotate(180deg);
            -webkit-transform :rotate(180deg);
            -o-transform        :rotate(180deg);
            -ms-transform       :rotate(180deg);
            transform           :rotate(180deg); }
    </style>
</head>
<body>
    <div id='square'>
        Square shape<br />
        created using<br />
        a simple div<br />
        element with<br />
        a 1px border
    </div>
</body>
</html>
```

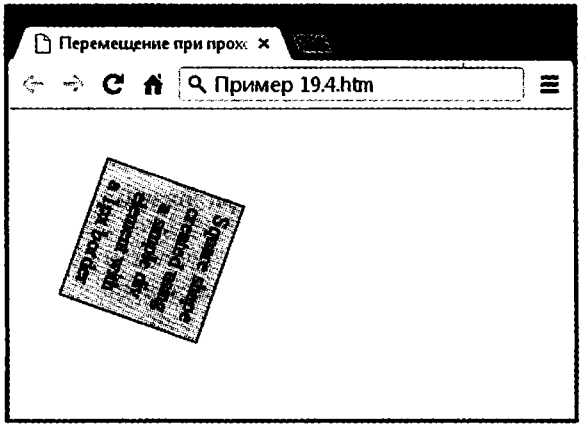



**Рис. 19.13.** Объект поворачивается и меняет цвет при прохождении над ним указателя мыши

Пример кода удовлетворяет требованиям всех разнообразных браузеров путем предоставления версий объявлений, характерных для тех или иных браузеров. Во всех самых последних браузерах (включая Internet Explorer 10 и выше) объект будет поворачиваться по часовой стрелке при прохождении над ним указателя мыши и в то же время будет медленно менять свой цвет с оранжевого на желтый.

CSS-переходы выполняются вполне продуманно, что выражается в том, что после прекращения перехода все плавно возвращается к своему исходному значению. Поэтому, если убрать указатель мыши до завершения перехода, он тут же пойдет в обратную сторону и начнет переход назад к своему исходному состоянию.

Теперь вы неплохо разбираетесь в том, что вам предлагает CSS и как этим можно воспользоваться для получения желаемых эффектов. В следующей главе мы сделаем еще один шаг в изучении CSS, рассмотрев его динамическое взаимодействие со свойствами объектной модели документа (DOM) посредством использования JavaScript.

# Проверьте ваши знания

1. Чем занимаются операторы CSS3 ^, $ и *?
2. Какое свойство используется для указания размера фонового изображения?
3. С помощью какого свойства можно указать радиус границы?
4. Как можно задать перетекание текста по нескольким колонкам?
5. Назовите четыре функции, с помощью которых можно указать CSS-цвета.
6. Как можно создать серую тень под каким-нибудь текстом с диагональным отступом вправо и вниз на 5 пикселов и с размытостью 3 пиксела?
7. Как можно показать многоточием, что текст усечен?
8. Как включить в состав своей веб-страницы веб-шрифты Google?
9. Какое CSS-объявление нужно использовать для поворота объекта на 90°?
10. Как указать переход объекта таким образом, чтобы при изменении любого из его свойств переход осуществлялся сразу в линейном режиме в течение половины секунды?

Ответы на эти вопросы можно найти в приложении А, в разделе «Ответы на вопросы главы 19».

# 20 Доступ к CSS из JavaScript

Вы уже поняли, что такое объектная модель документа — Document Object Model (DOM) и каскадные таблицы стилей — CSS, из этой главы вы узнаете, как к ним можно получать доступ непосредственно из кода JavaScript, что позволит вам создавать высокодинамичные и быстро реагирующие на действия пользователей веб-сайты.

Мы также рассмотрим использование прерываний, что позволит создавать анимацию или предоставлять любой код, который должен продолжать работу на веб-странице в фоновом режиме, например часы. Кроме того, я объясню, как в DOM добавляются новые элементы или удаляются существующие элементы, чтобы вам не приходилось заранее создавать элементы в HTML на тот случай, если коду JavaScript может чуть позже понадобиться получить к ним доступ.

## Еще одно обращение к функции getElementById

В главе 13 уже упоминалось, что символ `$` часто используется как имя функции, чтобы облегчить доступ к функции `getElementById`. Фактически основные среды программирования, такие как jQuery, применяют эту новую `$`-функцию и существенно расширили ее функциональные возможности.

Я тоже намереваюсь предоставить вам улучшенную версию этой функции, чтобы вы могли работать с элементами DOM и стилями CSS быстро и эффективно. Но чтобы избежать конфликтов со средами программирования, использующими символ `$`, я буду просто применять в качестве имени функции заглавную букву `O`, поскольку это первая буква слова object («объект»), а именно объект и будет возвращаться при вызове этой функции (тот самый объект, представленный идентификатором ID, переданным функции).

### Функция O

Основа функции `O` имеет следующий вид:

```
function O(obj)
```

```
    return document.getElementById(obj)
}
```

Только лишь этот код уже на 22 символа сокращает количество набираемого текста при вызове функции, но я намерен немного расширить функцию, позволяя передавать ей либо ID, либо объект, как показано в полной версии функции, представленной в примере 20.1.

**Пример 20.1.** Функция O

```
function O(obj)
{
    if (typeof obj == 'object') return obj
    else return document.getElementById(obj)
}
```

Если функции передается объект, она просто возвращает этот объект обратно. В противном случае она предполагает, что ей был передан ID, и возвращает объект, на который этот ID ссылается.

Но с какой стати мне захотелось добавить эту первую инструкцию, которая просто возвращает переданный ей объект?

## Функция S

Ответ на данный вопрос станет ясным, когда вы посмотрите на вспомогательную функцию, названную S и показанную в примере 20.2, которую я вам предоставляю для упрощения доступа к стилевым свойствам (или CSS) объекта.

**Пример 20.2.** Функция S

```
function S(obj)
{
    return O(obj).style
}
```

Для этой функции имя S выбрано потому, что это первая буква слова Style, а функция выполняет задачу возвращения свойства стиля (или подчиненного объекта) того элемента, на который она ссылается. Поскольку встроенная функция O принимает либо ID, либо объект, вы можете передавать функции S как ID, так и объект.

Рассмотрим, что получится, когда мы возьмем -элемент с ID myobj и установим для цвета его текста значение green (зеленый):

```
<div id='myobj'>Some text</div>

<script>
    O('myobj').style.color = 'green'
</script>
```

Предыдущий код справится с этой задачей, но значительно проще будет вызвать новую функцию S:

```
S('myobj').color = 'green'
```

Теперь рассмотрим случай, при котором объект, возвращенный в результате вызова функции O, сохранен, к примеру, в объекте по имени fred:

```
fred = O('myobj')
```

Благодаря тому способу, который используется в работе функции S, мы можем для изменения цвета на зеленый вызвать и этот объект:

```
S(fred).color = 'green'
```

Это означает, что при желании получить доступ к объекту непосредственно или через его ID, вы можете сделать это, передавая его либо функции O, либо функции S, в зависимости от того, что вам нужно. Нужно лишь помнить, что при передаче объекта (а не ID) ни в коем случае не следует брать его имя в кавычки.

## Функция C

Вам уже предоставлены две простые функции, упрощающие доступ к любому элементу на веб-странице и любому свойству стиля элемента. Но иногда вам понадобится одновременный доступ более чем к одному элементу. Это можно сделать путем присваивания имени класса CSS каждому такому элементу, как показано в следующем примере, где для каждого элемента используется класс myclass:

```
<div class='myclass'>Содержимое div-контейнера </a>
<p class='myclass'>Содержимое абзаца</p>
```

Если нужен доступ ко всем элементам страницы, использующим конкретный класс, можно воспользоваться функцией C (чье имя происходит от первой буквы в слове class), показанной в примере 20.3. Она вернет массив, состоящий из всех объектов, которые соответствуют предоставленному имени класса.

**Пример 20.3.** Функция C

```
function C(name)
{
    var elements = document.getElementsByTagName('*')
    var objects  = []

    for (var i = 0 ; i < elements.length ; ++i)
        if (elements[i].className == name)
            objects.push(elements[i])

    return objects
}
```

Разберем код по частям. В аргументе name содержится имя класса, по которому мы пытаемся извлечь объекты. Затем внутри функции создается новый объект по имени elements, содержащий все элементы документа, возвращенные путем вызова функции getElementsByTagName с аргументом '*', который означает «нужно найти все элементы»:

```
var elements = document.getElementsByTagName('*')
```

Затем создается новый массив по имени objects, куда будут помещаться все найденные объекты, соответствующие условию поиска:

```
var objects = []
```

Затем цикл for осуществляет перебор всех элементов, имеющихся в объекте elements, используя в качестве индекса переменную i:

```
for (var i = 0 ; i < elements.length ; ++i)
```

При каждом проходе цикла объект помещается в массив objects при условии, что значение свойства элемента className совпадает со строковым значением, переданным в аргументе name:

```
if (elements[i].className == name)
    objects.push(elements[i])
```

И наконец, когда цикл завершится, массив objects будет содержать все элементы в документе, который используют имя класса, являющееся значением переменной name, поэтому он возвращается функцией:

```
return objects
```

### Использование функции C

Для использования эту функцию следует просто вызвать, как показано далее, сохраняя возвращенный массив, чтобы иметь возможность получить доступ отдельно к каждому нужному элементу или (что чаще всего и бывает) ко всем элементам посредством использования цикла:

```
myarray = C('myclass')
```

Теперь можете делать с возвращенными объектами все, что нужно, например установить для их свойства textDecoration значение подчеркивания — 'underline':

```
for (i = 0 ; i < myarray.length ; ++i)
    S(myarray[i]).textDecoration = 'underline'
```

Этот код осуществляет последовательный перебор объектов в myarray[] и использует функцию S для ссылки на свойство стиля каждого объекта, задавая для свойства textDecoration значение 'underline'.

## Включение функций

Функции O и S используются во всей оставшейся части главы, поскольку они делают код короче и понятнее. Поэтому я сохранил их в файле OSC.js (наряду с функцией C, поскольку я полагаю, что она принесет вам большую пользу) в папке Chapter 20 в сопутствующем архиве примеров, который вы можете загрузить с веб-сайта http://lpmj.net.

Они могут быть включены в веб-страницу с помощью следующей инструкции, которую предпочтительнее поместить в блок <head> где-нибудь перед любым сценарием, работа которого зависит от вызова этих функций:

```
<script src='OSC.js'></script>
```

Содержимое файла OSC.js показано в примере 20.4.

**Пример 20.4.** Файл OSC.js

```
function O(obj)
{
    if (typeof obj == 'object') return obj
    else return document.getElementById(obj)
}

function S(obj)
{
    return O(obj).style
}

function C(name)
{
    var elements = document.getElementsByTagName('*')
    var objects  = []

    for (var i = 0 ; i < elements.length ; ++i)
        if (elements[i].className == name)
            objects.push(elements[i])

    return objects
}
```

# Обращение к свойствам CSS из JavaScript

Свойство textDecoration, использовавшееся в ранее показанном примере, представляет свойство CSS, имя которого в обычном виде содержит дефис: text-decoration. Но поскольку в JavaScript дефис зарезервирован для применения в качестве математического оператора, при доступе к свойству CSS, в имени которого используется дефис, этот дефис нужно опустить и перевести символ, следовавший непосредственно за ним, в верхний регистр.

Еще одним примером может послужить свойство font-size, на которое в JavaScript при помещении после оператора точки ссылаются как на fontSize:

```
myobject.fontSize = '16pt'
```

Вместо этого можно предоставить более развернутый код и воспользоваться функцией setAttribute, которая поддерживает (и фактически требует) стандартное имя свойства CSS:

```
myobject.setAttribute('font-size', '16pt')
```

Некоторые версии Microsoft Internet Explorer в определенных ситуациях слишком разборчивы в применении JavaScript-стиля имен, принадлежащих свойствам CSS. Имеется в виду применение к ним специальных версий правил, в которых используются характерные для браузера префиксы -ms-.

## Некоторые общие свойства

С помощью JavaScript вы можете изменить любое свойство любого элемента, имеющегося в веб-документе, примерно так же, как это делается с помощью CSS. Я уже показывал вам, как получить доступ к свойствам CSS, используя либо краткую форму JavaScript, либо функцию setAttribute (чтобы применить абсолютно такие же имена свойств, как и в CSS). Поэтому я не стану обременять вас детализацией всех этих сотен свойств. Вместо этого я покажу, как получить доступ к некоторым свойствам CSS, чтобы дать обзорное представление о возможностях их применения.

Сначала рассмотрим изменение нескольких свойств CSS из JavaScript, используя код примера 20.5, который в первую очередь загружает в себя три ранее упомянутые функции, затем создает -элемент и, наконец, запускает инструкции JavaScript, находящиеся внутри блока <script> кода HTML, с целью изменения различных атрибутов элемента  (рис. 20.1).

**Пример 20.5.** Обращение к свойствам CSS из JavaScript

```
<html>
    <head>
        <title>Обращение к свойствам CSS</title>
        <script src='OSC.js'></script>
    </head>
    <body>
        <div id='object'>Div-объект</div>

        <script>
             S('object').border     = 'solid 1px red'
             S('object').width      = '100px'
             S('object').height     = '100px'
             S('object').background = '#eee'
             S('object').color      = 'blue'
             S('object').fontSize   = '15pt'
             S('object').fontFamily = 'Helvetica'
             S('object').fontStyle  = 'italic'
        </script>
    </body>
</html>
```

От такого изменения свойств нет никакой практической пользы, поскольку можно также легко включить код CSS непосредственно в атрибуты элемента, но скоро мы будем изменять свойства в ответ на действия пользователя, вот тогда и проявится настоящая эффективность сочетания JavaScript и CSS.

## Другие свойства

JavaScript также открывает доступ к очень широкому диапазону других свойств, таких как ширина и высота окна браузера и любых появляющихся или присутствующих в браузере окон или фреймов, и к полезной информации, такой как родительское окно (если таковое имеется) и история URL-адресов, по которым осуществлялись визиты в текущем сеансе.

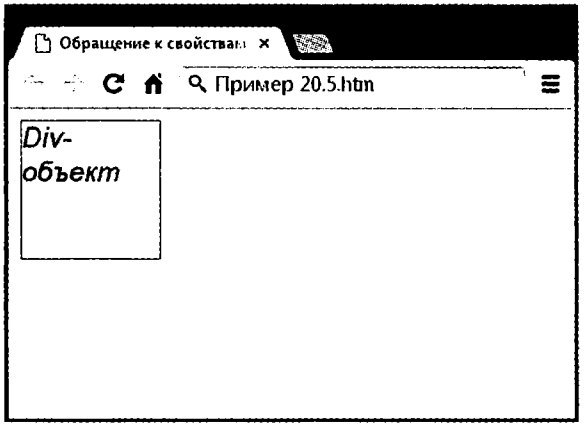



**Рис. 20.1.** Изменение стилей из JavaScript

Все эти свойства доступны из объекта `window` через оператор точка (например `window.name`). В табл. 20.1 перечислены все эти свойства с описаниями.

**Таблица 20.1.** Общие свойства объекта window

| Свойство | Описание |
|---|---|
| closed | Возвращает булево значение, показывающее, было ли закрыто окно |
| defaultStatus | Устанавливает или возвращает исходный текст панели состояния окна |
| document | Возвращает объект документа для окна |
| frames | Возвращает массив, состоящий из всех фреймов и i-фреймов окна |
| history | Возвращает для окна объект истории |
| innerHeight | Устанавливает или возвращает внутреннюю высоту области содержимого окна |
| innerWidth | Устанавливает или возвращает внутреннюю ширину области содержимого окна |
| length | Возвращает количество фреймов и i-фреймов окна |
| location | Возвращает местоположение объекта в окне |
| name | Устанавливает или возвращает имя окна |
| navigator | Возвращает для окна объект-навигатор |
| opener | Возвращает ссылку на то окно, из которого было создано данное окно |
| outerHeight | Устанавливает или возвращает внешнюю высоту окна, включая панель инструментов и полосу прокрутки |
| outerWidth | Устанавливает или возвращает внешнюю ширину окна, включая панель инструментов и полосу прокрутки |
| pageXOffset | Возвращает количество пикселов, на которое был горизонтально прокручен документ от левого края окна |
| pageYOffset | Возвращает количество пикселов, на которое был вертикально прокручен документ от верхнего края окна |
| parent | Возвращает для окна объект родительского окна |
| screen | Возвращает для окна объект экрана |

*Продолжение*

**Таблица 20.1** *(продолжение)*

| Свойство | Описание |
|---|---|
| screenLeft | Возвращает координату x окна относительно экрана во всех последних браузерах, кроме Mozilla Firefox (для которого нужно использовать screenX) |
| screenTop | Возвращает координату y окна относительно экрана во всех последних браузерах, кроме Mozilla Firefox (для которого нужно применять screenY) |
| screenX | Возвращает координату x окна относительно экрана во всех последних браузерах, кроме Opera, который возвращает неправильное значение; поддерживается в версиях Internet Explorer, предшествующих 9-й версии |
| screenY | Возвращает координату y окна относительно экрана во всех последних браузерах, кроме Opera, который возвращает неправильное значение; поддерживается в версиях Internet Explorer, предшествующих 9-й версии |
| self | Возвращает the current window |
| status | Устанавливает или возвращает текст на панели состояния окна |
| top | Возвращает верхнее окно браузера |

В отношении некоторых из этих свойств следует отметить такие моменты.

- Свойства `defaultStatus` и `status` могут быть установлены, только если пользователи изменили настройки своих браузеров и разрешили их применение (что маловероятно).
- Содержимое объекта `history` не может быть прочитано (поэтому нельзя посмотреть, какие адреса посещались вашими визитерами), но этот объект поддерживает свойство `length`, чтобы определить длину истории, а также методы `back`, `forward` и `go` для переходов на указанные страницы в истории.
- Когда нужно узнать, какое пространство доступно в текущем окне веб-браузера, следует просто прочитать значения свойств `window.innerHeight` и `window.innerWidth`. Я часто использую эти значения для размещения появляющихся в окне браузера диалоговых окон оповещения и подтверждения по центру.
- Объект `screen` поддерживает свойства, доступные только для чтения `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth` и `width`, и поэтому отлично подходит для извлечения информации о дисплее пользователя.

Многие из этих свойств могут быть просто бесценными при позиционировании на мобильных телефонах и планшетных устройствах, поскольку они дадут точную информацию об экранном пространстве, с которым придется работать, о типе используемого браузера и т. д.

Этого объема информации вполне достаточно для начала работы и для получения представления о многих новых и интересных приемах работы с JavaScript. Разумеется, существует намного больше доступных свойств и методов, которые могли бы быть рассмотрены в данной главе, но теперь, зная о том, как обращаться к свойствам и использовать их, вам нужен лишь информационный ресурс, на котором все они перечислены. Я рекомендую для начала обратиться к веб-сайту http://tinyurl.com/domproperties.

# Встроенный JavaScript

Использование тегов <script> не является единственным способом выполнения инструкций JavaScript. Получить доступ к JavaScript можно также и из тегов HTML, что и делается для повышения динамической интерактивности.

Например, для быстрого добавления эффекта при прохождении указателя мыши над объектом можно воспользоваться таким же кодом, который показан в теге <img /> в примере 20.6. Там изначально отображается картинка с яблоком, которая при прохождении над ней указателя мыши заменяется картинкой с апельсином (а при выходе указателя за пределы картинки возвращается картинка с яблоком).

**Пример 20.6.** Использование встроенного JavaScript

```
<html>
    <head>
        <title>Встроенный JavaScript</title>
    </head>
    <body>
        <img src='apple.png'
             onmouseover="this.src='orange.png'"
             onmouseout="this.src='apple.png'" />
    </body>
</html>
```

## Ключевое слово this

В предыдущем примере продемонстрировано применение ключевого слова this. Оно заставляет JavaScript работать с названным объектом, а именно с тегом <img />. Результат можно увидеть на рис. 20.2, где указатель мыши только что прошел над картинкой с яблоком.

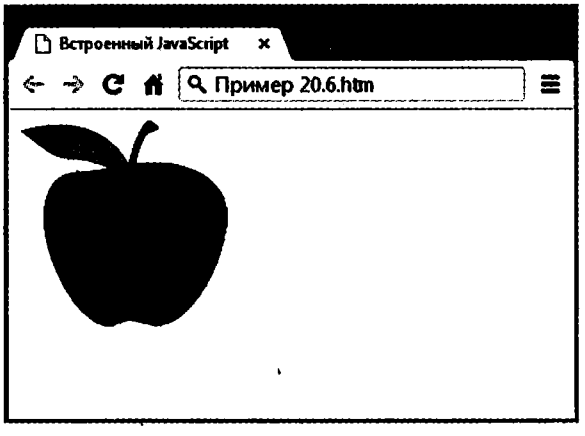



**Рис. 20.2.** Пример встроенного кода JavaScript, обрабатывающего прохождение указателя мыши над объектом

Когда ключевое слово this находится во встроенном вызове JavaScript, оно представляет вызываемый объект. А при использовании в методах класса оно представляет объект, к которому применяется метод.

# Привязка событий к объектам в сценарии

Предыдущий код является эквивалентом предоставления тегу <img /> идентификатора с последующей привязкой действий к событиям мыши этого тега, как в примере 20.7.

**Пример 20.7.** Невстроенный JavaScript

```
<html>
    <head>
        <title>Невстроенный JavaScript</title>
        <script src='OSC.js'></script>
    </head>
    <body>
        <img id='object' src='apple.png' />

        <script>
            O('object').onmouseover = function() { this.src = 'orange.png' }
            O('object').onmouseout  = function() { this.src = 'apple.png'  }
        </script>
    </body>
</html>
```

Этот код применяет ID объекта к тегу <img /> в блоке HTML, а затем продолжает работать с ним отдельно в блоке JavaScript, прикрепив к каждому событию безымянную функцию.

# Прикрепление к другим событиям

Какой бы JavaScript ни использовался, встроенный или отдельный, существуют события, к которым вы можете прикрепить действия. И активировать тем самым множество дополнительных функций, которые можете предоставить своим пользователям. В табл. 20.2 перечислены эти события и указаны условия их наступления.

**Таблица 20.2.** События и условия их наступления

| Событие | Условие его наступления |
|---|---|
| onabort | Когда загрузка изображения останавливается до ее завершения |
| onblur | Когда элемент теряет фокус |
| onchange | Когда изменяется любая часть формы |
| onclick | Когда происходит щелчок кнопкой мыши на объекте |
| ondblclick | Когда происходит двойной щелчок кнопкой мыши на объекте |
| onerror | Когда обнаруживается ошибка JavaScript |
| onfocus | Когда элемент получает фокус |
| onkeydown | Когда нажата клавиша (включая Shift, Alt, Ctrl и Esc) |

| Событие | Условие его наступления |
|---|---|
| onkeypress | Когда нажата клавиша (исключая Shift, Alt, Ctrl и Esc) |
| onkeyup | Когда клавиша отпущена |
| onload | Когда объект загрузился |
| onmousedown | Когда над элементом нажата кнопка мыши |
| onmousemove | Когда указатель мыши проходит над элементом |
| onmouseout | Когда указатель мыши покидает элемент |
| onmouseover | Когда указатель мыши заходит на элемент со стороны |
| onmouseup | Когда отпускается кнопка мыши |
| onsubmit | Когда отправляется форма |
| onreset | Когда сбрасываются данные формы |
| onresize | Когда изменяются размеры окна браузера |
| onscroll | Когда документ прокручивается |
| onselect | Когда выделяется какой-нибудь текст |
| onunload | Когда удаляется документ |

События нужно прикреплять только к тем объектам, для которых в них имеется смысл. Например, объект, не являющийся формой, не будет реагировать на событие onsubmit.

# Добавление новых элементов

Работая с JavaScript, вы можете манипулировать не только элементами и объектами, которые были предоставлены документу его кодом HTML. Вы можете создавать объекты по своему желанию, вставляя их в DOM.

Предположим, к примеру, что вам нужен новый элемент . Способ добавления его к веб-странице показан в примере 20.8.

**Пример 20.8.** Вставка элемента в DOM

```
<html>
    <head>
        <title>Добавление элементов</title>
        <script src='OSC.js'></script>
    </head>
    <body>
        В этом документе содержится только этот текст.<br /><br />

        <script>
            alert('Для добавления элемента щелкните на кнопке OK')

            newdiv    = document.createElement('div')
            newdiv.id = 'NewDiv'
            document.body.appendChild(newdiv)

            S(newdiv).border = 'solid 1px red'
            S(newdiv).width  = '100px'
            S(newdiv).height = '100px'
```

```
        newdiv.innerHTML = "Это новый объект, вставленный в DOM"
        tmp              = newdiv.offsetTop

        alert('Для удаления элемента щелкните на кнопке OK')
        newdiv.parentNode.removeChild(newdiv)
    </script>
  </body>
</html>
```

Сначала новый элемент создается с помощью функции `createElement`, затем вызывается функция `appendChild`, и элемент вставляется в DOM. После этого элементу присваиваются различные свойства, включая текст для его свойства `innerHTML` (внутреннего HTML). А затем, чтобы обеспечить немедленное отображение нового элемента на экране, значение его свойства `offsetTop` считывается во временную переменную `tmp`. Это заставляет DOM обновиться и вывести элемент на экран в любом браузере, который в противном случае выдержал бы паузу, прежде чем это сделать. В частности, это касается Internet Explorer. Результат показан на рис. 20.3.

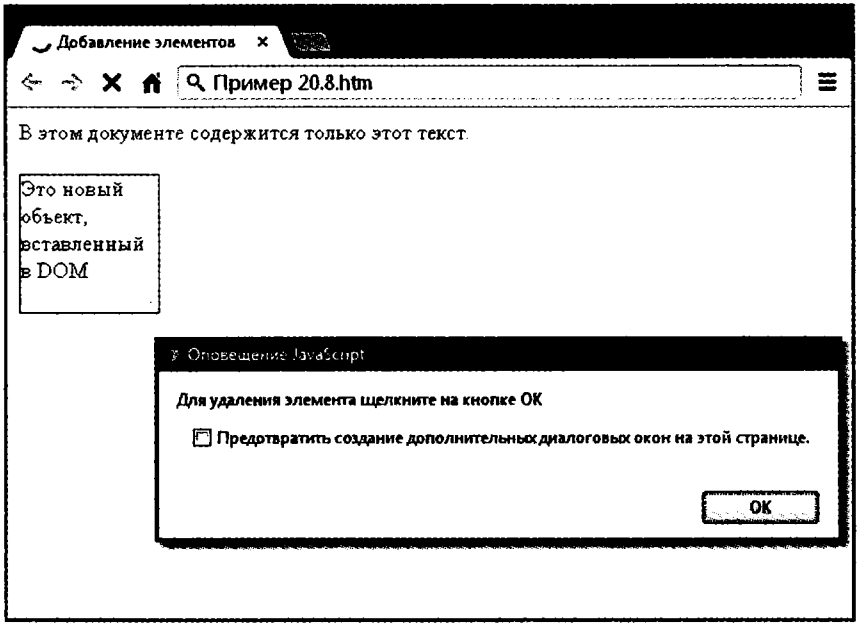



**Рис. 20.3.** Вставка нового элемента в DOM

Этот новый элемент точно такой же, как если бы он был включен в исходный HTML, и он открывает доступ к аналогичным свойствам и методам.

Иногда я использую технологию создания новых элементов, когда хочу создать окна, появляющиеся в окне браузера, потому что она не зависит от наличия запасных -элементов в DOM.

## Удаление элементов

Вы можете также удалить элементы из DOM, включая и те, которые не были вставлены с помощью кода JavaScript. Это даже проще, чем добавление элемента. Если предположить, что удаляется объект `element`, то это делается следующим образом:

```
element.parentNode.removeChild(element)
```

Этот код обращается к объекту `parentNode` элемента, поэтому он может удалить элемент из этого узла. Затем он вызывает метод этого объекта `removeChild`, передавая ему удаляемый объект. Но чтобы обеспечить немедленное обновление DOM на всех браузерах, возможно, будет предпочтительнее заменить предыдущую инструкцию следующим кодом:

```
pnode = element.parentNode
pnode.removeChild(element)
tmp   = pnode.offsetTop
```

Здесь первая инструкция помещает копию `element.parentNode` (родительского элемента объекта) в переменную `pnode`, которая (после того как дочерний элемент удаляется во второй строке) позволяет прочитать значение ее свойства `offsetTop` во временную переменную `tmp`, гарантируя тем самым полное обновление DOM.

## Альтернативы добавлению и удалению элементов

Вставка элемента предназначена для добавления к веб-странице абсолютно нового объекта. Но если вы намерены только скрывать и показывать объекты в соответствии с наступлением события `onmouseover` или какого-нибудь другого события, не забудьте, что есть пара свойств CSS, которые могут использоваться для этой цели без принятия таких радикальных мер, как создание и удаление элементов DOM.

Например, когда нужно сделать элемент невидимым, но оставить его на месте (оставляя на своих местах все окружающие его элементы), можно просто установить для свойства `visibility` объекта значение `'hidden'`:

```
myobject.visibility = 'hidden'
```

А для повторного отображения объекта можно воспользоваться следующим кодом:

```
myobject.visibility = 'visible'
```

Можно также свернуть элемент, чтобы он занимал нулевую ширину и высоту (и чтобы все окружающие его объекты заняли освободившееся пространство):

```
myobject.display = 'none'
```

Для последующего восстановления элемента в его исходных размерах можно воспользоваться следующим кодом:

```
myobject.display = 'block'
```

И конечно же, в вашем распоряжении всегда есть свойство innerHTML, с помощью которого можно изменить код HTML, примененный к элементу. Например:

```
myElement.innerHTML = '<b>Замена HTML</b>'
```

Можно также воспользоваться упомянутой ранее функцией O:

```
O('someid').innerHTML = 'Новое содержимое'
```

Можно также заставить элемент показаться исчезнувшим:

```
O('someid').innerHTML = ''
```

Не забывайте обо всех других полезных свойствах CSS, к которым можно обратиться из JavaScript. Например, для переключения объекта из видимого в невидимое состояние и обратно можно воспользоваться свойством непрозрачности, а для изменения размеров объекта можно изменить значения свойств width и height. И конечно же, применяя для свойства position значения 'absolute', 'static' или 'relative', вы можете даже поместить объект в любое место окна браузера (или снаружи).

# Использование прерываний

JavaScript предоставляет доступ к *прерываниям*, методу, с помощью которого можно попросить браузер вызвать ваш код после определенного периода времени или даже продолжать вызовы через указанные интервалы времени. Это даст вам средства обработки фоновых задач, таких как обмен данными с помощью Ajax или даже такие средства, как анимация веб-элементов.

Существует два типа прерываний, setTimeout и setInterval, сопровождающихся функциями clearTimeout и clearInterval для их выключения.

## Использование функции setTimeout

При вызове функции setTimeout передается код JavaScript или имя функции и значение в миллисекундах, отображающее продолжительность задержки запуска кода на выполнение:

```
setTimeout(dothis, 5000)
```

Ваша функция dothis может иметь следующий вид:

```
function dothis()
{
    alert('Это ваш будильник!');
}
```

Как ни удивительно, вы не можете просто указать alert() (с круглыми скобками) в качестве функции, вызываемой setTimeout, потому что функция будет тут же выполнена. Передавать имя функции, чтобы код был выполнен только по истечении указанного времени, можно только без круглых скобок, служащий для указания аргументов (например, alert).

### Передача строки

Если исполняемой функции нужно передать аргумент, то функции setTimeout можно также передать строковое значение, которое не будет выполняться, пока не наступит нужное время. Например:

```
setTimeout("alert('Hello!')", 5000)
```

Фактически если после каждой инструкции ставить точку с запятой, то можно передать столько строк кода JavaScript, сколько нужно:

```
setTimeout("document.write('Starting'); alert('Hello!')", 5000)
```

### Повторение тайм-аутов

Для предоставления повторяющихся прерываний, создаваемых функцией setTimeout, некоторые программисты используют технологию вызова функции setTimeout из кода, вызываемого этой же функцией, как в следующем примере, который инициирует бесконечный цикл вывода окон предупреждений:

```
setTimeout(dothis, 5000)

function dothis()
{
  setTimeout(dothis, 5000)
  alert('Я вас раздражаю!')
}
```

Теперь окно предупреждения будет появляться каждые пять секунд.

## Отмена тайм-аута

После установки тайм-аута вы можете отменить его, если предварительно сохранили значение, возвращенное при начальном вызове функции setTimeout:

```
handle = setTimeout(dothis, 5000)
```

Теперь, когда у вас есть это значение в переменной handle, вы можете отменить прерывание в любой момент, вплоть до истечения назначенного срока:

```
clearTimeout(handle)
```

В результате этого прерывание полностью забывается и код, назначенный ему для выполнения, никогда не выполняется.

## Использование функции setInterval

Самый простой способ установки регулярных прерываний заключается в использовании функции setInterval. Она работает точно так же, как и описанная выше, за исключением того, что, проявив себя после интервала, указанного вами в миллисекундах, она сделает это снова, после того как этот же интервал снова пройдет, и так далее до бесконечности, пока вы ее не остановите.

В примере 20.9 эта функция используется для вывода в браузере простых часов, показанных на рис. 20.4.

**Пример 20.9.** Часы, созданные с помощью прерываний

```
<html>
    <head>
        <title>Использование setInterval</title>
        <script src='OSC.js'></script>
    </head>
    <body>
        Текущее время: <span id='time'>00:00:00</span><br />

        <script>
            setInterval("showtime(O('time'))", 1000)

            function showtime(object)
            {
                var date = new Date()
                object.innerHTML = date.toTimeString().substr(0,8)
            }
        </script>
    </body>
</html>
```

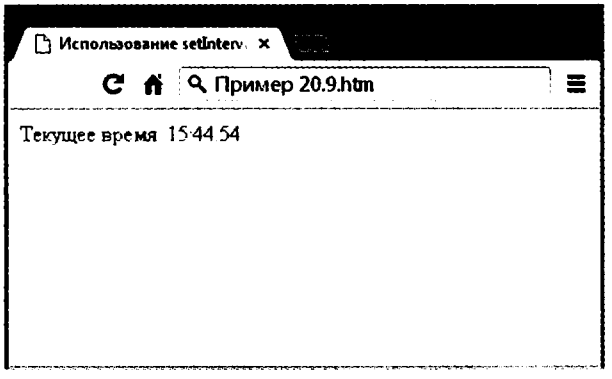



**Рис. 20.4.** Поддержка показаний правильного времени с помощью прерываний

При каждом вызове функции ShowTime она присваивает объекту date текущее время и дату с помощью вызова функции Date:

```
var date = new Date()
```

Затем свойству innerHTML объекта, переданного функции showtime (то есть object), устанавливается значение текущего времени в часах, минутах и секундах, как определено вызовом функции toTimeString. В результате возвращается строка «09:57:17 UTC+0530», которая затем усекается до первых восьми символов с помощью вызова функции substr:

```
object.innerHTML = date.toTimeString().substr(0,8)
```

### Использование функции

Чтобы воспользоваться этой функцией, сначала нужно создать объект, чье свойство innerHTML будет применено для отображения времени, как в следующем коде HTML:

```
Текущее время: <span id='time'>00:00:00</span>
```

Затем в блоке кода <script> вызов помещается в функцию setInterval:

```
setInterval("showtime(O('time'))", 1000)
```

Этот вызов передает функции setInterval строку, содержащую следующую инструкцию, настроенную на выполнение один раз в секунду (каждую 1000 миллисекунд):

```
showtime(O('time'))
```

В том редком случае, когда кто-нибудь отключил в своем браузере JavaScript (что некоторые делают из соображений безопасности), ваш JavaScript не запустится и пользователь увидит исходное значение 00:00:00.

### Отмена интервала

Чтобы остановить повторяющийся интервал, при первой установке интервала путем вызова функции setInterval вы должны пометить для себя в переменной handle дескриптор этого интервала:

```
handle = setInterval("showtime(O('time'))", 1000)
```

Теперь можно остановить часы в любое время, сделав следующий вызов:

```
clearInterval(handle)
```

Можно также настроить таймер на остановку через определенный период времени:

```
setTimeout("clearInterval(handle)", 10000)
```

Эта инструкция выдаст прерывание через 10 секунд (10 000 миллисекунд), которое очистит повторяющиеся интервалы.

## Использование прерываний для анимации

Путем сочетания нескольких свойств CSS с повторяющимся прерыванием можно создавать всевозможные анимации и эффекты.

Например, код в примере 20.10 перемещает прямоугольник по верхней части окна браузера, все время увеличивая его в размерах, как показано на рис. 20.5. Когда значение переменной LEFT сбрасывается в 0, анимация начинается снова.

**Пример 20.10.** Простая анимация

```
<html>
    <head>
        <title>Простая анимация</title>
        <script src='OSC.js'></script>
        <style>
```

```
            #box {
            position  :absolute;
            background:orange;
            border    :1px solid red; }
        </style>
    </head>
    <body>
        <div id='box'></div>

        <script>
            SIZE = LEFT = 0

            setInterval(animate, 30)

            function animate()
            {
                SIZE += 10
                LEFT += 3
                if (SIZE == 200) SIZE = 0
                if (LEFT == 600) LEFT = 0

                S('box').width  = SIZE + 'px'
                S('box').height = SIZE + 'px'
                S('box').left   = LEFT + 'px'
            }
        </script>
    </body>
</html>
```

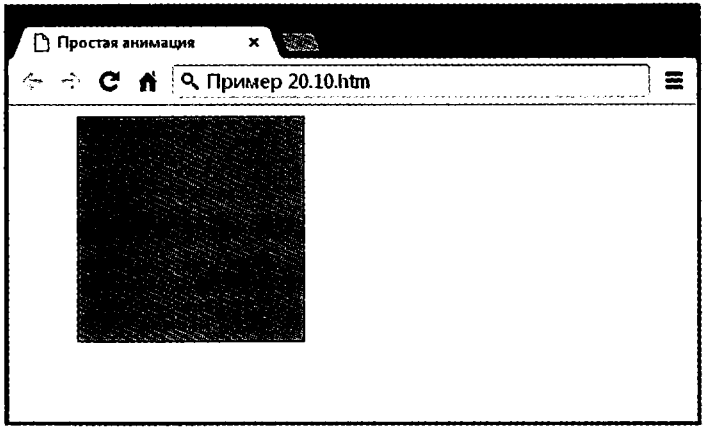



**Рис. 20.5.** Объект плавно движется слева, одновременно меняя свой размер

В блоке <head> документа объекту box устанавливается цвет фона 'orange' (оранжевый) со значением его границы (border) '1px solid red', а его свойству позиционирования position задается значение absolute, чтобы ему разрешалось перемещаться по окну браузера.

Затем в функции animate происходит постоянное обновление глобальных переменных SIZE и LEFT, а затем их значения применяются к атрибутам стиля width, height и left объекта box (с добавлением после каждого значения строки 'px' для указания, что значение в пикселах), таким образом анимируя объект с частотой один раз каждые 30 миллисекунд. Тем самым задается скорость 33,33 кадра в секунду (1000 / 30 миллисекунд).

Вот и завершается введение во все темы, рассматриваемые в данной книге, и теперь вы находитесь на пути превращения в опытного веб-разработчика. Но перед тем, как завершить книгу, в последней главе я хочу собрать все, что здесь рассматривалось, вместе, в единый проект, чтобы вы на практике увидели, как все технологии совмещаются друг с другом.

# Проверьте ваши знания

1. Для чего предназначены функции O, S и C?
2. Назовите два способа изменения CSS-атрибута объекта.
3. Какие свойства предоставляют доступную в окне браузера ширину и высоту?
4. Как можно задать какие-нибудь действия при прохождении указателя мыши над объектом, а затем при выходе за границы объекта?
5. Какая функция JavaScript создает новые элементы и какая функция добавляет их к DOM?
6. Как сделать элемент а) невидимым и б) сжатым до нулевых размеров?
7. Какая функция задает одиночное событие в будущем времени?
8. Какая функция устанавливает повторяющиеся события через указанный интервал времени?
9. Как можно освободить элемент от его места на веб-странице, чтобы он мог перемещаться?
10. Какая должна быть установлена задержка между событиями (в миллисекундах) для получения скорости анимации 50 кадров в секунду?

Ответы на эти вопросы можно найти в приложении A, в разделе «Ответы на вопросы главы 20».

# 21 Объединение технологий

В завершение книги я хочу привести вам реальный пример использования рассмотренных технологий, с которым вы можете досконально разобраться. В действительности это несколько примеров, объединенных в простом проекте социальной сети, имеющей все атрибуты, которые ожидают обнаружить на подобном сайте.

В разных файлах проекта представлены примеры создания таблиц MySQL и доступа к базе данных, таблиц стилей CSS, включения других файлов, управления сессией, доступа к DOM, Ajax-вызовов, обработки событий и ошибок, загрузки файлов на сервер, работы с изображениями и решения многих других задач.

Каждый файл, приводимый в качестве примера, является завершенной и самодостаточной программой, способной работать совместно с остальными файлами с целью построения полностью работоспособного сайта социальной сети. К тому же, включая таблицу стилей, вы можете полностью изменить внешний вид проекта. За счет своего небольшого объема и несложной конструкции конечный продукт особенно полезен на мобильных платформах, таких как смартфоны или планшетники, но он также хорошо будет работать и на полноразмерных настольных компьютерах.

Можно взять любой представляющийся полезным фрагмент кода и дополнить его в соответствии с поставленными задачами. Возможно, у вас даже появится желание создать на основе этих файлов собственную социальную сеть.

## Проектирование сайта социальной сети

Перед написанием кода я определяю для себя важные составляющие подобного сайта, среди которых:

- процесс регистрации;
- форма для входа на сайт;
- средство для завершения работы с сайтом;
- управление сессией;
- пользовательские профили с загруженными миниатюрными изображениями;
- каталог участников сети;
- добавление участников в список друзей;

- открытый и закрытый обмен сообщениями между участниками;
- способ стилевого оформления проекта.

Я решил назвать проект «Сообщество Робина» — Robin's Nest. Если выберете другое имя, то для внесения изменений потребуется модифицировать лишь одну строку кода (в `functions.php`).

# Информация на веб-сайте

Все примеры, приводимые в данной главе, можно найти на прилагаемом к книге веб-сайте, который размещен по адресу **http://lpmj.net**. Можно также загрузить примеры с этого сайта на свой компьютер, щелкнув на ссылке **Download 2nd Ed. Examples** (Загрузка примеров 2-го издания). В результате будет загружен архивный файл `2nd_edition_examples.zip`, содержимое которого можно извлечь и поместить в удобное для вас место.

С учетом того, что данная глава представляет особый интерес, внутри ZIP-файла есть папка **robinsnest**, в которой все следующие примеры сохранены с использованием тех имен, которые требуются этому учебному приложению. Поэтому вы можете просто скопировать все эти примеры в свою рабочую папку **web**, чтобы увидеть их в действии.

# Файл functions.php

Перейдем непосредственно к проекту и начнем с примера 21.1, `functions.php`, представляющего собой включаемый файл, в который входят основные функции. Но в этом файле содержатся не только функции. Я добавил в него сведения, необходимые для входа в базу данных, чтобы не использовать для этой цели лишний файл.

В первых шести строках кода определяются хост, имя базы данных, имя пользователя и пароль, используемый для входа в базу данных. Неважно, как вы назовете базу данных, главное, чтобы она уже существовала (создание новой базы данных рассматривалось в главе 8). Нужно также обеспечить присвоение переменным `$dbuser` и `$dbpass` правильных значений имени пользователя и пароля для входа в MySQL. Если они имеют такие значения, то выполнение следующих двух строк кода приведет к подключению к MySQL и выбору базы данных. Последняя из начальных инструкций устанавливает имя сайта социальной сети, присваивая значение `Robin's Nest` переменной `$appname`. Именно здесь при желании можно заменить это имя другим.

## Функции

В проекте используются пять основных функций:

- `createTable` — проверяет факт существования таблицы и создает отсутствующую таблицу;
- `queryMysql` — выдает запрос к MySQL, а в случае сбоя выводит сообщение об ошибке;

- destroySession — уничтожает PHP-сессию и очищает машину от ее данных для завершения сеанса работы пользователей;
- sanitizeString — удаляет потенциально вредный код или теги из информации, введенной пользователем;
- showProfile — отображает миниатюрные изображения пользователей и их записи About me (Обо мне), если таковые имеются.

Работа всех этих функций должна быть вам понятна. За исключением, может быть, функции showProfile, которая осуществляет поиск изображения по имени user.jpg (где user — это пользовательское имя текущего пользователя) и после успешного поиска выводит это изображение на экран. Она также отображает любой текст About me (Обо мне), который пользователь мог сохранить.

Все нуждающиеся в этом функции снабжены кодом обработки ошибок, который позволяет перехватывать любую опечатку или другие допущенные ошибки ввода и сгенерировать сообщение об ошибке. Но если какая-нибудь из этих функций используется на рабочем сервере, то вам, скорее всего, захочется предоставить для этой цели собственные обработчики ошибок, чтобы сделать код более дружелюбным по отношению к пользователю.

Наберите код примера 21.1 и сохраните его в файле functions.php (или загрузите файл с прилагаемого к книге веб-сайта), после чего вы будете готовы перейти к изучению следующего раздела.

**Пример 21.1.** functions.php

```
<?php // functions.php
$dbhost  = 'localhost';      // Эта строка вряд ли нуждается в изменении
$dbname  = 'anexistingdb';   // А значения этих переменных
                             // поменяйте...
$dbuser  = 'robinsnest';     // ... на те, что соответствуют
$dbpass  = 'apassword';      // ... вашим настройкам
$appname = "Robin's Nest";   // ... и предпочтениям

mysql_connect($dbhost, $dbuser, $dbpass) or die(mysql_error());
mysql_select_db($dbname) or die(mysql_error());

function createTable($name, $query)
{
    queryMysql("CREATE TABLE IF NOT EXISTS $name($query)");
    echo "Таблица '$name' создана или уже существовала<br />";
}

function queryMysql($query)
{
    $result = mysql_query($query) or die(mysql_error());
    return $result;
}

function destroySession()
{
```

```
    $_SESSION=array();

    if (session_id() != "" || isset($_COOKIE[session_name()]))
        setcookie(session_name(), '', time()-2592000, '/');

    session_destroy();
}

function sanitizeString($var)
{
    $var = strip_tags($var);
    $var = htmlentities($var);
    $var = stripslashes($var);
    return mysql_real_escape_string($var);
}

function showProfile($user)
{
    if (file_exists("$user.jpg"))
        echo "<img src='$user.jpg' align='left' />";

    $result = queryMysql("SELECT * FROM profiles WHERE user='$user'");

    if (mysql_num_rows($result))
    {
        $row = mysql_fetch_row($result);
        echo stripslashes($row[1])   "<br clear=left /><br />";
    }
}
?>
```

# Файл header.php

Чтобы сохранить единство стиля, каждая страница проекта должна иметь доступ к одному и тому же набору функций. Поэтому я поместил соответствующие настройки в файл header.php, показанный в примере 21.2. Этот файл включается практически во все остальные файлы, и он, в свою очередь, включает в себя файл functions.php. Поэтому в каждом файле необходимо включение только одного файла.

Код файла header.php начинается с вызова функции session_start. Из материалов главы 12 следует, что эта функция настраивает сессию, которая будет запоминать конкретные значения, необходимые для использования в различных PHP-файлах. Затем устанавливается тип документа и из файла osc.js берутся JavaScript-функции O, S и C, рассмотренные в главе 20.

После запуска сессии программа проверяет, присвоено ли значение элементу массива сессии с индексом 'user'. Если значение присвоено, значит, пользователь вошел на сайт и значение переменой $loggedin установлено как TRUE.

Благодаря использованию значения переменной $loggedin и блока if отображается один из двух наборов меню. Набор для не вошедшего на сайт пользователя предлагает выбор только из главной страницы — Home, регистрации — Sign up и входа на сайт — Log in, а версия меню для вошедших на сайт предлагает полный доступ к функциям проекта. Кроме того, если пользователь вошел на сайт, его имя появляется в скобках в заголовке страницы и помещается после основного заголовка. Везде, где нужно поместить имя пользователя, можно свободно ссылаться на переменную $user, потому что, если пользователь не вошел, эта переменная будет пуста и никак не повлияет на внешний вид выводимой информации.

Стилевые настройки, применяемые в этом файле, находятся в файле styles.css, который будет подробно рассмотрен в конце главы. Кроме всего прочего, с его помощью создается широкий заголовок с цветным фоном и ссылки на страницах превращаются в кнопки с закругленными углами.

**Пример 21.2.** header.php

```
<?php // header.php
session_start();
echo "<!DOCTYPE html>\n<html><head><script src='OSC.js'></script>";
include 'functions.php';

$userstr = ' (Guest)';

if (isset($_SESSION['user']))
{
    $user = $_SESSION['user'];
    $loggedin = TRUE;
    $userstr  = " ($user)";
}
else $loggedin = FALSE;

echo "<title>$appname$userstr</title><link rel='stylesheet'"
     "href='styles.css' type='text/css' />" .
     "</head><body><div class='appname'>$appname$userstr</div>";

if ($loggedin)
{
    echo "<br /><ul class='menu'>" .
         "<li><a href='members.php?view=$user'>Home</a></li>"
         "<li><a href='members.php'>Members</a></li>" .
         "<li><a href='friends.php'>Friends</a></li>".
         "<li><a href='messages.php'>Messages</a></li>" .
         "<li><a href='profile.php'>Edit Profile</a></li>" .
         "<li><a href='logout.php'>Log out</a></li></ul><br />";
}
else
{
    echo ("<br /><ul class='menu'>" .
         "<li><a href='index.php'>Home</a></li>" .
         "<li><a href='signup.php'>Sign up</a></li>"
```

```
        "<li><a href='login.php'>Log in</a></li></ul><br />" .
        "<span class='info'>&#8658; You must be logged in to "
        "view this page.</span><br /><br />");
        // Для просмотра этой страницы нужно войти на сайт
}
?>
```

Использование тега <br />, продемонстрированное в предыдущем примере, — быстрый, но довольно грубый способ создания разрядки в макете страницы. В данном примере он вполне приемлем, но вообще-то, наверное, для тонкой настройки разрядки элементов придется воспользоваться полями CSS.

# Файл setup.php

После создания двух включаемых файлов настала очередь настройки используемых ими MySQL-таблиц. Это делается с помощью кода из примера 21.3, setup.php, который следует набрать и загрузить в свой браузер до вызова любых других файлов, в противном случае будут получены многочисленные сообщения об ошибках MySQL.

Создаваемые таблицы весьма лаконичны и имеют следующие имена и столбцы:

- members — имя пользователя *user* (проиндексированный столбец), пароль *pass*;
- messages — идентификатор *id* (проиндексированный столбец), автор *auth* (проиндексированный столбец), адресат *recip*, тип сообщения *pm*, сообщение *message*;
- friends — имя пользователя *user* (проиндексированный столбец), имя пользователей-друзей *friend*;
- profiles — имя пользователя *user* (проиндексированный столбец), **About me** (Обо мне) *text*.

Поскольку функция createTable сначала проверяет факт существования таблицы, эта программа может быть безопасно вызвана несколько раз без выдачи сообщений об ошибках.

Вполне возможно, что при принятии решения о расширении проекта вам понадобится добавить к этим таблицам множество дополнительных столбцов. После принятия такого решения для пересоздания таблицы может потребоваться MySQL-команда DROP TABLE.

**Пример 21.3.** setup.php

```
<html><head><title>Setting up database</title></head><body>

<h3>Setting up...</h3> // Настройка...

<?php // setup.php
include_once 'functions.php';

createTable('members',
```

```
            'user VARCHAR(16),
            pass VARCHAR(16),
            INDEX(user(6))');

createTable('messages',
            'id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
            auth VARCHAR(16),
            recip VARCHAR(16),
            pm CHAR(1),
            time INT UNSIGNED,
            message VARCHAR(4096),
            INDEX(auth(6)),
            INDEX(recip(6))');

createTable('friends',
            'user VARCHAR(16),
            friend VARCHAR(16),
            INDEX(user(6)),
            INDEX(friend(6))');

createTable('profiles',
            'user VARCHAR(16),
            text VARCHAR(4096),
            INDEX(user(6))');
?>

<br />...done. //     завершена.
</body></html>
```

# Файл index.php

Это очень простой, но тем не менее необходимый файл, без которого у проекта не будет главной страницы. Он всего лишь отображает приветствие. В настоящем приложении это может быть страница, сообщающая о достоинствах вашего сайта, подталкивающая посетителя к регистрации.

Кстати, если все MySQL-таблицы созданы и включаемые файлы сохранены, вы можете загрузить файл примера 21.4, index.php, в свой браузер, чтобы получить первое представление о новом приложении. На экране должно появиться изображение, показанное на рис. 21.1.

**Пример 21.4.** index.php

```
<?php // index.php
include_once 'header.php';

echo "<br /><span class='main'>Welcome to Robin's Nest.";
                    // Добро пожаловать в Сообщество Робина

if ($loggedin) echo " $user, you are logged in.";
```

```
                    // вы вошли на сайт

else          echo ' please sign up and/or log in to join in.';
      // Пожалуйста, зарегистрируйтесь и (или) войдите на сайт
?>

</span><br /><br /></body></html>
```

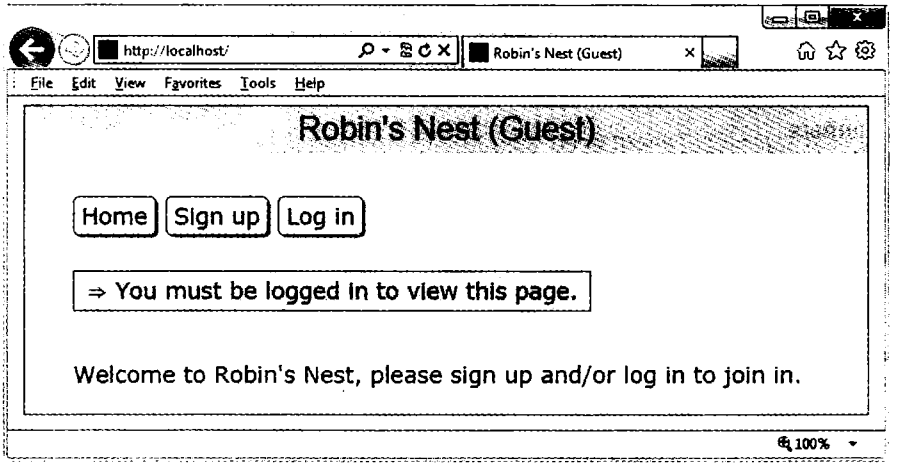



**Рис. 21.1.** Главная страница сайта

# Файл signup.php

Теперь нам нужен модуль, позволяющий пользователям присоединиться к новой сети, который показан в примере 21.5, `signup.php`. Это более длинная программа, но все ее части вам уже встречались.

Начнем ее изучение с блока HTML, расположенного в конце программы. Это простая форма, дающая возможность ввести имя пользователя и пароль. Но обратите внимание на использование пустого `<span>`-контейнера с атрибутом `id`, имеющим значение `'info'`. В этот контейнер будут помещены результаты Ajax-вызова, имеющегося в этой программе, с помощью которого проверяется возможность использования того имени пользователя, которое вы хотите. Полное описание принципов работы этого вызова дано в главе 17.

## Проверка возможности использования желаемого имени пользователя

Вернемся к началу программы, где имеется блок кода JavaScript, начинающийся с функции `checkUser`. Эта функция вызывается событием JavaScript `onBlur`, возникающим при перемещении фокуса за пределы принадлежащего форме поля `username`. Сначала эта функция вставляет в упоминавшийся ранее `span`-контейнер

(у которого id имеет значение 'info') пустую строку, которая очищает этот контейнер, если он уже имел какое-нибудь содержимое.

Затем делается запрос к программе checker.php, которая сообщает о возможности использования имени пользователя *user*. После чего возвращенный Ajax-вызовом результат — приветственное сообщение — помещается в -контейнер с идентификатором 'info'.

За блоком JavaScript следует PHP-код, известный по разделу главы 16, в котором рассматривалась проверка данных формы. Этот блок кода также использует функцию sanitizeString, чтобы удалить потенциально вредные символы перед поиском имени пользователя в базе данных, и если такое имя еще никем не используется, он вставляет новое имя пользователя $user и пароль $pass в таблицу базы данных.

После успешной регистрации пользователю предлагается войти на сайт. Более гибкой реакцией на появление нового пользователя мог бы стать автоматический вход на сайт, но я не захотел излишне усложнять код и оставил модули регистрации и входа не связанными друг с другом. Я уверен, что при желании вы сможете и сами без особого труда реализовать подобную функцию.

В этом файле используется CSS-класс fieldname, предназначенный для приведения в порядок полей формы и их точного выравнивания друг под другом в столбцах. При загрузке в браузер (вместе с показанным далее файлом checkuser.php) эта программа отобразит информацию, показанную на рис. 21.2, из которой видно, что Ajax-вызов помог определить доступность имени Robin для использования. Если нужно, чтобы в поле пароля показывались одни только звездочки, измените тип этого поля с text на password.

**Пример 21.5.** signup.php

```
<?php // signup.php
include_once 'header.php';

echo <<<_END
<script>
function checkUser(user)
{
    if (user.value == '')
    {
        O('info').innerHTML = ''
        return
    }

    params  = "user=" + user.value
    request = new ajaxRequest()
    request.open("POST", "checkuser.php", true)
    request.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded")
    request.setRequestHeader("Content-length", params.length)
    request.setRequestHeader("Connection", "close")

    request.onreadystatechange = function()
```

```
    {
        if (this.readyState == 4)
           if (this.status == 200)
              if (this.responseText != null)
                 O('info').innerHTML = this.responseText
    }
    request.send(params)
}

function ajaxRequest()
{
    try {var request = new XMLHttpRequest()}
    catch(e1){
        try {request = new ActiveXObject("Msxml2.XMLHTTP")}
        catch(e2){
            try {request = new ActiveXObject("Microsoft.XMLHTTP")}
            catch(e3){
                request = false
    } } }
    return request
}
</script>
<div class='main'><h3>Please enter your details to sign up</h3>
        // Введите, пожалуйста, свои регистрационные данные
_END;

$error = $user = $pass = "";
if (isset($_SESSION['user'])) destroySession();

if (isset($_POST['user']))
{
    $user = sanitizeString($_POST['user']);
    $pass = sanitizeString($_POST['pass']);

    if ($user == "" || $pass == "")
       $error = "Данные введены не во все поля<br /><br />";
    else
    {
      if (mysql_num_rows(queryMysql("SELECT * FROM members
                                     WHERE user='$user'")))
          $error = "Такое имя уже существует<br /><br />";
        else
            {
            queryMysql("INSERT INTO members VALUES('$user',
                                                   '$pass')");
            die("<h4>Account created</h4>Please Log in.<br />
                                                     <br />");
                        // Учетная запись создана,
                        // пожалуйста, войдите на сайт
        }
    }
```

```
}

echo <<<_END
<form method='post' action='signup.php'>$error
<span class='fieldname'>Username</span>
<input type='text' maxlength='16' name='user' value='$user'
    onBlur='checkUser(this)'/><span id='info'></span><br />
<span class='fieldname'>Password</span>
<input type='text' maxlength='16' name='pass'
    value='$pass' /><br />
_END;
?>

<span class='fieldname'> </span>
<input type='submit' value='Sign up' />
</form></div><br /></body></html>
```

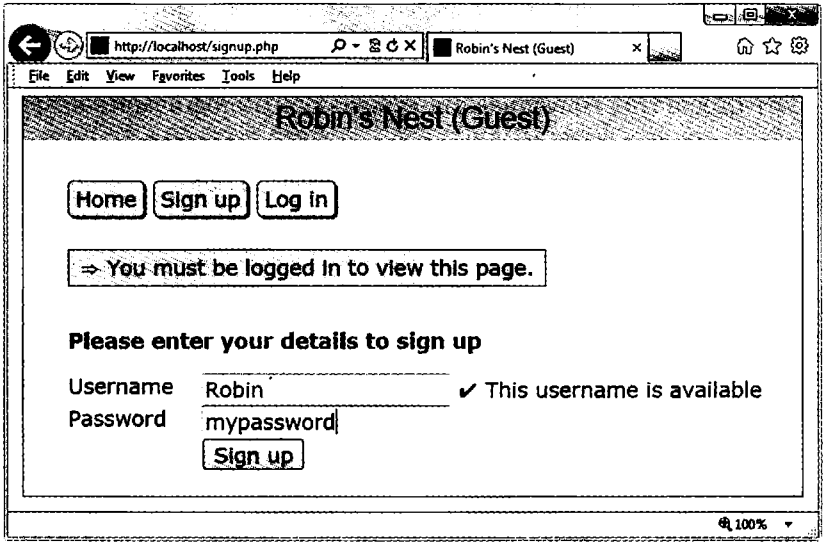



**Рис. 21.2.** Страница регистрации

При использовании рабочего сервера я не рекомендую сохранять пользовательские пароли так, как это сделано с целью экономии места и упрощения кода в данном проекте, то есть в открытом виде. К паролям нужно подмешивать произвольные строки и хранить их в виде хеш-строк, получаемых с помощью MD5 или других односторонних функций. Более подробно этот процесс рассмотрен в главе 12.

# Файл checkuser.php

Файл `checkuser.php`, показанный в примере 21.6, предназначен для работы с файлом `signup.php`. В нем содержится программа, осуществляющая поиск имени пользова-

теля в базе данных и возвращающая строку, свидетельствующую о том, что такое имя уже было кем-то использовано.

Поскольку работа этой программы зависит от функций sanitizeString и queryMysql, в нее в самом начале включается файл functions.php. Если переменная, являющаяся элементом массива $_POST, который имеет ключ 'user', имеет какое-нибудь значение, функция ищет это значение в базе данных. В зависимости от того, используется такое значение в качестве имени пользователя или нет, выводит либо строку «К сожалению, имя занято» (Sorry, already taken), либо строку «Это имя доступно» (Username available). Для решения данной задачи достаточно просто проверить значение, возвращенное функцией mysql_num_rows. Если будет возвращен нуль, значит, такое имя не найдено, а если единица, то запись с таким именем уже существует.

Для установки перед строкой либо крестика, либо галочки используются HTML-элементы ✘ и ✔.

**Пример 21.6.** checkuser.php

```
<?php // checkuser.php
include_once 'functions.php';

if (isset($_POST['user']))
{
    $user = sanitizeString($_POST['user']);

    if (mysql_num_rows(queryMysql("SELECT * FROM members
        WHERE user='$user'")))
        echo  "<span class='taken'> &#x2718; " .
              "Sorry, this username is taken</span>";
               // К сожалению, имя занято
    else echo "<span class='available'> &#x2714; ".
              "This username is available</span>";
               // Это имя доступно
}
?>
```

# Файл login.php

После того как пользователи получили возможность регистрироваться на сайте в файле login.php, показанном в примере 21.7, предоставляется код, который необходим для входа на сайт. Страница, выводимая этим кодом, как и страница регистрации, похожа на обычную HTML-форму, имеет простую проверку на отсутствие ошибок, а также использует функцию sanitizeString перед отправкой запроса к базе данных MySQL.

Стоит обратить особое внимание на присваивание в случае успешной проверки переменным сессии (элементам массива с ключами 'user' и 'pass') значений имени пользователя и пароля. На период активности текущей сессии эти переменные будут доступны всем программам проекта, позволяя им автоматически предоставлять доступ вошедшим на сайт пользователям.

Возможно, у вас возникнет вопрос, почему в случае успешного вход на сайт используется функция die. Это сделано из соображений экономии, поскольку данная команда объединяет в себе сразу две команды: echo и exit. Для стилевого оформления этого файла (как и большинства других), чтобы задать отступ содержимого от левого края, применяется класс main.

Если вызвать эту программу в браузере, появится изображение, показанное на рис. 21.3. Обратите внимание на то, как для маскировки пароля звездочками тегу <input /> был присвоен тип password, чтобы никто из заглядывающих пользователю через плечо не смог его увидеть.

**Пример 21.7.** login.php

```
<?php // login.php
include_once 'header.php';
echo "<div class='main'>
            <h3>Please enter your details to log in</h3>";
            // Введите, пожалуйста, свои данные для входа на сайт
$error = $user = $pass = "";

if (isset($_POST['user']))
{
    $user = sanitizeString($_POST['user']);
    $pass = sanitizeString($_POST['pass']);

    if ($user == "" || $pass == "")
    {
        $error = "Not all fields were entered<br />";
                // Данные введены не во все поля
    }
    else
    {
        $query = "SELECT user,pass FROM members
                WHERE user='$user' AND pass='$pass'";

        if (mysql_num_rows(queryMysql($query)) == 0)
        {
        $error = "<span class='error'>Username/Password
                    invalid</span><br /><br />";
         // Ошибка при вводе пары "имя пользователя – пароль"
        }
        else
        {
            $_SESSION['user'] = $user;
            $_SESSION['pass'] = $pass;
            die("You are now logged in." .
                "Please <a href='members.php?view=$user'>" .
                "click here</a> to continue.<br /><br />");
                // Вы уже вошли на сайт. Пожалуйста, щелкните на этой ссылке
        }
    }
```

```
}

echo <<<_END
<form method='post' action='login.php'>$error
<span class='fieldname'>Username</span><input type='text'
    maxlength='16' name='user' value='$user' /><br />
<span class='fieldname'>Password</span><input type='password'
    maxlength='16' name='pass' value='$pass' />
_END;
?>

<br />
<span class='fieldname'> </span>
<input type='submit' value='Login' />
</form><br /></div></body></html>
```

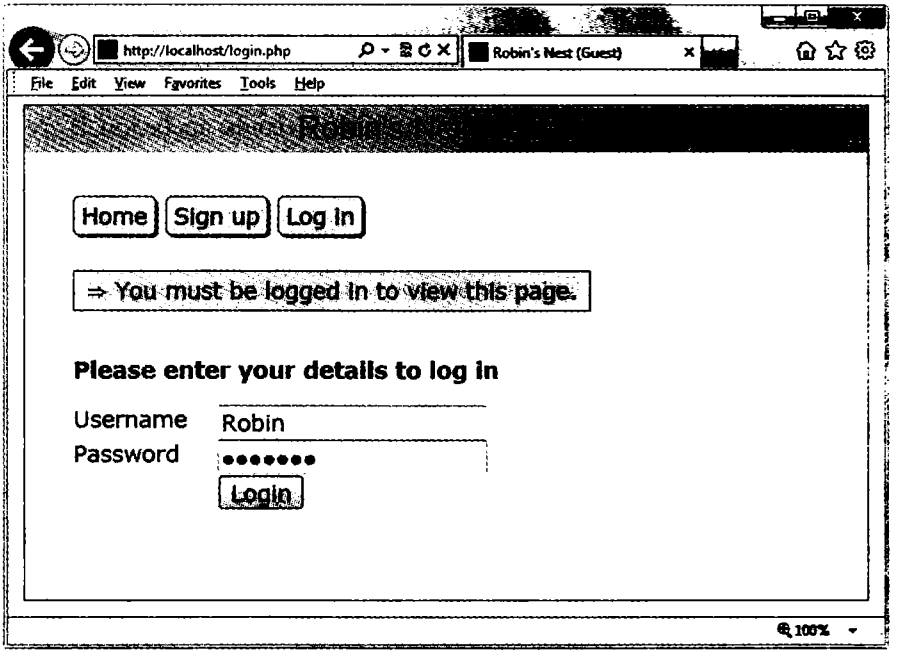



**Рис. 21.3.** Страница входа на сайт

# Файл profile.php

После регистрации и входа на сайт у новых пользователей может появиться желание создать профиль. Это можно будет сделать с помощью файла `profile.php`, код которого показан в примере 21.8. Я думаю, в этом коде можно найти несколько интересных для вас фрагментов, к которым относятся функции загрузки изображений на сайт, изменения их размера и четкости.

Для начала рассмотрим HTML-код, который находится в конце этого файла. Он похож на формы, которые вы только что видели, но на этот раз в форме используется параметр `enctype='multipart/form-data'`. Он позволяет одновременно отправлять более одного типа данных, разрешая отправку на сайт изображений вместе с текстом. В форме также есть элемент `<input />`, имеющий тип `file`, с помощью которого создается кнопка просмотра. Щелкнув на ней, пользователь может выбрать файл для загрузки на сайт.

При отправке данных формы выполняется код, который находится в начале программы. Перед тем как разрешить выполнение программы, этот код проверяет, вошел ли пользователь на сайт. И только после этого отображается заголовок страницы.

## Добавление текста в поле About Me (Обо мне)

Работа продолжается проверкой элемента с ключом `'text'`, принадлежащего массиву, который создан `POST`-запросом. Наличие в нем содержимого свидетельствует о том, что программе отправлен текст. Если текст отправлен, из него удаляются потенциально вредные фрагменты, а все длинные последовательности пробелов (а также символы возврата каретки и перевода строки) заменяются одним пробелом. В эту функцию включена двойная проверка безопасности, гарантирующая, что пользователь с таким именем действительно существует в базе данных и что перед вставкой этого текста в базу данных, где он превратится в сведения о пользователе в поле About Me (Обо мне), не сможет пройти никакая атака со стороны взломщика.

Если текст не отправлен, делается запрос к базе данных на обнаружение уже существующего текста, чтобы заранее заполнить текстовое окно, позволяя тем самым пользователю отредактировать текст.

## Добавление изображения профиля

Теперь перейдем к разделу, проверяющему системную переменную `$_FILES` на наличие загруженного на сайт изображения. Если изображение было загружено, создается строковая переменная `$saveto`. Этой переменной присваивается значение, состоящее из имени пользователя и расширения JPG. Например, для пользователя с именем `Jill` переменной `$saveto` будет присвоено значение `Jill.jpg`. Это значение является именем файла, в котором будет сохранено загруженное изображение, предназначенное для использования в профиле пользователя.

Сразу за этим проверяется тип загруженного изображения, которое принимается только в том случае, если имеет JPEG-, PNG- или GIF-формат. Если тип загруженного изображения не относится к разрешенным, флажок `$typeok` принимает значение `FALSE`, что препятствует выполнению последнего блока кода загрузки изображения. Но если изображение принимается, загруженное изображение присваивается переменной `$src`. Для этого используется одна из функций `imagecreatefrom`, соответствующая типу загруженного изображения. Теперь изображение находится в том формате, который может быть обработан средствами PHP.

## Обработка изображения

Сначала в переменных $w и $h сохраняются размеры изображения, для чего используется следующая инструкция, представляющая собой быстрый способ присваивания значений из массива отдельным переменным:

```
list($w, $h) = getimagesize($saveto);
```

Затем с использованием значения переменной $max (которое установлено в 100) вычисляются новые размеры, которые приведут к созданию нового изображения с таким же соотношением сторон, но размерами, не превышающими 100 пикселов. В результате этого переменные $tw и $th получают новые значения. При желании получить миниатюры меньшего или большего размера нужно просто соответствующим образом изменить значение переменной $max.

После этого вызывается функция imagecreatetruecolor, которая создает новую пустую картинку шириной $tw и высотой $th и сохраняет ее в переменной $tmp. Затем вызывается функция imagecopyresampled, которая изменяет размер изображения, сохраненного в переменной $src, на тот, который хранится в новой переменной $tmp. Иногда изменение размера изображения может привести к небольшой потере резкости получаемой копии, поэтому в следующем фрагменте кода используется функция imageconvolution, слегка повышающая резкость изображения.

И наконец, изображение сохраняется как JPEG-файл в том месте, которое определено значением переменной $saveto, после чего оба изображения — исходное и пустое, имеющее измененные размеры, — удаляются из памяти с помощью функции imagedestroy, возвращая системе занятую под них память.

## Отображение текущего профиля

И последнюю, но не менее важную задачу выполняет функция showProfile из файла functions.php, которая позволяет пользователю посмотреть, как выглядит текущий профиль, перед его редактированием. Эта функция вызывается до отображения формы HTML. Если профиля еще нет, ничего отображаться не будет.

При выводе изображения профиля с помощью CSS создается граница, тень и поле справа, чтобы отделить текст профиля от изображения.

Результат загрузки в браузер файла, код которого содержится в примере 21.8, показан на рис. 21.4. На этом рисунке можно увидеть, что текстовое поле было заранее заполнено текстом **About me** (Обо мне).

**Пример 21.8.** profile.php

```
<?php // profile.php
include_once 'header.php';

if (!$loggedin) die();

echo "<div class='main'><h3>Your Profile</h3>";
                          // Ваш профиль

if (isset($_POST['text']))
```

```
{
    $text = sanitizeString($_POST['text']);
    $text = preg_replace('/\s\s+/', ' ', $text);

    if (mysql_num_rows(queryMysql("SELECT * FROM profiles WHERE user='$user'")))
        queryMysql("UPDATE profiles SET text='$text' where user='$user'");
    else queryMysql("INSERT INTO profiles VALUES('$user', '$text')");
}
else
{
    $result = queryMysql("SELECT * FROM profiles WHERE user='$user'");

    if (mysql_num_rows($result))
    {
        $row  = mysql_fetch_row($result);
        $text = stripslashes($row[1]);
    }
    else $text = "";
}

$text = stripslashes(preg_replace('/\s\s+/', ' ', $text));
if (isset($_FILES['image']['name']))
{
    $saveto = "$user.jpg";
    move_uploaded_file($_FILES['image']['tmp_name'], $saveto);
    $typeok = TRUE;

    switch($_FILES['image']['type'])
    {
     case "image/gif":   $src = imagecreatefromgif($saveto); break;
     case "image/jpeg":
      //Как обычный, так и прогрессивный JPEG-формат
     case "image/pjpeg": $src = imagecreatefromjpeg($saveto); break;
     case "image/png":   $src = imagecreatefrompng($saveto); break;
        default:            $typeok = FALSE; break;
    }

    if ($typeok)
    {
        list($w, $h) = getimagesize($saveto);

        $max = 100;
        $tw  = $w;
        $th  = $h;

        if ($w > $h && $max < $w)
        {
            $th = $max / $w * $h;
            $tw = $max;
        }
        elseif ($h > $w && $max < $h)
```

```
        {
            $tw = $max / $h * $w;
            $th = $max;
        }
        elseif ($max < $w)
        {
            $tw = $th = $max;
        }

        $tmp = imagecreatetruecolor($tw, $th);
        imagecopyresampled($tmp, $src, 0, 0, 0, 0, $tw, $th, $w, $h);
        imageconvolution($tmp, array(array(-1, -1, -1),
            array(-1, 16, -1), array(-1, -1, -1)), 8, 0);
        imagejpeg($tmp, $saveto);
        imagedestroy($tmp);
        imagedestroy($src);
    }
}

showProfile($user);

echo <<<_END
<form method='post' action='profile.php' enctype='multipart/form-data'>
<h3>Enter or edit your details and/or upload an image</h3>
// Введите или отредактируйте сведения и (или) загрузите изображение
<textarea name='text' cols='50' rows='3'>$text</textarea><br />
_END;
?>
```

# Файл members.php

Используя файл members.php, код которого показан в примере 21.9, пользователи сайта смогут найти других участников сети и добавить их в список своих друзей (или удалить их оттуда, если они уже числились друзьями). У этой программы есть два режима: первый показывает пользовательские профили, а второй выдает список всех участников и их отношений к вам.

## Просмотр профилей пользователей

В первом режиме проверяется элемент массива GET-вызова, имеющий ключ 'view'. Если такой элемент существует, значит, пользователь хочет просмотреть чей-то профиль, поэтому программа использует функцию showProfile, а также предоставляет две ссылки для друзей и сообщений пользователей.

## Добавление и удаление друзей

Затем проверяются два элемента массива GET-вызова с ключами 'add' и 'remove'. Если один или другой имеют значение, то оно будет представлять собой имя пользователя,

которого нужно либо добавить в список друзей, либо удалить из этого списка. Для этого выполняется поиск записи пользователя в MySQL-таблице `friends` с последующей вставкой имени пользователя в таблицу или удалением его из этой таблицы.

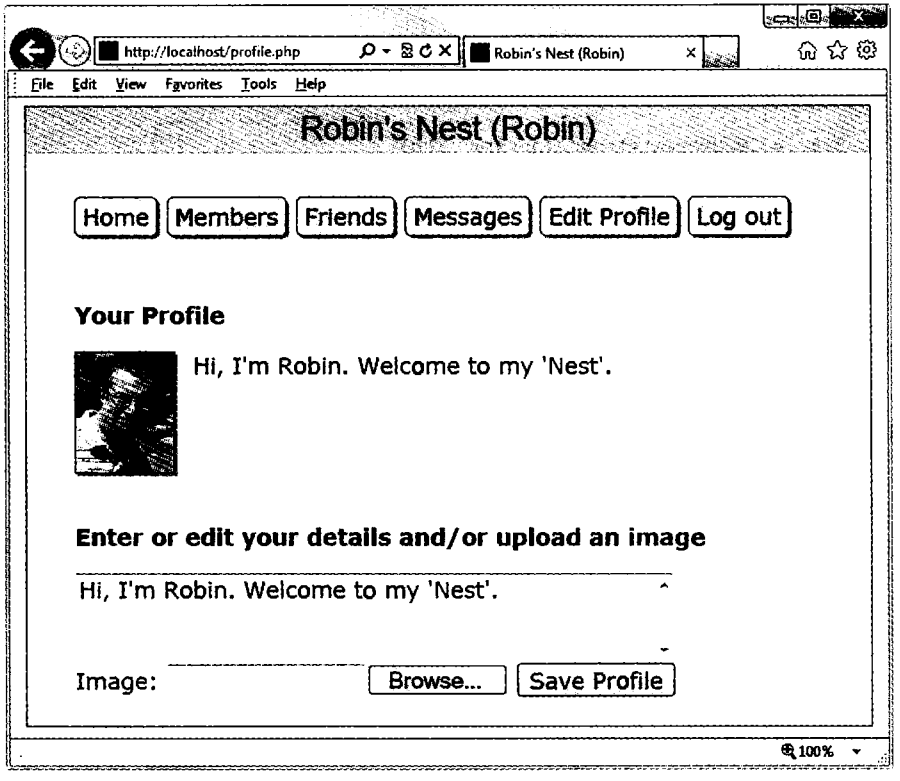



**Рис. 21.4.** Редактирование профиля пользователя

Разумеется, каждая отправленная переменная сначала проходит обезвреживающую обработку, осуществляемую функцией `sanitizeString`, которая призвана обеспечить ее безопасное использование в MySQL.

## Вывод списка всех участников

В последнем блоке кода выдается SQL-запрос на вывод списка всех имен пользователей. Перед выводом заголовка страницы число возвращенных имен присваивается переменной `$num`.

Затем с помощью цикла `for` осуществляется последовательный перебор всех участников с извлечением сведений о них и последующим поиском их в таблице `friends` с целью определения, проявляют ли они интерес к дружбе с пользователем или проявляет ли пользователь свой интерес к дружбе с ними. Если обнаруживается взаимный интерес, такие участники классифицируются как взаимные друзья.

Переменная $t1 имеет ненулевое значение в том случае, когда пользователь проявил интерес к дружбе с другим участником, а переменная $t2 имеет ненулевое значение, когда другой участник заинтересовался дружбой с пользователем. В зависимости от значений этих переменных текст, отображаемый после каждого имени пользователя, показывает степень взаимоотношений его владельца с текущим пользователем, если таковые имеются.

Кроме того, для отображения взаимоотношений используются соответствующие значки. Двунаправленная стрелка означает, что пользователи являются взаимными друзьями. Стрелка влево показывает, что пользователь проявил интерес к дружбе с другим участником. Стрелка вправо показывает, что дружбой с пользователем заинтересовался другой участник.

И наконец, в зависимости от заинтересованности пользователя в дружбе с другим участником ему предоставляется ссылка для добавления этого участника в список друзей или для удаления его оттуда.

При вызове в браузере программы из примера 21.9 будет выведена страница, показанная на рис. 21.5. Обратите внимание на то, как пользователю предлагается заинтересоваться дружбой с тем участником, к которому еще не проявлен интерес (follow), но если участник уже проявил интерес к дружбе с пользователем, то напротив его имени появляется ссылка recip (ответить), позволяющая ответить ему взаимностью. Если пользователь уже заинтересовался дружбой с другим участником, он может выбрать ссылку drop (отклонить) для удаления этой заинтересованности.

**Пример 21.9.** members.php

```
<?php // members.php
include_once 'header.php';

if (!$loggedin) die();

echo "<div class='main'>";

if (isset($_GET['view']))
{
    $view = sanitizeString($_GET['view']);

    if ($view == $user) $name = "Your";
    else                $name = "$view's";

    echo "<h3>$name Profile</h3>";
    showProfile($view);
    echo "<a class='button' href='messages.php?view=$view'>"
         "View $name messages</a><br /><br />";
    die("</div></body></html>");
}

if (isset($_GET['add']))
{
    $add = sanitizeString($_GET['add']);
```

```
    if (!mysql_num_rows(queryMysql("SELECT * FROM friends
        WHERE user='$add' AND friend='$user'")))
        queryMysql("INSERT INTO friends VALUES ('$add', '$user')");
}
elseif (isset($_GET['remove']))
{
    $remove = sanitizeString($_GET['remove']);
    queryMysql("DELETE FROM friends WHERE user='$remove' AND friend='$user'");
}

$result = queryMysql("SELECT user FROM members ORDER BY user");
$num    = mysql_num_rows($result);

echo "<h3>Other Members</h3><ul>"; // Другие участники

for ($j = 0 ; $j < $num ; ++$j)
{
    $row = mysql_fetch_row($result);
    if ($row[0] == $user) continue;

    echo "<li><a href='members.php?view=$row[0]'>$row[0]</a>";
    $follow = "follow"; // Проявить заитересованность в дружбе

    $t1 = mysql_num_rows(queryMysql("SELECT * FROM friends
        WHERE user='$row[0]' AND friend='$user'"));
    $t2 = mysql_num_rows(queryMysql("SELECT * FROM friends
        WHERE user='$user' AND friend='$row[0]'"));

    if (($t1 + $t2) > 1) echo " &harr; is a mutual friend";
      // Двунаправленная стрелка, взаимный друг
    elseif ($t1)         echo " &larr; you are following";
      // Стрелка влево, вы заинтересованы в дружбе
    elseif ($t2)       { echo " &rarr; is following you";
                         $follow = "recip"; }
      // Стрелка вправо, проявляет интерес к дружбе с вами

    if (!$t1) echo " [<a href='members.php?add=".$row[0]    . "'>$follow</a>]";
    else      echo " [<a href='members.php?remove=".$row[0] . "'>drop</a>]";
     // Снять заинтересованность в дружбе
}
?>

<br /></div></body></html>
```

На рабочем сервере могут быть тысячи или даже сотни тысяч пользователей, поэтому, скорее всего, возникнет потребность в существенной модификации этой программы, чтобы включить поддержку поиска в тексте About me (Обо мне), поддержку разбиения выводимой информации на страницы и т. д.

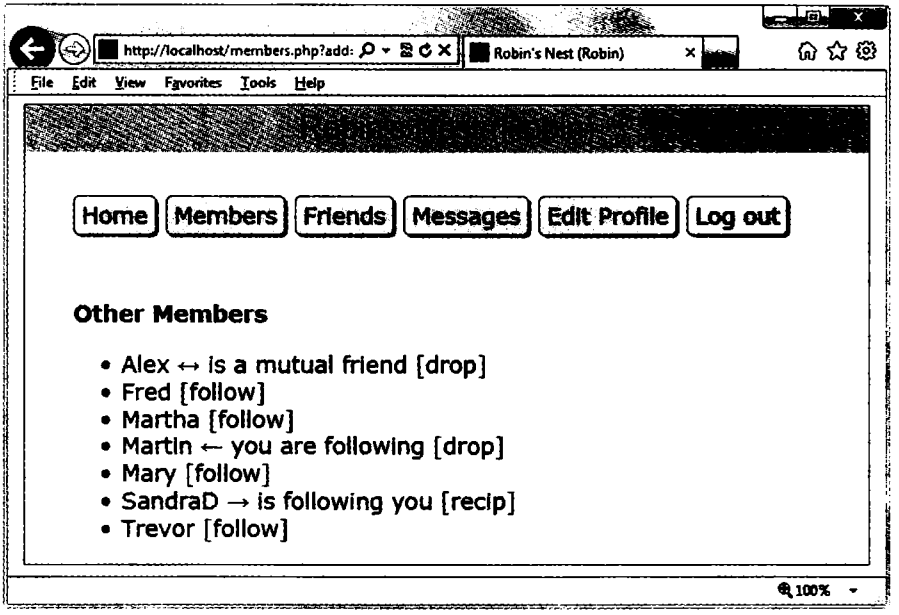



**Рис. 21.5.** Использование модуля участников

# Файл friends.php

Код модуля `friends.php`, показывающий друзей пользователя наряду со степенью проявления интереса к дружбе, приведен в примере 21.10. Этот код выполняет исследование таблицы `friends`, похожее на исследование, которое проводится в программе `members.php`, но только применительно к одному пользователю. Затем этот модуль показывает всех друзей пользователя и проявляющих интерес к дружбе, а также тех участников, к дружбе с которыми пользователь сам проявляет интерес.

Все люди, проявляющие интерес к дружбе, сохраняются в массиве по имени `$followers`, а все люди, к дружбе с которыми проявляется интерес, сохраняются в массиве по имени `$following`. Затем для извлечения всех людей, которые проявляют интерес к дружбе и к которым проявлен взаимный интерес, применяется следующий весьма лаконичный фрагмент кода:

```
$mutual = array_intersect($followers, $following);
```

Функция `array_intersect` извлекает всех участников, являющихся общими для обоих массивов, и возвращает новый массив, который содержит только этих людей. Затем этот массив сохраняется в переменной `$mutual`. Теперь можно воспользоваться функцией `array_diff` для каждого из массивов `$followers` и `$following`, чтобы в них содержались только те люди, которые не являются взаимными друзьями:

```
$followers = array_diff($followers, $mutual);
$following = array_diff($following, $mutual);
```

В результате этого в массиве $mutual будут содержаться только взаимные друзья, в массиве $followers — только люди, проявляющие интерес к дружбе (без взаимных друзей), а в массиве $following — только люди, к дружбе с которыми проявляется интерес (также без взаимных друзей).

При наличии этих массивов упрощается решение задачи отдельного отображения каждой категории участников. Результат этого решения показан на рис. 21.6. Функция PHP sizeof возвращает количество элементов в массиве. Здесь я использую ее только для вызова кода, когда размер массива не является нулевым (то есть друзья данного типа существуют). Обратите внимание на то, как путем использования переменных $name1, $name2 и $name3 в соответствующих местах код может сообщить о том, что вы (пользователь) смотрите собственный список друзей, используя слова *Your* (Ваши) и *You are* (Вы), вместо того чтобы просто отобразить имя пользователя.

Если хотите вывести на экран информацию о профиле пользователя, можно убрать символы комментария из закомментированной строки кода.

**Пример 21.10.** friends.php

```
<?php // friends.php
include_once 'header.php';

if (!$loggedin) die();

if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
else                      $view = $user;

if ($view == $user)
{
    $name1 = $name2 = "Your";    // Ваши
    $name3 =          "You are"; // Вы
}
else
{
    $name1 = "<a href='members.php?view=$view'>$view</a>'s";
    $name2 = "$view's";
    $name3 = "$view is";
}

echo "<div class='main'>";

// Если хотите вывести здесь профиль пользователя, уберите знаки
// комментария из следующей строки
// showProfile($view);
$followers = array();
$following = array();

$result = queryMysql("SELECT * FROM friends WHERE user='$view'");
$num    = mysql_num_rows($result);

for ($j = 0 ; $j < $num ; ++$j)
```

```
{
    $row           = mysql_fetch_row($result);
    $followers[$j] = $row[1];
}

$result = queryMysql("SELECT * FROM friends WHERE friend='$view'");
$num    = mysql_num_rows($result);

for ($j = 0 ; $j < $num ; ++$j)
{
    $row           = mysql_fetch_row($result);
    $following[$j] = $row[0];
}

$mutual    = array_intersect($followers, $following);
$followers = array_diff($followers, $mutual);
$following = array_diff($following, $mutual);
$friends   = FALSE;

if (sizeof($mutual))
{
    echo "<span class='subhead'>$name2 mutual friends</span><ul>";
                          // Взаимные друзья
    foreach($mutual as $friend)
        echo "<li><a href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
}

if (sizeof($followers))
{
    echo "< span class='subhead'>$name2 followers</span><ul>";
            // Интересующиеся в дружбе с...
    foreach($followers as $friend)
        echo "<li><a href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
}

if (sizeof($following))
{
    echo "< span class='subhead'>$name3 following</span><ul>";
              // Заинтересован в дружбе с...
    foreach($following as $friend)
        echo "<li><a href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
}

if (!$friends) echo "<br />You don't have any friends yet.<br /><br />";
       // Пока у вас нет друзей
```

```
echo "<a class='button' href='messages.php?view=$view'>" .
     "View $name2 messages</a>";
?>

</div><br /></body></html>
```

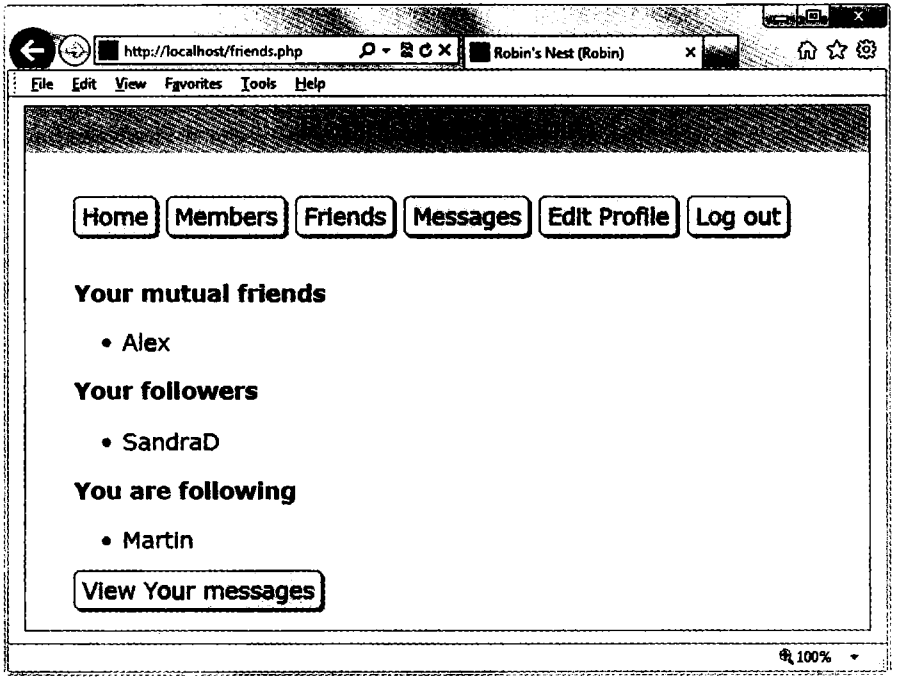



**Рис. 21.6.** Отображение друзей пользователя и заинтересованности во взаимной дружбе

# Файл messages.php

Код последнего из основных модулей, `messages.php`, показан в примере 21.11. Этот модуль начинает работу с проверки наличия отправленного сообщения в элементе POST-массива, имеющего ключ `'text'`. Если сообщение имеется, оно вставляется в таблицу `messages`. Одновременно с этим сохраняется и значение элемента, имеющего ключ `'pm'`. Значение этого элемента свидетельствует об открытом или закрытом статусе сообщения. Нуль представляет открытое, а единица — закрытое сообщение.

Затем отображаются пользовательский профиль и форма для ввода сообщения, а также переключатели для выбора между отправкой закрытого (private) или открытого (public) сообщения. После этого показываются все сообщения: если они имеют статус открытого сообщения, их могут просматривать все пользователи, а закрытые сообщения могут просматривать только отправитель и получатель. Все это управляется двумя запросами к базе данных MySQL. В дополнение к этому, когда сообщение имеет статус закрытого, оно представлено словом whispered (прошептал) и отображается курсивом.

И наконец, программа отображает две ссылки: для обновления сообщений (refresh the messages) в том случае, когда другой пользователь за время просмотра опубликовал новое сообщение, и для просмотра друзей пользователя (view the user's friends). Здесь опять используется прием с переменными $name1 и $name2, чтобы при просмотре вашего собственного профиля вместо имени пользователя отображалось слово *Your* (Ваши).

Результат запуска этой программы в браузере показан на рис. 21.7. Обратите внимание на то, как пользователям, просматривающим собственные сообщения, предоставляется ссылка для того, чтобы можно было стереть (erase) любое из нежелательных сообщений.

**Пример 21.11.** messages.php

```
<?php // messages.php
include_once 'header.php';

if (!$loggedin) die();

if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
else                      $view = $user;

if (isset($_POST['text']))
{
    $text = sanitizeString($_POST['text']);

    if ($text != "")
    {
        $pm   = substr(sanitizeString($_POST['pm']),0,1);
        $time = time();
        queryMysql("INSERT INTO messages VALUES(NULL, '$user',
            '$view', '$pm', $time, '$text')");
    }
}

if ($view != "")
{
    if ($view == $user) $name1 = $name2 = "Your"; // Ваши
    else
    {
        $name1 = "<a href='members.php?view=$view'>$view</a>'s";
        $name2 = "$view's";
    }

    echo "<div class='main'><h3>$name1 Messages</h3>";
                                        // Сообщения
    showProfile($view);

    echo <<<_END
<form method='post' action='messages.php?view=$view'>
Type here to leave a message:<br />
// Наберите здесь текст, чтобы оставить сообщение
```

```
<textarea name='text' cols='40' rows='3'></textarea><br />
Public<input type='radio' name='pm' value='0' checked='checked' />
Private<input type='radio' name='pm' value='1' />
<input type='submit' value='Post Message' /></form><br />
                          // Опубликовать сообщение
_END;

    if (isset($_GET['erase']))
    {
        $erase = sanitizeString($_GET['erase']);
        queryMysql("DELETE FROM messages WHERE id=$erase AND recip='$user'");
    }

    $query  = "SELECT * FROM messages WHERE recip='$view' ORDER BY time DESC";
    $result = queryMysql($query);
    $num    = mysql_num_rows($result);

    for ($j = 0 ; $j < $num ; ++$j)
    {
        $row = mysql_fetch_row($result);

        if ($row[3] == 0 || $row[1] == $user || $row[2] == $user)
        {
            echo date('M jS \'y g:ia:', $row[4]);
            echo " <a href='messages.php?view=$row[1]'>$row[1]</a> ";

            if ($row[3] == 0)
                 echo "wrote: &quot;$row[5]&quot; ";
                    // Сообщил
            else echo "whispered: <span class='whisper'>"
                      "&quot;$row[5]&quot;</span> ";
                    // Прошептал

            if ($row[2] == $user)
                echo "[<a href='messages.php?view=$view" .
                                "&erase=$row[0]'>erase</a>]";
                                            // Стереть

            echo "<br />";
        }
    }
}

if (!$num) echo "<br /><span class='info'>No messages yet</span><br
/><br />";
                              // Пока сообщений нет

echo "<br /><a class='button' href='messages.php?view=$view'>Refresh messages</a>".
                              // Обновить сообщения
     "<a class='button' href='friends.php?view=$view'>View $name2 friends</a>";?>

</div><br /></body></html>
```

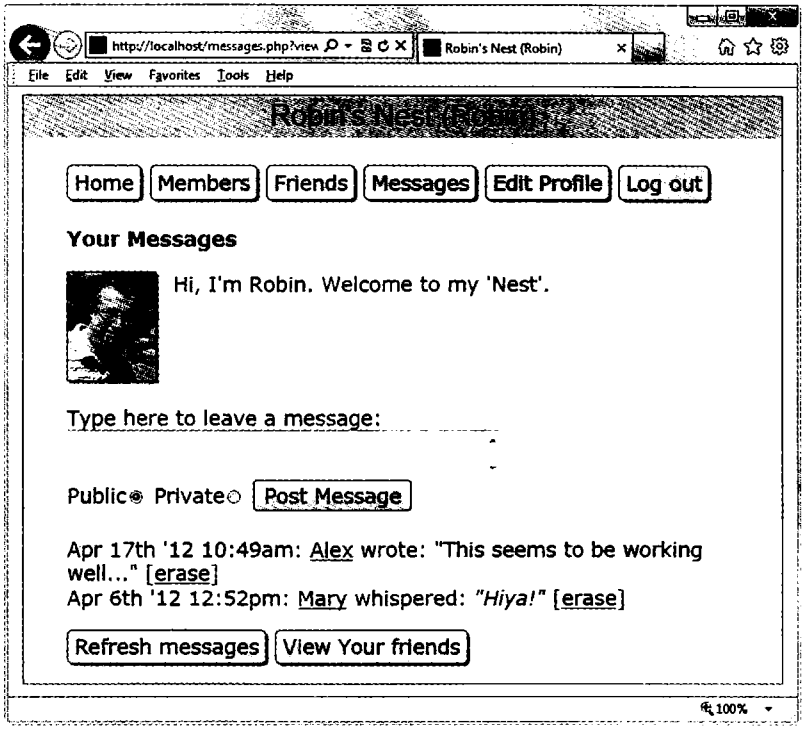



**Рис. 21.7.** Модуль передачи сообщений

# Файл logout.php

Модуль `logout.php`, код которого показан в примере 21.12, является завершающим ингредиентом рецепта нашей социальной сети. Он отображает страницу выхода с сайта, которая закрывает сессию и удаляет любые связанные с ней данные и cookie-файлы. Результат вызова этой программы можно увидеть на рис. 21.8, где на этот раз к пользователю обращена просьба щелкнуть на ссылке. Эта ссылка приведет его на главную страницу, предназначенную для пользователей, еще не вошедших на сайт, с которой удалены все ссылки в верхней части экрана, предназначенные для вошедших на сайт пользователей. Разумеется, вы можете создать код на JavaScript или на PHP, который сразу перенаправил бы пользователя на эту страницу, чтобы очистить страницу выхода из сайта от ненужных элементов.

**Пример 21.12.** `logout.php`

```
<?php // logout.php
include_once 'header.php';

if (isset($_SESSION['user']))
{
    destroySession();
```

```
    echo "<div class='main'>You have been logged out. Please " .
         "<a href='index.php'>click here</a> to refresh the screen.";
                  // Вы уже покинули сайт. Пожалуйста...
                  // ... щелкните здесь, чтобы обновить экран
}                          .
else echo "<div class='main'><br />" .
          "You cannot log out because you are not logged in";
          // Вы не можете завершить сеанс работы,
          // потому что не входили на сайт
?>

<br /><br /></div></body></html>
```

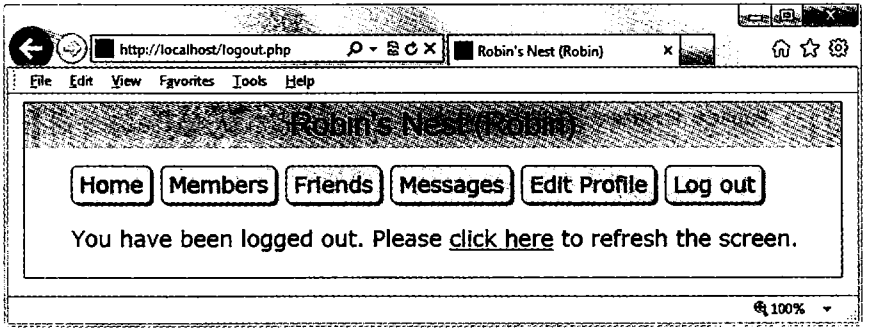



**Рис. 21.8.** Страница выхода с сайта

# Файл styles.css

Таблица стилей, используемая для этого проекта, показана в примере 21.13. В нем содержится несколько наборов объявлений.

- * — с помощью универсального селектора устанавливается используемое в проекте по умолчанию семейство шрифтов и размер шрифта.
- body — задается ширина окна проекта, горизонтальная центровка, указывается цвет фона и задается граница окна.
- html — устанавливается цвет фона блока HTML.
- img — задается граница, тень и правое поле для всех изображений.
- li a, и .button — удаляются подчеркивания у гиперссылок во всех тегах <a>, которые находятся внутри элемента <li>, и у всех элементов, использующих класс button.
- li:ahover и .button:hover — устанавливается цвет отображения текста в элементах <li> и в элементах, использующих класс button, при прохождении над ними указателя мыши.
- .appname — задаются свойства для заголовков, применяющих класс appname, включая центровку, цвета фона и текста, семейство шрифтов и размер шрифта, а также отступы.

- .fieldname — устанавливается ширина элементов, использующих класс fieldname, но сначала для них задается плавающая модель.
- .main — применяется отступ для тех элементов, которые используют этот класс.
- .info — задействуется для отображения важной информации: для ее элементов, использующих этот класс, устанавливаются цвета фона и текста, применяются граница и отступы.
- .menu li, и .button — обеспечивается выстраивание всех элементов <li> и элементов, использующих класс button, в линию, задается наличие у них отступов, границы, цветов фона и первого плана, правого поля, скругленных границ и тени (в результате чего получается эффект кнопки).
- .subhead — выделяется блок текста.
- .taken, .available, .error и .whisper — устанавливаются цвета и стили шрифтов, используемых для отображения различных типов информации.

**Пример 21.13.** Таблица стиля проекта

```
/* styles.css */

* {
    font-family:verdana,sans-serif;
    font-size  :14pt; }

body {
    width     :700px;
    position  :relative;
    margin    :7px auto;
    background:#f8f8f8;
    border    :1px solid #888; }

html {
    background:#fff }

img {
    border            :1px solid black;
    margin-right      :15px;
    -moz-box-shadow   :2px 2px 2px #888;
    -webkit-box-shadow:2px 2px 2px #888;
    box-shadow        :2px 2px 2px #888; }

li a, .button {
    text-decoration:none; }

li a:hover, .button:hover {
    color:green; }

.appname {
    text-align :center;
    background :#eb8;
    color      :#40d;
```

```
    font-family:helvetica;
    font-size  :20pt;
    padding    :4px; }

.fieldname {
    float:left;
    width:120px; }

.main {
    margin-left:40px; }

.info {
    background :lightgreen;
    color      :blue;
    border     :1px solid green;
    padding    :5px 10px;
    margin-left:40px; }

.menu li, .button {
    display           :inline;
    padding           :4px 6px;
    border            :1px solid #777;
    background        :#ddd;
    color             :#d04;
    margin-right      :8px;
    border-radius     :5px;
    -moz-box-shadow   :2px 2px 2px #888;
    -webkit-box-shadow:2px 2px 2px #888;
    box-shadow        :2px 2px 2px #888; }

.subhead {
    font-weight:bold; }

.taken, .error {
    color:red; }

.available {
    color:green; }

.whisper {
    font-style:italic;
    color     :#006600; }
```

Ну вот, собственно, и все. Если вы создадите что-нибудь на основе этого кода или других примеров, приведенных в данной книге, или извлечете пользу из них каким-либо другим образом, я буду рад тому, что мне удалось вам в чем-то помочь. и спасибо за чтение данного издания.

Но перед тем, как закрыть книгу и приступить к практическому применению на просторах Всемирной паутины только что полученных навыков, пожалуйста, просмотрите следующие за этой главой приложения. В них содержится масса дополнительной информации, которая может оказаться полезной для вас.

# Приложение А. Ответы на контрольные вопросы

## Ответы на вопросы главы 1

1. Четырьмя компонентами, необходимыми для создания полностью динамических веб-страниц, являются веб-сервер (такой как Apache), язык сценариев на стороне сервера (PHP), база данных (MySQL) и язык сценариев на стороне клиента (JavaScript).
2. HTML означает язык гипертекстовой разметки — HyperText Markup Language: сама веб-страница, включая текст и команды разметки.
3. Так же как практически все процессоры баз данных, MySQL воспринимает команды на структурированном языке запросов — Structured Query Language (SQL). SQL является способом общения с MySQL любого пользователя (а также PHP-программы).
4. Сценарии PHP работают на сервере, а сценарии JavaScript — на машине клиента. У языка PHP есть средства общения с базой данных, позволяющие хранить и извлекать данные, но его средствами невозможно провести быстрое и динамическое изменение веб-страницы, просматриваемой пользователем. У языка JavaScript имеются совершенно противоположные достоинства и недостатки.
5. CSS означает каскадные таблицы стилей — Cascading Style Sheets: правила задания стилей и разметки, применяемые к элементам HTML- документа.
6. Некоторые из этих технологий подконтрольны компаниям, которые собирают сведения об ошибках и занимаются их устранением, как и любые другие компании, производящие программные продукты. Но программы с открытым кодом зависят также от сообщества разработчиков, поэтому ваш отчет об ошибке может быть рассмотрен любым пользователем, хорошо разбирающимся в коде. Возможно, когда-нибудь и вам доведется исправлять ошибки, допущенные в инструментарии, имеющем открытый код.

# Ответы на вопросы главы 2

1. WAMP означает Windows, Apache, MySQL и PHP, в то время как M в MAMP означает Mac вместо Windows, а L в LAMP — Linux. Все эти аббревиатуры ссылаются на полноценные решения для хостинга динамических веб-страниц.
2. Оба адреса, и 127.0.0.1, и http://localhost, являются способами ссылки на локальный компьютер. При правильной настройке WAMP или MAMP для вызова исходной страницы на локальном сервере в адресную строку браузера можно вводить любой из этих адресов.
3. FTP означает протокол передачи файлов — File Transfer Protocol. Программа FTP используется для передачи файлов между клиентом и сервером в обе стороны.
4. Чтобы обновить файлы, их нужно отправить на удаленный сервер с помощью FTP-программы, что может существенно увеличить время разработки, если это действие повторяется многократно на протяжении одного рабочего сеанса.
5. Специализированные редакторы программ обладают множеством автоматических функций и способны подсвечивать проблемные участки кода даже до его запуска на выполнение.

# Ответы на вопросы главы 3

1. Тег, инициирующий интерпретацию PHP-кода, имеет вид `<?php ... ?>` и может быть сокращен до пары тегов `<? ... ?>`.
2. Для отдельной строки комментария можно воспользоваться сочетанием символов //, а комментарии, занимающие несколько строк, можно заключить в пары символов /* ... */.
3. Все PHP-инструкции должны заканчиваться точкой с запятой (;).
4. Все имена PHP-переменных, исключая имена констант, должны начинаться с символа $.
5. Переменные содержат значения, которые могут быть строкой, числом или другими данными.
6. Выражение `$variable = 1` является инструкцией присваивания, а выражение `$variable == 1` — инструкцией сравнения. Инструкцию `$variable = 1` следует использовать для присваивания значения переменной `$variable`. Инструкцию `$variable == 1` необходимо применять в последующих строках программы для определения того, равно значение переменной `$variable` единице или нет. Если, задумав выполнение сравнения, воспользоваться по ошибке инструкцией `$variable = 1`, скорее всего, произойдут два нежелательных события: переменной `$variable` будет присвоено значение 1 и будет неизменно возвращаться истинное значение, независимо от предыдущего значения этой переменной.

7. Дефисы зарезервированы для операторов вычитания. Если бы в именах переменных было разрешено использование дефисов, это затруднило бы интерпретацию такой конструкции, как `$current-user`, что в любом случае приводило бы к двусмысленности программного кода.
8. Имена переменных чувствительны к регистру букв. Поэтому `$This_Variable` и `$this_variable` являются совершенно разными переменными.
9. Пробелы в именах переменных недопустимы, поскольку они сбьют с толку парсер (синтаксический анализатор) PHP. Вместо них следует использовать знаки подчеркивания (_).
10. Чтобы перевести значение переменной из одного типа в другой, нужно сослаться на эту переменную, и PHP автоматически преобразует ее тип.
11. Разница между `++$j` и `$j++` ни в чем не будет проявляться до тех пор, пока значение переменной `$j` не будет проходить проверку, присваиваться другой переменной или передаваться функции в качестве параметра. В таких случаях использование выражения `++$j` приводит к увеличению значения `$j` на единицу до выполнения проверки или другой инструкции, а применение выражения `$j++` приводит к тому, что сначала выполняется инструкция, а затем значение переменной `$j` увеличивается на единицу.
12. В большинстве случаев, когда не нужно соблюдать приоритетность, операторы `&&` и `and` являются взаимозаменяемыми. При необходимости соблюдения приоритетности у оператора `&&` более высокий уровень приоритета, чем у оператора `and`.
13. Чтобы создать многострочную команду `echo` или инструкцию присваивания, можно воспользоваться кавычками или конструкцией `<<< _END ... _END`.
14. Значения констант нельзя переопределять, потому что по самой их сути, будучи единожды определенными, они сохраняют свое значение до прекращения работы программы.
15. Для изменения исходного предназначения кавычек можно воспользоваться сочетанием `\'` для отключения одинарной кавычки или сочетанием `\"` для отключения двойной кавычки.
16. Команды `echo` и `print` похожи друг на друга, за исключением того, что `print` является функцией PHP и воспринимает один аргумент, а `echo` является конструкцией, которая может воспринимать несколько аргументов.
17. Функции предназначены для выделения отдельных частей кода в самостоятельные изолированные блоки, на которые можно ссылаться по одному лишь имени функции.
18. Объявив переменную глобальной, ее можно сделать доступной для всех частей PHP-программы.
19. Если данные сгенерированы внутри функции, их можно передать всей остальной программе путем возвращения значения с помощью инструкции `return` или изменения значения глобальной переменной.
20. При объединении строки с числом получается еще одна строка.

# Ответы на вопросы главы 4

1. В PHP TRUE представляет значение 1, а FALSE представляет значение NULL, которое можно рассматривать как «ничто» и которое выводится на экран как пустая строка.
2. Самые простые формы выражений – это литералы, к которым относятся числа и строки, и переменные, которые просто вычисляются в самих себя.
3. Разница между унарными, бинарными и трехкомпонентными операторами состоит в количестве необходимых им операндов (им требуется соответственно один, два и три операнда).
4. Наилучший способ установки собственной приоритетности операторов состоит в заключении тех подвыражений, которым нужно придать более высокий уровень приоритета, в круглые скобки.
5. Взаимосвязанность операторов относится к направлению обработки выражения (слева направо или справа налево).
6. Оператор тождественности используется в том случае, когда нужно обойти присущее PHP автоматическое изменение типа операнда (которое называется также приведением типов).
7. К условным инструкциям относятся `if`, `switch` и оператор `?`.
8. Чтобы пропустить выполнение текущей итерации цикла и перейти к следующей итерации, используется инструкция `continue`.
9. Циклы, в которых используется инструкция `for`, считаются более мощными, чем циклы `while`, потому что они поддерживают два дополнительных параметра, которые управляют работой цикла.
10. Большинство условных выражений в инструкциях `if` и `while` являются литералами (или булевыми выражениями), и поэтому они инициируют выполнение кода только в том случае, если вычисляются как TRUE. Числовые выражения инициируют выполнение, когда их значение является ненулевым. Строковые выражения инициируют выполнение, когда вычисляются как непустая строка. Значение NULL вычисляется как ложное, поэтому оно не инициирует выполнение кода.

# Ответы на вопросы главы 5

1. Использование функций избавляет от необходимости многократного копирования или переписывания одних и тех же фрагментов кода, объединяя набор инструкций и позволяя использовать для его вызова простое имя.
2. По умолчанию функции могут возвращать одно значение. Но применение массивов, ссылок и глобальных переменных позволяет возвращать любое количество значений.
3. При обращении к переменной по имени, например при присваивании ее значения другой переменной или передаче ее функции, значение переменной копируется.

При этом изменение значения копии не приводит к изменению исходного значения. Но если вы ссылаетесь на переменную, используется только указатель (или ссылка) на ее значение, поэтому на одно и то же значение ссылается сразу несколько имен. Изменение значения по ссылке приводит к изменению исходного значения.

4. Под областью видимости понимаются части программы, из которых может быть получен доступ к переменной. Например, переменная с глобальной областью видимости может быть доступна из всех частей PHP-программы.
5. Чтобы включить один файл в состав другого файла, можно воспользоваться инструкциями `include` или `require` или их более безопасными вариантами `include_once` и `require_once`.
6. Функция является набором инструкций, на который производится ссылка по имени и который может принимать и возвращать значения. Объект может состоять из нуля или нескольких функций (которые применительно к нему называются методами), а также из переменных (называемых свойствами объекта). Все они объединяются в одно образование.
7. Для создания в PHP нового объекта используется ключевое слово `new`: `$object = new Class`.
8. Для создания подкласса используется ключевое слово `extends`, которое является частью следующей синтаксической конструкции: `class` *Подкласс* `extends` *Родительский_класс*.
9. Чтобы при создании объекта вызвать инициализирующую часть кода, нужно внутри класса создать метод-конструктор `__construct` и поместить в него код.
10. Явного объявления свойств внутри класса не требуется, поскольку они после первого же применения будут объявлены неявным образом. Но явное объявление считается правилом хорошего тона, потому что оно улучшает читаемость кода, упрощает его отладку и приносит особую пользу тем людям, которым приходится обслуживать ваш код.

# Ответы на вопросы главы 6

1. Числовой массив может быть проиндексирован с помощью чисел или числовых переменных. Ассоциативный массив использует для индексации элементов буквенно-цифровые идентификаторы.
2. Основное преимущество от использования ключевого слова `array` состоит в том, что оно дает возможность присваивать массиву сразу несколько значений без повторения имени этого массива.
3. Как функция `each`, так и структура организации цикла `foreach...as` возвращают элементы из массива. Обе они приступают к работе с начала массива и увеличивают значение указателя на единицу, обеспечивая всякий раз возвращение следующего элемента, и обе они возвращают `FALSE` при достижении конца массива. Разница между ними в том, что функция `each` возвращает

только один элемент, поэтому она обычно помещается в цикл. Конструкция foreach...as уже является циклом, выполняемым снова и снова до тех пор, пока не закончатся элементы массива или пока цикл не будет прерван явным образом.

4. Для создания многомерного массива нужно элементам основного массива присвоить значения, представляющие собой дополнительные массивы.
5. Для подсчета количества элементов в массиве может использоваться функция count.
6. Функция explode предназначена для извлечения частей строки, которые разделены идентификатором, например, для извлечения слов, разделенных в предложении пробелами.
7. Чтобы вернуть внутренний указатель текущего элемента массива PHP на первый элемент этого массива, вызывается функция reset.

# Ответы на вопросы главы 7

1. Для отображения числа с плавающей точкой следует использовать спецификатор преобразования %f.
2. Для приема строки Happy Birthday и вывода строки **Happy можно воспользоваться инструкцией printf: printf("%'*7.5s", "Happy Birthday");.
3. Для выдачи информации из printf не в браузер, а в переменную следует воспользоваться альтернативной функцией sprintf.
4. Чтобы создать отметку времени Unix для времени и даты, представленных в виде «7:11am May 2nd, 2016», нужно воспользоваться командой $timestamp = mktime(7, 11, 0, 5, 2, 2016);.
5. Чтобы открыть файл в режиме чтения и записи с усечением его размера и установкой указателя на начало файла, следует в функции fopen воспользоваться режимом доступа к файлу w+.
6. Для удаления файла file.txt следует применить PHP-команду unlink('file.txt');.
7. Для чтения целиком всего файла используется функция file_get_contents. Эта же функция, если ей предоставить URL-адрес, позволяет прочитать файл из Интернета.
8. Сведения о файлах, загруженных на сервер, содержатся в ассоциативном PHP-массиве $_FILES.
9. Системные команды можно запускать с помощью PHP-функции exec.
10. В XHTML 1.0 тег <input type=file name=file size=10> должен быть заменен следующим тегом, в котором используется правильный синтаксис: <input type="file" name="file" size="10" />. Все параметры должны быть заключены в кавычки, а теги, не имеющие закрывающего тега, должны быть закрыты с использованием сочетания />.

# Ответы на вопросы главы 8

1. В MySQL точка с запятой используется для разделения или для завершения команд. Если забыть ввести этот символ, MySQL выдаст приглашение и будет ожидать его ввода.
2. Для просмотра доступных баз данных следует набрать команду `SHOW databases`. Для просмотра таблиц, использующихся в базе данных, следует набрать команду `SHOW tables`. (Эти команды нечувствительны к регистру букв.)
3. Для создания нового пользователя применяется команда `GRANT`:

```
GRANT PRIVILEGES ON newdatabase.* TO 'newuser'
   IDENTIFIED BY 'newpassword';
```

4. Для просмотра структуры таблицы нужно набрать команду `DESCRIBE` *имя_таблицы*;.
5. Индекс в MySQL предназначен для существенного сокращения времени доступа к базе данных за счет поддержки индексов в одной или нескольких ключевых графах, в которых после индексации можно проводить быстрый поиск с целью обнаружения в таблице нужной строки.
6. Индекс `FULLTEXT` позволяется запросам, в которых используется естественный язык, находить ключевые слова, если они имеются в графе или графах, проиндексированных в режиме `FULLTEXT`. Способ поиска во многом похож на тот, для которого используется поисковый механизм.
7. К стоповым относятся слова, имеющие настолько широкое распространение, что нет никакого смысла включать их в индекс `FULLTEXT` или использовать при поиске. Но все же они должны участвовать в поиске, когда входят в состав большой строки, заключенной в кавычки.
8. По сути, спецификатор `SELECT DISTINCT` воздействует только на отображение, выбирая всего лишь одну строку и исключая все ее дубликаты. Спецификатор `GROUP BY` не исключает, а объединяет все строки, у которых есть одно и то же значение в графе. Поэтому спецификатор `GROUP BY` хорошо справляется с осуществлением таких операций, как `COUNT`, ведущей подсчеты в группе строк. А спецификатор `SELECT DISTINCT` для таких целей не годится.
9. Для возвращения только тех строк, в которых в каком-нибудь месте графы `author` таблицы `classics` содержится слово `Langhorne`, используется следующая команда:

```
SELECT * FROM classics WHERE author LIKE "%Langhorne%";
```

10. Для объединения двух таблиц у них должна быть общей хотя бы одна графа, например номер ID или, как в случае с таблицами `classics` и `customers`, графа `isbn`.

# Ответы на вопросы главы 9

1. Под словом *отношение (relationship)* понимается связь между двумя элементами данных, обладающими какими-нибудь взаимными ассоциациями, например:

книга и ее автор или книга и покупатель, который ее приобрел. Реляционная (то есть учитывающая отношения) база данных, например MySQL, специализируется на хранении и извлечении подобных отношений.

2. К процессу удаления повторяющихся данных и оптимизации таблиц применяется термин *нормализация*.
3. Существуют три правила первой нормальной формы: 1) в ней не должно быть никаких повторяющихся граф, содержащих одни и те же типы данных; 2) все графы должны содержать только одно значение; 3) для уникальной идентификации каждой строки должен использоваться первичный ключ.
4. Чтобы таблица отвечала требованиям второй нормальной формы, графы, в которых данные повторяются в нескольких строках, должны быть перемещены в собственные таблицы.
5. При отношении «один ко многим» первичный ключ из таблицы со стороны «один» должен быть добавлен в качестве отдельной графы (внешнего ключа) к таблице на стороне «многие».
6. Для создания базы данных, в которой имеется отношение «многие ко многим», нужно создать промежуточную таблицу, содержащую ключи из двух других таблиц. В результате эти две другие таблицы смогут ссылаться друг на друга посредством третьей таблицы.
7. Чтобы инициировать MySQL-транзакцию, используется либо команда `BEGIN`, либо команда `START TRANSACTION`. Для прекращения транзакции и отмены всех действий выдается команда `ROLLBACK`. Для завершения транзакции и совершения всех действий выдается команда `COMMIT`.
8. Для изучения подробностей работы запроса можно воспользоваться командой `EXPLAIN`.
9. Для создания в файле `publications.sql` резервной копии базы данных `publications` можно воспользоваться следующей командой:

```
mysqldump -u user -ppassword publications > publications.sql
```

# Ответы на вопросы главы 10

1. Для подключения к базе данных MySQL используется стандартная PHP-функция `mysql_connect`.
2. Работу функции `mysql_result` нельзя признать оптимальной в том случае, если запрашивается более одной ячейки, поскольку она извлекает из базы данных только одну ячейку и поэтому должна вызываться многократно, в то время как функция `mysql_fetch_row` извлекает сразу всю строку.
3. Как правило, метод `POST` больше подходит для отправки данных формы, чем метод `GET`, поскольку поля отправляются непосредственным образом, а не добавляются к URL-адресу. У этого метода есть несколько преимуществ, особенно в том, что он не позволяет вставлять поддельные данные в адресную строку браузера. (И тем не менее он не обеспечивает полноценной защиты от вставки поддельных данных.)

4. Для определения последнего значения, введенного в столбец со свойством AUTO_INCREMENT, нужно воспользоваться функцией mysql_insert_id.
5. Для отключения опасных символов в строке и подготовки ее к использованию в MySQL применяется PHP-функция mysql_real_escape_string.
6. Атака внедрения при межсайтовом скриптинге может быть предотвращена с помощью функции htmlentities.

# Ответы на вопросы главы 11

1. Для передачи данных формы PHP-программе GET-методом используется массив $_GET, а POST-методом — массив $_POST.
2. Свойство register_globals включалось по умолчанию в версиях PHP, предшествующих версии 4.2.0. Недостаток такой настройки заключался в том, что отправленные данные полей формы автоматически присваивались в качестве значений PHP-переменным, открывая лазейку в системе безопасности для потенциальных взломщиков, которые могли попытаться внедрить в PHP-код свои вставки, инициализируя переменные по своему усмотрению.
3. Несмотря на то что оба этих поля при заполнении формы воспринимают текст, разница между ними заключается в том, что в текстовом поле может содержаться только одна строка, а в текстовой области может быть несколько строк и в ней осуществляется перенос слов на новую строку.
4. Для предоставления пользователю возможности выбора в веб-форме трех взаимоисключающих вариантов следует воспользоваться переключателями, поскольку флажки допускают множественный выбор.
5. Для отправки из веб-формы группы значений при использовании только одного имени поля следует воспользоваться не обычным именем поля, а именем массива с квадратными скобками, например choices[]. Тогда каждое значение помещается в массив, длина которого будет соответствовать количеству отправленных элементов.
6. Чтобы отправить данные поля формы, не отображая их на экране браузера, следует поместить эти данные в скрытое поле, в котором используется параметр type="hidden".
7. Элемент формы, а также сопровождающий его текст или графику можно заключить в теги <label> и </label> и гарантировать тем самым возможность выбора щелчком кнопки мыши на всей области объединенного элемента.
8. Чтобы превратить HTML в такой формат, который может быть отображен на экране, но не сможет интерпретироваться браузером как код HTML, используется PHP-функция htmlentities.

# Ответы на вопросы главы 12

1. Cookie должны быть переданы до кода HTML веб-страницы, потому что они отсылаются в виде составной части заголовков.

2. Для сохранения cookie на машине веб-браузера применяется функция `set_cookie`.
3. Для удаления cookie нужно его выдать заново, но при этом установить срок истечения действия на уже прошедшее время.
4. При использовании HTTP-аутентификации имя пользователя и пароль сохраняются в элементах массива `$_SERVER['PHP_AUTH_USER']` и `$_SERVER['PHP_AUTH_PW']`.
5. Функция `md5` считается мощным средством защиты, потому что это односторонняя функция, превращающая строку в 32-символьное шестнадцатеричное число, не поддающееся обратному преобразованию, из-за чего установить прежнее значение строки практически невозможно.
6. При подмешивании произвольных данных, прежде чем преобразовать строку с помощью функции `md5`, в нее добавляются лишние символы (известные только программисту). Благодаря этому лобовая словарная атака практически обречена на провал.
7. PHP-сессия — это группа уникальных для текущего пользователя переменных.
8. Чтобы инициировать PHP-сессию, используется функция `session_start`.
9. Хищение сессии происходит в том случае, когда взломщик каким-то образом добывает ID существующей сессии и пытается ее захватить.
10. Фиксация сессии заключается в попытке принудительного навязывания серверу ID сессии, не позволяя ему создавать собственный идентификатор.

# Ответы на вопросы главы 13

1. В качестве контейнера для кода JavaScript используются теги `<script>` и `</script>`.
2. По умолчанию информация, выводимая кодом JavaScript, будет добавлена к той части документа, в которой находится этот код. Если он находится в заголовке, она будет добавлена в заголовок, а если в теле документа, то в тело.
3. Код JavaScript, принадлежащий другому источнику, может быть включен в ваш документ либо путем копирования и последующей вставки этого кода, либо более распространенным способом — включением его в качестве составляющей тега `<script src='filename.js'>`.
4. В JavaScript эквивалентом PHP-команд `echo` и `print` служит функция (или метод) `document.write`.
5. Чтобы создать комментарий в JavaScript, нужно перед текстом однострочного комментария поставить символы `//`, а многострочный комментарий поместить между символами `/*` и `*/`.
6. Оператором объединения строк в JavaScript служит знак «плюс» (+).
7. Внутри функции JavaScript можно определить переменную, имеющую локальную область видимости, поставив в начале первой инструкции присваивания значения этой переменной ключевое слово `var`.

8. Чтобы во всех основных браузерах отобразить URL-адрес, присвоенный ссылке, имеющей ID со значением thislink, можно воспользоваться двумя следующими командами:

```
document.write(document.getElementById('thislink').href)
document.write(thislink.href)
```

9. Команды для загрузки предыдущей страницы, ссылка на которую хранится в истории браузера, имеют следующий вид:

```
history.back()
history.go(-1)
```

10. Для замены текущего документа главной страницей веб-сайта oreilly.com можно воспользоваться следующей командой:

```
document.location.href = 'http://oreilly.com'
```

# Ответы на вопросы главы 14

1. Наиболее заметной разницей между булевыми значениями в PHP и в JavaScript является то, что PHP распознает ключевые слова TRUE, true, FALSE и false, а в JavaScript поддерживаются только ключевые слова true и false. Кроме того, в PHP TRUE имеет значение 1, а FALSE имеет значение NULL, а в JavaScript они представлены значениями true и false, которые могут быть возвращены в виде строчных значений.
2. Разница между унарными, бинарными и трехкомпонентными операторами состоит в количестве необходимых для них операндов (им требуется соответственно один, два и три операнда).
3. Наилучший способ установки собственной приоритетности операторов состоит в заключении части выражения, которая должна быть вычислена в первую очередь, в круглые скобки.
4. Оператор тождественности используется в тех случаях, когда нужно обойти присущее JavaScript автоматическое изменение типа операнда.
5. Самыми простыми формами выражений являются литералы (такие как числа и строки) и переменные, которые просто вычисляются в самих себя.
6. К трем условным инструкциям относятся if, switch и оператор ?.
7. Большинство условных выражений в инструкциях if и while являются литералами (или булевыми выражениями), и поэтому они инициируют выполнение кода только в том случае, если вычисляются как true. Числовые выражения инициируют выполнение, когда результатом их вычисления является ненулевое значение. Строковые выражения инициируют выполнение, когда вычисляются как непустая строка. Значение NULL вычисляется как ложное, поэтому оно не инициирует выполнение кода.
8. Циклы, в которых используется инструкция for, считаются более мощными по сравнению с циклами while, потому что они поддерживают два дополнительных параметра, которые управляют работой цикла.

9. Инструкция with воспринимает в качестве своего параметра объект. При ее использовании объект указывается только один раз, а затем внутри блока устанавливается связь этого объекта с каждой инструкцией.
10. Чтобы изящно обрабатывать ошибки, нужно воспользоваться функцией try, которая отправит любую возникшую ошибку соответствующей функции catch, где вы можете устранить ошибку или предоставить альтернативный код. Код можно также привязать к событию onerror.

# Ответы на вопросы главы 15

1. Имена функций и переменных в JavaScript чувствительны к регистру используемых в них букв. Имена Count, count и COUNT представляют совершенно разные переменные.
2. Для создания функции, которая воспринимает и обрабатывает неограниченное число параметров, доступ к параметрам организуется через массив arguments, являющийся составной частью всех функций.
3. Один из способов возвращения из функции нескольких значений заключается в помещении всех этих значений в массив с последующим его возвращением.
4. При определении класса для ссылки на текущий объект используется ключевое слово this.
5. Методы класса не обязательно должны определяться внутри определения самого класса. Если метод класса определяется за пределами конструктора, имя этого метода должно быть присвоено объекту this внутри определения класса.
6. Новый объект создается с помощью ключевого слова new.
7. Доступность свойства или метода может быть обеспечена всем объектам класса без тиражирования этого свойства или метода внутри объекта путем использования ключевого слова prototype для создания единственного экземпляра, который затем передается по ссылке всем объектам класса.
8. Для создания многомерного массива внутри основного массива помещается подмассив.
9. Синтаксис, который следует использовать для создания ассоциативного массива, имеет структуру ключ : значение, заключенную в фигурные скобки, как показано в следующем примере:

```
assocarray = {"forename"   "Paul", "surname"   "McCartney",
"group" : "Beatles"}
```

10. Инструкция, используемая для сортировки в убывающем порядке массива, состоящего из чисел, будет иметь следующий вид:

```
numbers.sort(function(a,b){return b   a})
```

# Ответы на вопросы главы 16

1. Отправить данные формы на проверку до их отправки на сервер можно добавлением к тегу `<form>` JavaScript-метода `onSubmit`. Функция проверки должна возвращать `true`, если форма должна быть отправлена на сервер, или `false`, если она не прошла проверку.
2. Для проверки соответствия строки регулярному выражению в JavaScript используется метод `test`.
3. Регулярные выражения, соответствующие символам, не использующимся в словах, могут иметь вид `/[^\w]/`, `/[\W]/`, `/[^a-zA-Z0-9_]/` и т. д.
4. Для проверки соответствия как слову fox , так и слову fix можно воспользоваться регулярным выражением `/f[oi]x/`.
5. Регулярное выражение, соответствующее любому отдельному слову, за которым следует символ, не использующийся в словах, может иметь следующий вид: `/\w+\W/g`.
6. Функция JavaScript, использующая регулярное выражение для проверки наличия слова fox в строке The quick brown fox, может иметь следующий вид:

```
document.write(/fox/.test("The quick brown fox"))
```

7. Функция PHP, использующая регулярное выражение для замены всех экземпляров слова the в строке The cow jumps over the moon словом my, может иметь следующий вид:

```
$s=preg_replace("/the/i", "my", "The cow jumps over the moon");
```

8. Для предварительного заполнения полей формы значениями используется ключевое слово HTML `value`, которое помещается в тег `<input>` и принимает там форму `value="`*значение*`"`.

# Ответы на вопросы главы 17

1. Необходимость создания функции для создания новых объектов `XMLHTTPRequest` обусловлена тем, что браузеры Microsoft используют два разных метода их создания, а все остальные основные браузеры применяют третий, совершенно иной метод. За счет создания функции, тестирующей используемый браузер, можно обеспечить работу кода на всех основных браузерах.
2. Цель применения конструкции `try...catch` состоит в настройке на перехват ошибки при выполнении кода, находящегося внутри инструкции `try`. Если его выполнение вызовет ошибку, то вместо выдачи общей ошибки будет выполнен код блока `catch`.
3. У объекта `XMLHTTPRequest` имеется шесть свойств и шесть методов (см. табл. 17.1 и 17.2).
4. Определить завершение Ajax-вызова можно по значению 4, которое примет свойство `readyState`.

5. Когда Ajax-вызов успешно завершится, принадлежащее объекту свойство status получит значение 200.
6. Значение, возвращенное успешно завершенным Ajax-вызовом, содержится в свойстве responseText объекта XMLHTTPRequest.
7. DOM-дерево, созданное из XML, возвращенного успешно завершенным Ajax-вызовом, содержится в свойстве responseXML объекта XMLHTTPRequest.
8. Для указания функции обратного вызова, обрабатывающей Ajax-ответы, нужно присвоить имя функции свойству onreadystatechange объекта XMLHTTPRequest. Можно также воспользоваться безымянной встроенной функцией.
9. Для инициирования Ajax-запроса вызывается метод send объекта XMLHTTPRequest.
10. Основное различие между Ajax GET- и POST-запросами состоит в том, что GET-запросы присоединяют данные к URL-адресу, а POST-запросы передают данные в качестве параметра метода send и требуют правильной формы заголовков, отправляемых в первоочередном порядке.

# Ответы на вопросы главы 18

1. Чтобы импортировать одну таблицу стилей в другую, используется инструкция @import, например: `@import url('styles.css');`.
2. Чтобы импортировать таблицу стилей в документ, можно воспользоваться HTML-тегом <link />, например: `<link rel='stylesheet' type='text/css' href='styles.css' />`.
3. Чтобы непосредственно встроить стиль в элемент, используется атрибут style, например: `<div style='color:blue;'>`.
4. Разница между идентификатором CSS и классом CSS заключается в том, что идентификатор применяется только к одному элементу, а класс можно применить ко многим элементам.
5. В CSS-объявлениях для имен идентификаторов в качестве префикса используется символ решетки (#), а для имен классов применяется символ точки (.), например #myid и .myclass.
6. Точка с запятой в CSS используется в качестве разделителя объявлений.
7. Чтобы добавить к таблице стилей комментарий, его нужно поставить между маркерами открытия и закрытия комментария, /* и */.
8. В CSS указать на соответствие любому элементу можно с помощью универсального селектора *.
9. Чтобы выбрать в CSS группу разных элементов и (или) типов элементов, между каждым элементом, идентификатором, классом и т. д. нужно ставить запятую.
10. Чтобы одно CSS-объявление из пары, имеющей одинаковые приоритеты, получило преимущество над другим, к нему добавляется объявление !important, например: `p{ color:#ff0000 !important; }`.

# Ответы на вопросы главы 19

1. CSS3-операторы ^, $ и * в указанном порядке соответствуют началу, концу и любой части строки.
2. Для указания размера фонового изображения используется свойство background-size, например: `background-size:800px 600px;`.
3. Радиус границы можно указать с помощью свойства `border-radius`, например: `border-radius:20px;`.
4. Перетекание текста по нескольким колонкам можно задать с помощью свойств `column-count`, `column-gap` и `column-rule` или их вариантов, характерных для того или иного браузера, например: `column-count: 3; column-gap:1em; column-rule:1px solid black;`.
5. Четырьмя функциями, с помощью которых можно указать CSS-цвета, являются `hsl`, `hsla`, `rgb` и `rgba`. Например: `color:rgba(0%,60%,40%,0.4);`.
6. Чтобы создать серую тень под каким-нибудь текстом, с диагональным отступом вправо и вниз на 5 пикселов и с размытостью 3 пиксела, можно воспользоваться следующим объявлением: `text-shadow: 5px 5px 3px #888;`.
7. Показать с помощью многоточия, что текст усечен, можно с помощью следующего объявления: `text-overflow:ellipsis;`.
8. Чтобы включить в состав своей веб-страницы веб-шрифты Google, нужно сначала их выбрать на веб-сайте **http://google.com/webfonts**. Затем, если выбран, предположим, шрифт Lobster, он включается в тег `<link>`, например: `<link href='http://fonts.googleapis.com/css?family=Lobster' />`. Можно также сослаться на шрифт в CSS-объявлении, например в таком: `h1{ font-family:'Lobster', arial, serif; }`.
9. Для поворота объекта на 90° нужно воспользоваться CSS-объявлением `transform: rotate(90deg);`.
10. Чтобы указать переход объекта таким образом, чтобы при изменении любого из его свойств переход осуществлялся сразу в линейном режиме в течение половины секунды, нужно воспользоваться следующим объявлением: `transition: all .5s linear;`.

# Ответы на вопросы главы 20

1. Функция `O` возвращает объект по его идентификатору, функция `S` возвращает свойство стиля объекта, а функция `C` возвращает массив всех объектов, к которым обращается заданный класс.
2. Изменить CSS-атрибут объекта можно с помощью функции `setAttribute`, например: `myobject.setAttribute('font-size', '16pt')`. Можно также изменить атрибут непосредственно (что обычно и делается), используя немного измененные имена там, где это требуется, например: `myobject.fontSize = '16pt'`. (Следует напомнить, что в JavaScript дефис зарезервирован для использования в качестве математического оператора, поэтому при обращении к CSS-свойству, в имени которого

используется дефис, этот дефис нужно опустить, а символ, за которым он следовал, указать в верхнем регистре.)

3. Свойствами, предоставляющими доступную в окне браузера ширину и высоту, являются `window.innerHeight` и `window.innerWidth`.
4. Задать какие-нибудь действия при прохождении указателя мыши над объектом, а затем при выходе за границы объекта можно, привязав код, совершающий эти действия к событиям `onmouseover` и `onmouseout`.
5. Для создания нового элемента нужно воспользоваться таким кодом, как, например, `elem = document.createElement('span')`, а для добавления нового элемента к DOM применяется такой код, как, например, `document.body.appendChild(elem)`.
6. Чтобы сделать элемент невидимым, нужно установить для его свойства `visibility` значение `'hidden'` (а для возвращения ему видимости нужно задать значение `'visible'`). Чтобы сжать элемент до нулевых размеров, следует выбрать для его свойства `display` значение `'none'` (а для восстановления его размеров — значение `'block'`).
7. Чтобы задать одиночное событие в будущем времени, нужно вызвать функцию `setTimeout`, передав ей код или имя функции для выполнения, и значение задержки в миллисекундах.
8. Для установки повторяющегося события через указанный интервал времени нужно вызвать функцию `setInterval`, передав ей код или имя функции для выполнения, и значение задержки между повторениями — в миллисекундах.
9. Чтобы освободить элемент от его места на веб-странице, позволив ему перемещаться, нужно установить для его свойства `position` значение `'relative'`, `'absolute'` или `'fixed'`. Для восстановления элемента на его исходном месте значение этого свойства следует задать в `'static'`.
10. Для получения скорости анимации 50 кадров в секунду нужно установить задержку между прерываниями равной 20 миллисекундам. Для вычисления этого значения необходимо разделить 1000 миллисекунд на желаемую скорость анимации.

# Приложение Б. Интернет-ресурсы

В этом приложении перечислены полезные веб-сайты, предоставляющие материал, использованный в данной книге, или другие ресурсы для совершенствования ваших веб-программ.

## Веб-сайты, относящиеся к PHP

- http://codewalkers.com.
- http://developer.yahoo.com/php/.
- http://easyphp.org.
- http://forums.devshed.com.
- http://free-php.net.
- http://hotscripts.com/category/php/.
- http://htmlgoodies.com/beyond/php/.
- http://php.net.
- http://php.resourceindex.com.
- http://php-editors.com.
- http://phpbuilder.com.
- http://phpfreaks.com.
- http://phpunit.de.
- http://w3schools.com/php/.
- http://zend.com.

## Веб-сайты, относящиеся к MySQL

- http://code.google.com/edu/tools101/mysql.html.
- http://launchpad.net/mysql/.

- http://mysql.com.
- http://php.net/mysql.
- http://planetmysql.org.
- http://sun.com/software/products/mysql/.
- http://sun.com/systems/solutions/mysql/resources.jsp.
- http://w3schools.com/PHP/php_mysql_intro.asp.

# Веб-сайты, относящиеся к JavaScript

- http://developer.mozilla.org/en/JavaScript.
- http://dynamicdrive.com.
- http://javascript.about.com.
- http://javascript.internet.com.
- http://javascript.com.
- http://javascriptkit.com.
- http://w3schools.com/JS/.
- http://www.webreference.com/js/.

# Веб-сайты, относящиеся к Ajax

- http://ajax.asp.net.
- http://ajaxian.com.
- http://ajaxmatters.com.
- http://developer.mozilla.org/en/AJAX.
- http://developer.yahoo.com/yui/.
- http://dojotoolkit.org.
- http://jquery.com.
- http://mochikit.com.
- http://mootools.net.
- http://openjs.com.
- http://prototypejs.org.
- http://sourceforge.net/projects/clean-ajax.
- http://w3schools.com/Ajax/.

# Веб-сайты с разнообразными ресурсами

- http://apachefriends.org.
- http://easyphp.org.

- http://eclipse.org.
- http://editra.org.
- http://fireftp.mozdev.org.
- http://sourceforge.net/projects/glossword/.
- http://mamp.info/en/.
- http://pear.php.net.
- http://programmingforums.org.
- http://putty.org.
- http://smarty.net.
- http://wampserver.com/en/.

# Веб-сайты с ресурсами издательства O'Reilly

- http://onlamp.com.
- http://onlamp.com/php/.
- http://onlamp.com/onlamp/general/mysql.csp.
- http://oreilly.com/ajax/.
- http://oreilly.com/javascript/.
- http://oreilly.com/mysql/.
- http://oreilly.com/php/.
- http://oreillynet.com/javascript/.

# Приложение B. MySQL's FULLTEXT Stopwords

В этом приложении содержится более 500 стоповых слов (stopwords), упоминавшихся в пункте «Создание индекса FULLTEXT» раздела «Индексы» главы 8. Стоповыми называются слова, которые считаются настолько распространенными, что не могут представлять ценности для поиска или хранения в индексе FULLTEXT. Теоретически игнорирование этих слов слабо влияет на результаты большинства поисков в режиме FULLTEXT, но позволяет существенно сократить объем и повысить эффективность работы баз данных MySQL. Здесь приведены слова в нижнем регистре, но этот перечень касается также слов, представленных в верхнем и смешанном регистрах.

**A**

A's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, asking, associated, at, available, away, awfully.

**B**

Be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by.

**C**

C'mon, c's, came, can, can't, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, couldn't, course, currently.

**D**

Definitely, described, despite, did, didn't, different, do, does, doesn't, doing, don't, done, down, downwards, during.

**E**

Each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except.

**F**

Far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore.

**G**

Get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings.

**H**

Had, hadn't, happens, hardly, has, hasn't, have, haven't, having, he, he's, hello, help, hence, her, here, here's, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however.

**I**

I'd, i'll, i'm, i've, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, isn't, it, it'd, it'll, it's, its, itself.

**J**

Just.

**K**

Keep, keeps, kept, know, knows, known.

**L**

Last, lately, later, latter, latterly, least, less, lest, let, let's, like, liked, likely, little, look, looking, looks, ltd.

**M**

Mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself.

**N**

Name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere.

**O**

Obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own.

**P**

Particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides.

**Q**

Que, quite, qv.

**R**

Rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right.

**S**

Said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, shouldn't, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure.

**T**

T's, take, taken, tell, tends, th, than, thank, thanks, thanx, that, that's, thats, the, their, theirs, them, themselves, then, thence, there, there's, thereafter, thereby, therefore, therein, theres, thereupon, these, they, they'd, they'll, they're, they've, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two.

**U**

Un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually.

**V**

Value, various, very, via, viz, vs.

**W**

Want, wants, was, wasn't, way, we, we'd, we'll, we're, we've, welcome, well, went, were, weren't, what, what's, whatever, when, whence, whenever, where, where's, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, who's, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, won't, wonder, would, would, wouldn't.

**Y**

Yes, yet, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves.

**Z**

Zero.

# Приложение Г. Функции MySQL

За счет функций, встроенных в MySQL, существенно сокращается время выполнения сложных запросов и упрощается их конструкция. Если есть желание получить более полную информацию обо всех доступных функциях, можно обратиться к материалам веб-сайтов по следующим URL-адресам:

- строковые функции: http://dev.mysql.com/doc/refman/5.0/en/string-functions.html;
- функции даты и времени: http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html.

Но для ускорения далее приводятся описания наиболее востребованных функций MySQL.

## Строковые функции

### CONCAT()

```
CONCAT(str1, str2, ...)
```

Возвращает результат объединения *str1*, *str2* и любых других параметров (или NULL, если все аргументы имеют значение NULL). Если какой-нибудь из аргументов имеет двоичную форму (binary), то результат будет возвращен в виде двоичной последовательности, в противном случае результат будет в виде строки, не имеющей двоичного формата. Следующий код возвращает строку MySQL:

```
SELECT CONCAT('My', 'S', 'QL');
```

### CONCAT_WS()

```
CONCAT_WS(separator, str1, str2, ...)
```

Эта функция работает так же, как и CONCAT, за исключением того, что она между объединяемыми элементами вставляет разделитель *separator*. Если разделитель имеет значение NULL, то результат тоже будет NULL, но значения NULL могут использоваться

и в качестве других аргументов, которые в таком случае будут пропущены. Следующий код возвращает строку Truman,Harry,S:

```
SELECT CONCAT_WS(',' 'Truman', 'Harry', 'S');
```

## LEFT()

LEFT(*str*, *len*)

Возвращает *len* самых левых символов из строки *str* (или NULL, если какой-нибудь из аргументов имеет значение NULL). Следующий код возвращает строку Chris:

```
SELECT LEFT('Christopher Columbus', '5');
```

## RIGHT()

RIGHT(*str*, *len*)

Возвращает *len* самых правых символов из строки *str* (или NULL, если какой-нибудь из аргументов имеет значение NULL). Следующий код возвращает строку Columbus:

```
SELECT RIGHT('Christopher Columbus', '8');
```

## MID()

MID(*str*, *pos*, *len*)

Возвращает до *len* символов из строки *str*, начиная с позиции *pos*. Если аргумент *len* опущен, то возвращаются все символы до конца строки. Для аргумента *pos* можно использовать отрицательное значение, тогда он будет представлять позицию символа, вычисляемую с конца строки. Первой позицией в строке является 1. Следующий код возвращает строку stop:

```
SELECT MID('Christopher Columbus', '6', '4');
```

## LENGTH()

LENGTH(*str*)

Возвращает длину строки *str* в байтах. Имейте в виду, что при встрече многобайтовых символов учитываются все их байты. Если нужно узнать число символов в строке, следует воспользоваться функцией CHAR_LENGTH. Следующий код возвращает значение 15:

```
SELECT LENGTH('Mark Zuckerberg');
```

## LPAD()

LPAD(*str*, *len*, *padstr*)

Возвращает строку *str*, дополненную до длины *len* символами *padstr*, добавляемыми в начало строки. Если строка *str* длиннее, чем *len*, то строка возвращается усеченной до *len* символов. Этот код:

```
SELECT LPAD('January', '8', ' ');
SELECT LPAD('February', '8', ' ');
SELECT LPAD('March', '8', ' ');
SELECT LPAD('April', '8', ' ');
SELECT LPAD('May', '8', ' ');
```

вернет следующие строки:

```
 January
February
   March
   April
     May
```

Обратите внимание на то, как все строки были дополнены до восьми символов.

## RPAD

```
RPAD(str, len, padstr)
```

Эта функция работает так же, как и функция LPAD, за исключением того, что она возвращает строку, дополненную символами не слева, а справа. Следующий код возвращает строку Hi!!!:

```
SELECT RPAD('Hi', '5', '!');
```

## LOCATE()

```
LOCATE(substr, str, pos)
```

Возвращает позицию первой же встреченной подстроки *substr* в строке *str*. Если функции передан параметр *pos*, то поиск начинается с позиции *pos*. Если *substr* не была найдена в строке *str*, возвращается значение 0.

Следующий код возвращает значения 5 и 11, поскольку при вызове первой функции возвращается позиция первого встреченного слова unit, а вторая функция начинает поиск только с седьмого символа и поэтому возвращает позицию второго появления этого слова в строке:

```
SELECT LOCATE('unit', 'Community unit');
SELECT LOCATE('unit', 'Community unit', 7);
```

## LOWER()

```
LOWER(str)
```

Эта функция является прямой противоположностью функции UPPER. Она возвращает строку *str*, все буквы которой переводятся в нижний регистр. Следующий код возвращает строку queen elizabeth ii:

```
SELECT LOWER('Queen Elizabeth II');
```

## UPPER()

UPPER(*str*)

Эта функция является прямой противоположностью функции LOWER. Она возвращает строку *str*, все буквы которой переводятся в верхний регистр. Следующий код возвращает строку I CAN'T HELP SHOUTING:

```
SELECT UPPER('I can't help shouting');
```

## QUOTE()

QUOTE(*str*)

Возвращает строку, помещенную в кавычки, которая будет готова к использованию в инструкции SQL, для чего в ней отключаются все неоднозначно толкуемые символы. Возвращаемая строка заключается в одинарные кавычки, а перед всеми имеющимися в ней одинарными кавычками, обратными слешами, ASCII-символами NUL и Ctrl-Z устанавливается обратный слеш. Если аргумент имеет значение NULL, возвращаемое значение является словом NULL, не заключенным в кавычки. Код примера возвращает следующую строку:

```
'I\'m hungry'
```

Обратите внимание на то, как символ одинарной кавычки (') был заменен символами /'.

```
SELECT QUOTE("I'm hungry");
```

## REPEAT()

REPEAT(*str*, *count*)

Возвращает строку, содержащую *count* копий строки *str*. Если *count* меньше единицы, возвращается пустая строка. Если какой-нибудь из параметров имеет значение NULL, возвращается NULL. Следующий код возвращает строки Ho Ho Ho и Merry Christmas:

```
SELECT REPEAT('Ho', 3), 'Merry Christmas';
```

## REPLACE()

REPLACE(*str*, *from*, *to*)

Возвращает строку *str*, в которой все появления строки *from* заменены строкой *to*. При проведении поиска подстроки *from* поиск и замена чувствительны к регистру. Следующий код возвращает строку Cheeseburger and Coke:

```
SELECT REPLACE('Cheeseburger and Fries', 'Fries', 'Coke');
```

## TRIM()

TRIM([*specifier remove* FROM] *str*)

Возвращает строку *str*, из которой удалены все префиксы и суффиксы, имеющие значение *remove*. В качестве *specifier* может быть указан один из спецификаторов BOTH (оба), LEADING (ведущие) или TRAILING (замыкающие). Если спецификатор не указан, предполагается спецификатор BOTH. Строка *remove* является необязательным параметром, и если она опущена, то удаляются пробелы. Следующий код возвращает строки No Padding и Hello__:

```
SELECT TRIM(' No Padding ');
SELECT TRIM(LEADING '_' FROM '__Hello__');
```

## LTRIM() и RTRIM()

LTRIM(*str*) и RTRIM(*str*)

Функция RTRIM возвращает строку *str*, у которой удалены все пробелы в начале, а функция RTRIM делает то же самое, но в отношении замыкающих пробелов. Следующий код возвращает строки «No Padding » и « No Padding»:

```
SELECT LTRIM(' No Padding ');
SELECT RTRIM(' No Padding ');
```

# Функции для работы с датами

Даты являются важной составной частью большинства баз данных. При проведении финансовых транзакций должны записываться даты, при повторных выставлениях счетов должны учитываться сроки истечения действия кредитных карт и т. д. Поэтому неудивительно, что в MySQL имеется широкий спектр функций для работы с датами.

## CURDATE()

CURDATE()

Возвращает текущую дату в формате YYYY-MM-DD или YYYMMDD в зависимости от того, в каком контексте используется функция, строковом или числовом. 2 мая 2016 года следующий код вернул бы значения 2016-05-02 и 20160502:

```
SELECT CURDATE();
SELECT CURDATE() + 0;
```

## DATE()

DATE(*expr*)

Извлекает дату из выражения DATETIME, переданного в аргументе *expr*. Следующий код возвращает значение 1961-05-02:

```
SELECT DATE('1961-05-02 14:56:23');
```

# DATE_ADD()

DATE_ADD(*date*, INTERVAL *expr unit*)

Возвращает результат добавления выражения *expr*, в котором к дате применяется единица измерения *unit*. Аргумент *date* является стартовой датой или значением DATETIME, а *expr* для отрицательных интервалов может начинаться с минуса (-). В табл. ПГ.1 показаны типы интервалов, поддерживаемые и ожидаемые в качестве значений *expr*. Обратите внимание на приведенные в этой таблице примеры, которые показывают, в каких случаях значение *expr* должно быть заключено в кавычки, чтобы база данных MySQL смогла их правильно интерпретировать (при любых сомнениях лучше добавить кавычки, которые не помешают работе).

**Таблица ПГ.1.** Ожидаемые значения expr

| **Тип** | **Ожидаемое значение** *expr* | **Пример** |
|---|---|---|
| MICROSECOND | МИКРОСЕКУНДЫ | 111111 |
| SECOND | СЕКУНДЫ | 11 |
| MINUTE | МИНУТЫ | 11 |
| HOUR | ЧАСЫ | 11 |
| DAY | ДНИ | 11 |
| WEEK | НЕДЕЛИ | 11 |
| MONTH | МЕСЯЦЫ | 11 |
| QUARTER | КВАРТАЛЫ | 1 |
| YEAR | ГОДЫ | 11 |
| SECOND_MICROSECOND | 'СЕКУНДЫ.МИКРОСЕКУНДЫ' | 11.22 |
| MINUTE_MICROSECOND | 'МИНУТЫ.МИКРОСЕКУНДЫ' | 11.22 |
| MINUTE_SECOND | 'МИНУТЫ:СЕКУНДЫ' | '11:22' |
| HOUR_MICROSECOND | 'ЧАСЫ.МИКРОСЕКУНДЫ' | 11.22 |
| HOUR_SECOND | 'ЧАСЫ:МИНУТЫ:СЕКУНДЫ' | '11:22:33' |
| HOUR_MINUTE | 'ЧАСЫ:МИНУТЫ' | '11:22' |
| DAY_MICROSECOND | 'ДНИ.МИКРОСЕКУНДЫ' | 11.22 |
| DAY_SECOND | 'ДНИ ЧАСЫ:МИНУТЫ:СЕКУНДЫ' | '11 22:33:44' |
| DAY_MINUTE | 'ДНИ ЧАСЫ:МИНУТЫ' | '11 22:33' |
| DAY_HOUR | 'ДНИ ЧАСЫ' | '11 22' |
| YEAR_MONTH | 'ГОДЫ-МЕСЯЦЫ' | '11-2' |

Для вычитания интервала дат можно также воспользоваться функцией DATE_SUB. Но функциями DATE_ADD и DATE_SUB можно вообще не пользоваться, поскольку MySQL допускает использование непосредственных арифметических операций с датами. Данный код:

```
SELECT DATE_ADD('1975-01-01', INTERVAL 77 DAY);
SELECT DATE_SUB('1982-07-04', INTERVAL '3-11' YEAR_MONTH);
SELECT '2016-12-31 23:59:59' + INTERVAL 1 SECOND;
SELECT '2000-01-01' - INTERVAL 1 SECOND;
```

возвращает следующие значения:

```
1975-03-19
1978-08-04
2017-01-01 00:00:00
1999-12-31 23:59:59
```

Обратите внимание на то, как в последних двух командах используются непосредственные арифметические операции с датами без обращения к функциям.

# DATE_FORMAT()

```
DATE_FORMAT(date, format)
```

Эта функция возвращает значение даты *date*, отформатированное в соответствии со строкой форматирования *format*. В табл. ПГ.2 показаны спецификаторы, которые могут использоваться в строке форматирования *format*. Учтите, что символ % нужно ставить так, как показано в таблице, то есть впереди каждого спецификатора. Следующий код возвращает заданную дату и время в виде «Thursday May 4th 2016 03:02 AM»:

```
SELECT DATE_FORMAT('2016-05-04 03:02:01', '%W %M %D %Y %h:%i %p');
```

**Таблица ПГ.2.** Спецификаторы, использующиеся в функции DATE_FORMAT

| Спецификатор | Описание |
|---|---|
| %a | Сокращенное название дня недели (Sun — Sat) |
| %b | Сокращенное название месяца (Jan — Dec) |
| %c | Месяц в числовом формате (0–12) |
| %D | День месяца с английским суффиксом (0th, 1st, 2nd, 3rd...) |
| %d | День месяца в числовом формате (00–31) |
| %e | День месяца в числовом формате (0–31) |
| %f | Микросекунды (000000–999999) |
| %H | Час (00–23) |
| %h | Час (01–12) |
| %I | Час (01–12) |
| %i | Минуты в числовом формате (00–59) |
| %j | День года (001–366) |
| %k | Час (0–23) |
| %l | Час (1–12) |
| %M | Название месяца (January — December) |
| %m | Месяц в числовом формате (00–12) |
| %p | AM или PM (до или после полудня) |
| %r | Время в 12-часовом формате (hh:mm:ss, за которыми следует AM или PM) |
| %S | Секунды (00–59) |
| %s | Секунды (00–59) |
| %T | Время в 24-часовом формате (hh:mm:ss) |
| %U | Неделя (00–53), когда первым днем недели считается воскресенье |
| %u | Неделя (00–53), когда первым днем недели считается понедельник |

*Продолжение*

**Таблица ПГ.2** *(продолжение)*

| Спецификатор | Описание |
|---|---|
| %V | Неделя (00–53), когда первым днем недели считается воскресенье; используется со спецификатором %X |
| %v | Неделя (00–53), когда первым днем недели считается понедельник; используется со спецификатором %x |
| %W | Название дня недели (Sunday — Saturday) |
| %w | День недели (0 — воскресенье — 6 — суббота) |
| %X | Год для недели, в которой первым днем считается воскресенье, в числовом формате, четыре цифры; используется вместе с %V |
| %x | Год для недели, в которой первым днем считается понедельник, в числовом формате, четыре цифры; используется вместе с %v |
| %Y | Год в числовом формате, четыре цифры |
| %y | Год в числовом формате, две цифры |
| %% | Символ % как таковой |

# DAY()

DAY(*date*)

Возвращает для даты *date* день месяца в диапазоне от 1 до 31 или возвращает 0 для дат, содержащих нулевую составляющую дней, таких как «0000-00-00» или «2010-00-00». Для возвращения таких же значений можно также воспользоваться функцией DAYOFMONTH. Следующий код возвращает значение 3:

```
SELECT DAY('2016-02-03');
```

# DAYNAME()

DAYNAME(*date*)

Возвращает название дня недели для даты *date*. Следующий код возвращает строку Saturday:

```
SELECT DAYNAME('2016-02-03');
```

# DAYOFWEEK()

DAYOFWEEK(*date*)

Возвращает номер дня недели для даты *date* в диапазоне от 1 для воскресенья до 7 для субботы. Следующий код возвращает значение 7:

```
SELECT DAYOFWEEK('2016-02-03');
```

# DAYOFYEAR()

DAYOFYEAR(*date*)

Возвращает день года для даты *date* в диапазоне от 1 до 366. Следующий код возвращает значение 34:

```
SELECT DAYOFYEAR('2016-02-03');
```

## LAST_DAY()

```
LAST_DAY(date)
```

Возвращает последний день месяца для заданной в формате DATETIME даты *date*. Если аргумент имеет неправильный формат, возвращает NULL. Данный код:

```
SELECT LAST_DAY('2016-02-03');
SELECT LAST_DAY('2016-03-11');
SELECT LAST_DAY('2016-04-26');
```

возвращает следующие значения:

```
2016-02-29
2016-03-31
2016-04-30
```

Оправдывая все ожидания, функция корректно возвращает 29-й день февраля, 31-й день марта и 30-й день апреля 2011 года.

## MAKEDATE()

```
MAKEDATE(year, dayofyear)
```

Возвращает дату, соответствующую предоставленному году *year* и дню года *dayofyear*. Если *dayofyear* имеет нулевое значение, результат будет иметь значение NULL. Следующий код возвращает дату «2016-10-01»:

```
SELECT MAKEDATE(2016,274);
```

## MONTH()

```
MONTH(date)
```

Возвращает месяц даты *date* в диапазоне от 1 до 12 с января по декабрь. Для дат, у которых часть, относящаяся к месяцу, имеет нулевое значение, например «0000-00-00» или «2012-00-00», возвращает нуль. Следующий код возвращает значение 7:

```
SELECT MONTH('2016-07-11');
```

## MONTHNAME()

```
MONTHNAME(date)
```

Возвращает полное название месяца для даты *date*. Следующий код возвращает строку July:

```
SELECT MONTHNAME('2016-07-11');
```

## SYSDATE()

Возвращает текущую дату и время в виде значения в формате либо YYYY-MM-DD HH:MM:SS, либо YYYYMMDDHHMMSS, в зависимости от того, в каком контексте

используется функция, строковом или числовом. Аналогичным образом работает функция NOW, за исключением того, что она возвращает время и дату только на момент запуска текущей инструкции, а функция SYSDATE возвращает время и дату именно на момент вызова самой функции. 19 декабря 2016 года следующий код вернет значения 2016-12-19 19:11:13 и 20161219191113:

```
SELECT SYSDATE(),
SELECT SYSDATE() + 0;
```

# YEAR()

```
YEAR(date)
```

Возвращает год для даты *date* в диапазоне от 1000 до 9999 или 0 для нулевой даты. Следующий код возвращает год 1999:

```
SELECT YEAR('1999-08-07');
```

# WEEK()

```
WEEK(date [, mode])
```

Возвращает номер недели для даты *date*. Если функции передан необязательный параметр *mode*, возвращенный номер недели будет модифицирован в соответствии с описанием, приведенным в табл. ПГ.3. Можно также воспользоваться функцией WEEKOFYEAR, работа которой эквивалентна работе функции WEEK при использовании режима 3. Следующий код возвращает номер недели, равный 14:

```
SELECT WEEK('2016-04-04', 1);
```

**Таблица ПГ.3.** Режимы работы, поддерживаемые функцией WEEK

| Режим | Первый день недели | Диапазон | Когда неделя 1 – это первая неделя... |
|---|---|---|---|
| 0 | Воскресенье | 0–53 | С воскресеньем в этом году |
| 1 | Понедельник | 0–53 | С более чем тремя днями в этом году |
| 2 | Воскресенье | 1–53 | С воскресеньем в этом году |
| 3 | Понедельник | 1–53 | С более чем тремя днями в этом году |
| 4 | Воскресенье | 0–53 | С более чем тремя днями в этом году |
| 5 | Понедельник | 0–53 | С понедельником в этом году |
| 6 | Воскресенье | 1–53 | С более чем тремя днями в этом году |
| 7 | Понедельник | 1–53 | С понедельником в этом году |

# WEEKDAY()

```
WEEKDAY(date)
```

Возвращает номер дня недели для даты *date* в диапазоне от 0 (понедельник) до 6 (суббота). Следующий код возвращает значение 1:

```
SELECT WEEKDAY('2016-04-04');
```

# Функции для работы с временем

Иногда приходится работать не с датой, а с временем, и MySQL предоставляет для этого большое количество функций.

## CURTIME()

```
CURTIME()
```

Возвращает текущее время в виде значения, имеющего формат HH:MM:SS или HHMMSS.uuuuuu в зависимости от того, в каком контексте используется функция, строковом или числовом. Значение дается с учетом текущего часового пояса. При текущем времени 11:56:23 следующий код возвращает значения 11:56:23 и 115623.000000:

```
SELECT CURTIME()
SELECT CURTIME() + 0;
```

## HOUR()

```
HOUR(time)
```

Возвращает значение часа для времени *time*. Следующий код возвращает значение 11:

```
SELECT HOUR('11:56:23');
```

## MINUTE()

```
MINUTE(time)
```

Возвращает значение минуты для времени *time*. Следующий код возвращает значение 56:

```
SELECT MINUTE('11:56:23');
```

## SECOND()

```
SECOND(time)
```

Возвращает значение секунды для времени *time*. Следующий код возвращает значение 23:

```
SELECT SECOND('11:56:23');
```

## MAKETIME()

```
MAKETIME(hour, minute, second)
```

Возвращает значение времени, вычисленное на основе аргументов часа *hour*, минуты *minute* и секунды *second*. Следующий код возвращает время 11:56:23:

```
SELECT MAKETIME(11, 56, 23);
```

## TIMEDIFF()

```
TIMEDIFF(expr1, expr2)
```

Возвращает разницу между *expr1* и *expr2 (expr1 – expr2)* в виде значения времени. Оба аргумента, *expr1* и *expr2*, должны быть выражениями одинакового типа в формате TIME или DATETIME. Следующий код возвращает значение 01:37:38:

```
SELECT TIMEDIFF('2000-01-01 01:02:03', '1999-12-31 23:24:25');
```

## UNIX_TIMESTAMP()

```
UNIX_TIMESTAMP([date])
```

Эта функция при вызове без необязательного аргумента *date* возвращает в формате беззнакового целого числа то количество секунд, которое прошло с нуля часов, нуля минут и нуля секунд универсального синхронного времени (UTC) 1 января 1970 года. Если функции передается параметр *date*, тогда возвращаемое значение содержит количество секунд, которое прошло с 1970 года по указанную дату *date*. Следующий код возвращает значение 946684800 (количество секунд, которое прошло до начала нового тысячелетия), а затем возвращает отметку времени TIMESTAMP, представляющую текущее время системы Unix на момент запуска функции:

```
SELECT UNIX_TIMESTAMP('2000-01-01');
SELECT UNIX_TIMESTAMP();
```

## FROM_UNIXTIME()

```
FROM_UNIXTIME(unix_timestamp [, format])
```

Возвращает параметр *unix_timestamp* либо в формате строки YYYY-MM-DD HH:MM:SS, либо в формате числа YYYYMMDDHHMMSS.uuuuuu в зависимости от того, в каком контексте используется функция, строковом или числовом. Если задан необязательный параметр *format*, результат форматируется в соответствии со спецификаторами, показанными в табл. 8.11. Следующий код возвращает строки «2000-01-01 00:00:00» и «Saturday January 1st 2000 12:00 AM»:

```
SELECT FROM_UNIXTIME(946684800);
SELECT FROM_UNIXTIME(946684800, '%W %M %D %Y %h:%i %p');
```