Максим Кузнецов
Игорь Симдянов

PHP 5/6

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.068+800.92PHP5/6
ББК 32.973.26–018.1
К89

Кузнецов, М. В.

К89 PHP 5/6 / М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2010. — 1024 с.: ил. — (В подлиннике)

ISBN 978-5-9775-0304-4

Рассмотрены самые последние версии языка разработки серверных сценариев PHP — 5.3 и 6.0. Подробно описан язык PHP, в том числе вопросы объектно-ориентированного программирования на PHP, обработки исключительных ситуаций, взаимодействия с MySQL и многое другое. В книге обсуждаются все вопросы, с которыми может столкнуться Web-разработчик, начиная с создания инструментария для быстрой разработки Web-приложений и последних нововведений языка программирования PHP и заканчивая вопросами безопасности и особенностями программирования клиент-серверных приложений. Книга имеет практическую направленность, т. к. содержит множество примеров, взятых из практики разработки динамических Web-сайтов.

Для программистов и Web-разработчиков

УДК 681.3.068+800.92PHP5/6
ББК 32.973.26–018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 82,56.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12.

ISBN 978-5-9775-0304-4

© Кузнецов М. В., Симдянов И. В., 2009
© Оформление, издательство "БХВ-Петербург", 2009

Оглавление

Введение **1**

Нововведения PHP 6 1
Путеводитель по книге 2
On-line-поддержка книги 4
Благодарности 4

Глава 1. Интернет **5**

1.1. История развития Интернета 5
1.2. Принципы работы Интернета 7
1.3. Место и роль PHP в Интернете 12
1.3.1. Серверные технологии 13
UNIX-подобная операционная система 13
Web-сервер 13
Серверный язык 13
Файлы и базы данных 14
Электронная почта 15
1.3.2. Клиентские технологии 15
Web-браузеры, HTML 15
Каскадные таблицы стилей CSS и XML 16
Flash-ролики 16
FTP-клиенты 17
Удаленный доступ к серверу. Протокол SSH 17

Глава 2. Быстрый старт **18**

2.1. Скрипты 18
2.2. Начальные и конечные теги 20
2.3. Использование точки с запятой 22
2.4. Составные выражения. Фигурные скобки 24
2.5. Комментарии 25

Глава 3. Переменные и типы данных **27**

3.1. Объявление переменной. Оператор = ... 27
3.2. Типы данных 28
3.3. Целые числа 29

3.4. Вещественные числа ... 30
3.5. Строки ... 31
3.6. Кавычки ... 31
3.7. Оператор <<< ... 35
3.8. Обращение к неинициализированной переменной. Замечания (Notice) ... 35
3.9. Специальный тип *NULL* ... 37
3.10. Логический тип ... 38
3.11. Уничтожение переменной. Конструкция *unset()* ... 38
3.12. Проверка существования переменной. Конструкции *isset()* и *empty()* ... 39
3.13. Определение типа переменной ... 41
3.14. Неявное приведение типов ... 46
3.15. Явное приведение типов ... 48
3.16. Динамические переменные ... 52

Глава 4. Константы ... 54

4.1. Объявление константы. Функция *define()* ... 54
4.2. Функции для работы с константами ... 58
4.3. Константы с динамическими именами. Функция *constant()* ... 58
4.4. Проверка существования константы ... 59
4.5. Предопределенные константы ... 60

Глава 5. Операторы и конструкции языка ... 61

5.1. Объединение строк. Оператор "точка" ... 61
5.2. Конструкция *echo*. Оператор "запятая" ... 62
5.3. Арифметические операторы ... 63
5.4. Поразрядные операторы ... 67
5.5. Операторы сравнения ... 72
5.6. Условный оператор *if* ... 75
5.7. Логические операторы ... 77
5.8. Условный оператор *x ? y : z* ... 84
5.9. Переключатель *switch* ... 85
5.10. Цикл *while* ... 90
5.11. Цикл *do ... while* ... 96
5.12. Цикл *for* ... 97
5.13. Включение файлов ... 101
5.14. Подавление вывода ошибок. Оператор @ ... 106
5.15. Приоритет выполнения операторов ... 108

Глава 6. Массивы ... 110

6.1. Создание массива ... 110
6.2. Ассоциативные и индексные массивы ... 116
6.3. Многомерные массивы ... 121
6.4. Интерполяция элементов массива в строки ... 122
6.5. Конструкция *list()* ... 123

6.6. Обход массива ... 125
6.7. Цикл *foreach* ... 129
6.8. Сечения массива ... 131
6.9. Слияние массивов ... 133
6.10. Сравнение массивов ... 136
6.11. Проверка существования элементов массива ... 140
6.12. Строки как массивы ... 144
6.13. Количество элементов в массиве ... 145
6.14. Сумма элементов массива ... 147
6.15. Случайные элементы массива ... 147
6.16. Сортировка массивов ... 150
6.17. Добавление/удаление элементов массива ... 158
6.18. Работа с ключами массива ... 165
6.19. Преобразование массивов ... 170
6.20. Суперглобальные массивы. Массив *$_SERVER* ... 177
6.20.1. Элемент *$_SERVER['DOCUMENT_ROOT']* ... 178
6.20.2. Элемент *$_SERVER['HTTP_ACCEPT']* ... 178
6.20.3. Элемент *$_SERVER['HTTP_ACCEPT_LANGUAGE']* ... 179
6.20.4. Элемент *$_SERVER['HTTP_HOST']* ... 180
6.20.5. Элемент *$_SERVER['HTTP_REFERER']* ... 180
6.20.6. Элемент *$_SERVER['HTTP_USER_AGENT']* ... 181
6.20.7. Элемент *$_SERVER['REMOTE_ADDR']* ... 182
6.20.8. Элемент *$_SERVER['SCRIPT_FILENAME']* ... 182
6.20.9. Элемент *$_SERVER['SERVER_NAME']* ... 182
6.20.10. Элемент *$_SERVER['REQUEST_METHOD']* ... 183
6.20.11. Элемент *$_SERVER['QUERY_STRING']* ... 183
6.20.12. Элемент *$_SERVER['PHP_SELF']* ... 184
6.20.13. Элемент *$_SERVER['REQUEST_URI']* ... 184

Глава 7. Функции ... 185

7.1. Объявление и вызов функции ... 185
7.2. Параметры функции ... 188
7.3. Передача параметров по значению и ссылке ... 189
7.4. Необязательные параметры ... 190
7.5. Переменное количество параметров ... 191
7.6. Глобальные переменные ... 193
7.7. Статические переменные ... 194
7.8. Возврат массива функцией ... 195
7.9. Рекурсивные функции ... 196
7.10. Вложенные функции ... 197
7.11. Динамическое имя функции ... 198
7.12. Анонимные функции ... 199
7.13. Проверка существования функции ... 201
7.14. Неявное выполнение функций. Оператор *declare()* ... 206
7.15. Вспомогательные функции ... 211

Глава 8. Взаимодействие PHP с HTML 214

8.1. Передача параметров методом *GET* 214
8.2. HTML-форма и ее обработчик 218
8.3. Текстовое поле 223
8.4. Поле для приема пароля 224
8.5. Текстовая область 225
8.6. Скрытое поле 226
8.7. Флажок 227
8.8. Список 229
8.9. Переключатель 231
8.10. Загрузка файла на сервер 232

Глава 9. Строковые функции 237

9.1. Функции для работы с символами 237
9.2. Поиск в строке 243
9.3. Замена в тексте 248
9.4. Преобразование регистра 253
9.5. Работа с HTML-кодом 255
9.6. Экранирование 264
9.7. Локальные настройки (локаль) 266
9.8. Форматный вывод 273
9.9. Преобразование кодировок 282
9.10. Сравнение строк 292
9.11. Хранение данных 298
9.12. Работа с путями к файлам и каталогам 301
9.13. Объединение и разбиение строк 303

Глава 10. Регулярные выражения 313

10.1. Как изучать регулярные выражения? 313
10.2. Синтаксис регулярных выражений 314
10.3. Функции для работы с регулярными выражениями 317
10.4. Функция *preg_match()* 318
10.5. Функция *preg_match_all()* 323
10.6. Функция *preg_replace()* 326
10.7. Функция *preg_replace_callback()* 330
10.8. Функция *preg_split()* 332
10.9. Функция *preg_quote()* 334

Глава 11. Дата и время 336

11.1. Формирование даты и времени 336
11.2. Географическая привязка 342
11.3. Форматирование даты и времени 348

Глава 12. Математические функции ... 362

12.1. Предопределенные константы ... 362
12.2. Поиск максимума и минимума ... 363
12.3. Генерация случайных чисел ... 365
12.4. Преобразование значений между различными системами счисления ... 367
12.5. Округление чисел ... 370
12.6. Логарифмические и степенные функции ... 373
12.7. Тригонометрические функции ... 378
12.8. Информационные функции ... 383
12.9. Вычисления с повышенной точностью ... 389

Глава 13. Файлы и каталоги ... 392

13.1. Создание файлов ... 392
13.2. Манипулирование файлами ... 398
13.3. Чтение и запись файлов ... 401
13.3.1. Чтение файлов ... 404
13.3.2. Запись файлов ... 410
13.3.3. Обязательно ли закрывать файлы? ... 414
13.3.4. Дозапись файлов ... 415
13.3.5. Уменьшение размера файла ... 416
13.3.6. Блокировка файлов ... 417
13.3.7. Файлы CSV ... 422
13.3.8. Чтение и разбор ini-файла ... 424
13.3.9. Прямое манипулирование файловым указателем ... 427
13.4. Создание и работа со ссылками ... 430
13.5. Атрибуты файла ... 432
13.6. Права доступа ... 438
13.7. Состояние файловой системы ... 442
13.8. Каталоги ... 443
13.9. Архивные файлы ... 450

Глава 14. HTTP-заголовки ... 454

14.1. Функции для управления HTTP-заголовками ... 455
14.2. Кодировка страницы ... 457
14.3. HTTP-коды состояния ... 457
14.4. Список HTTP-заголовков ... 461
14.5. Подавление кэширования ... 464

Глава 15. Cookie ... 468

Глава 16. Сессии ... 473

16.1. Функции обработки сессии ... 473
16.2. Перегрузка механизма сессии ... 477

Глава 17. Управление выводом ... 481

17.1. Функции управления выводом ... 481
17.2. Размер и тип документа ... 483
17.3. Подсветка ссылок ... 486
17.4. Вложенная буферизация ... 487
17.5. Обработчики буферов ... 488
17.6. Список применяемых обработчиков ... 489
17.7. Сжатие страницы ... 491
17.8. Преобразование кодировки страницы ... 491
17.9. Преобразование адресов ссылок ... 492

Глава 18. Средства шифрования ... 494

18.1. Необратимое шифрование ... 494
18.2. Система регистрации и аутентификации ... 497
18.3. Обратимое шифрование ... 501

Глава 19. Сетевое взаимодействие. Сокеты и CURL ... 502

19.1. Обращение к удаленным страницам ... 502
19.2. Сокеты ... 504
19.3. Библиотека CURL ... 507
19.4. Получение HTTP-заголовков с сервера ... 515
19.5. Определение размера файла на удаленном хосте ... 519
19.6. Отправка данных методом POST ... 519
19.7. Фальсификация реферера ... 522
19.8. Фальсификация пользовательского агента ... 526
19.9. Фальсификация cookie ... 527
19.10. Работа с доменами и IP-адресами ... 530
19.11. Определение IP-адреса по сетевому адресу ... 530
19.12. Определение сетевого адреса по IP-адресу ... 531
19.13. Следование реферальному серверу ... 532

Глава 20. Взаимодействие с интерпретатором PHP ... 535

20.1. Настройки конфигурационного файла php.ini ... 535
20.1.1. Параметры языка PHP ... 536
20.1.2. Директивы безопасности ... 539
20.1.3. Настройка подсветки PHP-кода ... 545
20.1.4. Кэш файловой системы ... 547
20.1.5. Взаимодействие с клиентом ... 547
20.1.6. Ограничение ресурсов ... 548
20.1.7. Обработка ошибок ... 549
20.1.8. Обработка данных ... 552
20.1.9. Загрузка файлов ... 558
20.1.10. Сетевой доступ ... 558

20.1.11. Подключение расширений ... 558
20.1.12. Настройка сессии ... 559
20.1.13. Настройка даты и времени ... 561
20.2. Изменение настроек php.ini средствами Apache ... 561
20.3. Функции управления интерпретатором PHP ... 564

Глава 21. Объекты и классы ... 567

21.1. Введение в объектно-ориентированное программирование ... 567
21.2. Создание класса ... 568
21.3. Создание объекта ... 569
21.4. Инкапсуляция. Спецификаторы доступа ... 571
21.5. Методы класса. Член *$this* ... 572
21.6. Дамп объекта ... 576
21.7. Специальные методы класса ... 577
21.8. Функции для работы с методами и классами ... 578
21.9. Конструктор. Метод *__construct()* ... 580
21.10. Параметры конструктора ... 582
21.11. Деструктор. Метод *__destruct()* ... 584
21.12. Автозагрузка классов. Функция *__autoload()* ... 586
21.13. Аксессоры. Методы *__set()* и *__get()* ... 586
21.14. Проверка существования члена класса. Метод *__isset()* ... 588
21.15. Уничтожение члена класса. Метод *__unset()* ... 589
21.16. Динамические методы. Метод *__call()* ... 591
21.17. Интерполяция объекта. Метод *__toString()* ... 593
21.18. Экспорт объектов. Метод *__set_state()* ... 595
21.19. Статические члены класса ... 600
21.20. Статические методы класса ... 603
21.21. Константы класса ... 603
21.22. Предопределенные константы ... 605
21.23. Клонирование объекта ... 607
21.24. Управление процессом клонирования. Метод *__clone()* ... 608
21.25. Сериализация объектов ... 610
21.26. Управление сериализацией. Методы *__sleep()* и *__wakeup()* ... 612

Глава 22. Наследование ... 621

22.1. Наследование ... 621
22.2. Спецификаторы доступа и наследование ... 624
22.3. Перегрузка методов ... 627
22.4. Полиморфизм ... 628
22.5. Абстрактные классы ... 630
22.6. Абстрактные методы ... 631
22.7. *Final*-методы класса ... 632
22.8. *Final*-классы ... 634

Глава 23. Интерфейсы ... 635

23.1. Создание интерфейса ... 635
23.2. Интерфейсы и наследование классов ... 637
23.3. Реализация нескольких интерфейсов ... 639
23.4. Проверка существования интерфейса ... 640
23.5. Наследование интерфейсов ... 641
23.6. Реализует ли объект интерфейс? ... 643

Глава 24. Обработка ошибок и исключительных ситуаций ... 647

24.1. Обработка ошибок ... 647
24.2. Обработка ошибок и исключения ... 660
24.2.1. Синтаксис исключений ... 660
24.2.2. Интерфейс класса *Exception* ... 662
24.2.3. Генерация исключений в функциях ... 666
24.2.4. Стек обработки исключительной ситуации ... 669
24.2.5. Генерация исключений в классах ... 671
24.2.6. Генерация исключений в иерархиях классов ... 675
24.2.7. Использование объекта класса *Exception* в строковом контексте ... 678
24.2.8. Создание собственных исключений ... 679
24.2.9. Создание новых типов исключений ... 683
24.2.10. Перехват исключений производных классов ... 685
24.2.11. Что происходит, когда исключения не перехватываются? ... 687
24.2.12. Вложенные контролируемые блоки ... 689
24.2.13. Повторная генерация исключений ... 691

Глава 25. Отражения ... 694

25.1. Иерархия классов отражения ... 694
25.2. Отражение функции. Класс *ReflectionFunction* ... 695
25.3. Отражение параметра функции. Класс *ReflectionParameter* ... 700
25.4. Отражение класса. Класс *ReflectionClass* ... 702
25.5. Отражение объекта. Класс *ReflectionObject* ... 709
25.6. Отражение метода класса. Класс *ReflectionMethod* ... 710
25.7. Отражение члена класса. Класс *ReflectionProperty* ... 712
25.8. Исключения механизма отражения ... 715
25.9. Отражение расширения. Класс *ReflectionExtension* ... 716
25.10. Вспомогательный класс *Reflection* ... 719

Глава 26. Предопределенные классы ... 721

26.1. Класс *Dir* ... 721
26.2. Классы времени ... 723
26.3. Библиотека SPL ... 730
26.3.1. Итераторы ... 731
26.3.2. Интерфейс *Iterator* ... 734
26.3.3. Класс *DirectoryIterator* ... 736

26.3.4. Класс *FilterIterator* ... 738
26.3.5. Класс *LimitIterator* ... 739
26.3.6. Рекурсивные итераторы ... 740

Глава 27. Работа с СУБД MySQL ... 742

27.1. Введение в СУБД и SQL ... 743
27.2. Первичные ключи ... 746
27.3. Создание и удаление базы данных ... 747
27.4. Выбор базы данных ... 749
27.5. Типы данных ... 751
27.6. Создание и удаление таблиц ... 756
27.7. Вставка числовых значений в таблицу ... 763
27.8. Вставка строковых значений в таблицу ... 764
27.9. Вставка календарных значений ... 766
27.10. Вставка уникальных значений ... 769
27.11. Механизм *AUTO_INCREMENT* ... 769
27.12. Многострочный оператор *INSERT* ... 770
27.13. Удаление данных ... 771
27.14. Обновление записей ... 772
27.15. Выборка данных ... 775
27.16. Условная выборка ... 776
27.17. Псевдонимы столбцов ... 782
27.18. Сортировка записей ... 783
27.19. Вывод записей в случайном порядке ... 786
27.20. Ограничение выборки ... 786
27.21. Вывод уникальных значений ... 787
27.22. Объединение таблиц ... 789
27.23. Функции MySQL ... 791
27.23.1. Математические функции ... 791
Вычисление площади треугольников ... 793
Округление результатов вычисления ... 795
27.23.2. Функции даты и времени ... 796
Форматирование календарных значений ... 801
Вычисление возраста человека ... 802
Преобразование даты в UNIXSTAMP-формат ... 804
27.23.3. Строковые функции ... 805
Изменение кодировки строки ... 809
Первые несколько символов строки ... 810
Извлечение инициалов ... 810
Изменение регистра строки ... 811
27.23.4. Функции шифрования ... 812
Обратимое шифрование ... 814
Необратимое шифрование ... 815
27.23.5. Агрегатные функции ... 816
Среднее значение ... 817
Сортировка агрегатных значений ... 820

Подсчет количества записей в таблице ... 820
Объединение значений группы ... 823
Поиск минимального и максимального значений ... 825
Сумма столбца ... 826
27.23.6. Разное ... 827
Преобразование IP-адреса ... 828

Глава 28. Взаимодействие MySQL и PHP ... 830

28.1. Библиотека php_mysql ... 830
28.1.1. Установка соединения с базой данных ... 830
28.1.2. Выбор базы данных ... 833
28.1.3. Выполнение SQL-запросов ... 835
28.1.4. Получение результатов запроса ... 836
28.1.5. Количество строк в таблице ... 844
28.1.6. Экранирование данных. SQL-инъекции ... 846
28.2. Библиотека php_mysqli ... 853
28.2.1. Создание базы данных ... 860
28.2.2. Создание и заполнение таблицы ... 861
28.2.3. Заполнение связанных таблиц ... 862
28.2.4. Вывод данных ... 865
28.2.5. Повторное чтение результирующей таблицы ... 867
28.2.6. Количество строк в таблице ... 868
28.2.7. Удаление данных ... 869
28.2.8. Сортировка ... 871
28.2.9. Параметризация SQL-запросов ... 873

Глава 29. FTP-менеджер ... 875

29.1. Функции для работы с FTP-сервером ... 875
29.2. Какой объем жесткого диска занимает сайт? ... 879
29.3. Перенос сайта с одного хоста на другой ... 881

Глава 30. Электронная почта ... 885

30.1. Отправка почтового сообщения ... 885
30.2. Рассылка писем ... 887
30.3. Отправка писем с вложением ... 888
30.4. Отправка писем со встроенными изображениями ... 891

Глава 31. Динамические изображения. Библиотека GDLib ... 894

31.1. Информационные функции ... 894
31.1.1. Текущая версия библиотеки GDLib ... 895
31.1.2. Формат файла ... 897
31.1.3. Размер файла ... 898
31.1.4. Получение MIME-типа файла ... 899
31.2. Функции создания изображений ... 901

31.3. Функции сохранения и вывода изображений ... 903
31.4. Функции преобразования изображений ... 905
31.4.1. Создание уменьшенной копии изображения ... 906
31.4.2. Поворот изображения ... 908
31.5. Функции для работы с цветом ... 909
31.5.1. Заливка изображения цветом ... 911
31.5.2. Получение цвета заданного пиксела ... 913
31.5.3. Уменьшение количества цветов в изображении ... 914
31.5.4. Замена одного цвета другим ... 915
31.6. Функции рисования ... 916
31.6.1. Кривая Безье ... 918
31.6.2. Построение гистограммы средствами GDLib ... 922
31.6.3. Построение круговой диаграммы средствами GDLib ... 924
31.7. Функции настройки рисования ... 925
31.7.1. Пунктирная линия ... 926
31.8. Функции для работы с текстом ... 928
31.8.1. Защитное изображение для HTML-формы ... 930

Заключение ... 935

ПРИЛОЖЕНИЯ ... 937

Приложение 1. Установка и настройка PHP, Web-сервера Apache и MySQL-сервера ... 939

П1.1. Где взять дистрибутивы? ... 939
П1.1.1. Дистрибутив PHP ... 940
П1.1.2. Дистрибутив Apache ... 940
П1.1.3. Дистрибутив MySQL ... 941
П1.2. Установка Web-сервера Apache под Windows ... 942
П1.3. Установка Web-сервера Apache под Linux ... 944
П1.4. Настройка виртуальных хостов ... 945
П1.5. Настройка кодировки по умолчанию ... 949
П1.6. Управление запуском и остановкой Web-сервера Apache ... 949
П1.7. Управление Apache из командной строки ... 950
П1.8. Установка PHP под Windows ... 951
П1.8.1. Установка PHP в качестве модуля ... 951
П1.8.2. Установка PHP как CGI-приложения ... 952
П1.9. Установка PHP под Linux ... 954

Приложение 2. Установка MySQL ... 955

П2.1. Установка MySQL под Windows ... 955
П2.1.1. Процесс установки ... 955
П2.1.2. Постинсталляционная настройка ... 960
П2.1.3. Проверка работоспособности MySQL ... 966

П2.2. Установка MySQL под Linux ... 968
П2.3. Конфигурационный файл ... 971
П2.4. Утилита mysql ... 974
П2.4.1. Командная строка ... 974
П2.4.2. Работа с утилитой mysql в диалоговом режиме ... 978
П2.5. Перенос баз данных с одного сервера на другой ... 982
П2.5.1. Копирование бинарных файлов ... 983
П2.5.2. Создание SQL-дампа ... 984

Рекомендуемая литература ... 987

HTML, XML, CSS, JavaScript и Flash ... 989
PHP и Perl ... 991
СУБД MySQL ... 993
Интернет и Web-сервер Apache ... 994
Регулярные выражения ... 995
UNIX-подобные операционные системы ... 995
Методология программирования ... 997

Предметный указатель ... 999

Введение

PHP является молодым и динамично развивающимся языком программирования, который используется главным образом для создания самых разнообразных Web-приложений. Быстрое развитие языка, с одной стороны, предполагает регулярное введение новых конструкций, функций, библиотек, директив, а с другой стороны, — изъятие старых конструкций, использование которых оказалось неудобным или провоцировало создание небезопасного кода. Стадию динамичного развития проходит любой язык программирования, определить ее очень просто по быстрой смене версий и количеству нововведений в этих версиях. Со временем интервал между версиями увеличивается, каждая новая версия содержит все меньше и меньше революционных нововведений. Современный PHP пока находится в стадии быстрого развития, однако в ближайшие несколько лет прогнозируется переход к плавному изменению, характерному для состоявшихся языков.

Анонсируя PHP 6, разработчики заранее объявили о нововведениях, которые будут в новой версии. Однако затем ими было принято решение не создавать дистрибутив с нуля, а постепенно вносить изменения в текущую версию PHP 5. Если раньше, для того чтобы попробовать новинки языка, требовалось загрузить нестабильный дистрибутив, предназначенный для разработчиков, то теперь особенности PHP 6 постепенно и плавно появляются в версии PHP 5. Таким образом, к тому моменту, когда все обновления будут внедрены в язык, разработчикам останется только поменять цифру. Поэтому нами было принято решение о создании руководства по языку PHP с учетом всех последних нововведений.

Эта книга будет интересна как начинающим разработчикам, которым необходимо последовательное изложение языка, так и опытным программистам, желающим познакомиться с нововведениями языка, появившимися в последнее время.

Нововведения PHP 6

Не все нововведения языка реализованы на данный момент, — к примеру, нет сечений массивов и строк, отсутствует (в стабильных версиях) поддержка Unicode на уровне ядра, в расширении для работы с графикой не поддерживается работа с анимированными GIF-файлами. Список этот можно еще продолжить, однако интереснее отметить то, что уже реализовано:

- исправлены многочисленные непоследовательности объектно-ориентированной модели;

- значение `E_ALL` директивы `error_reporting` конфигурационного файла php.ini теперь включает `E_STRICT` *(см. разд. 3.8)*;
- средний параметр *y* в условном операторе `x ? y : z`, начиная с PHP 6.0, не является обязательным *(см. разд. 5.8)*;
- ускорена работа оператора `@` *(см. главу 5)*;
- функциональность цикла `foreach` расширена для работы с многомерными массивами;
- в операторе `break` удалена поддержка динамических меток;
- регулярные выражения POSIX исключены из ядра PHP и доступны лишь как расширение;
- введены новые предопределенные массивы для работы с датой и временем;
- изменен синтаксис множества функций (главным образом, за счет увеличения числа необязательных параметров).

Путеводитель по книге

Книга содержит введение, 31 главу, заключение и два приложения. В конце книги вы найдете обзор литературы в области Web-разработки с подробными комментариями авторов.

Глава 1 посвящена краткой истории развития Интернета, а также места и роли языка PHP в Web-технологиях, благодаря которым стало возможным существование всемирной сети.

Глава 2 содержит описание простейших базовых конструкций языка и служит введением для тех читателей, для которых PHP является первым языком программирования.

Глава 3 посвящена переменным и подробно освещает жизненный цикл переменных: создание, уничтожение, отличие типов переменных друг от друга, явное и неявное преобразование типов переменных.

Глава 4 посвящена пользовательским и предопределенным константам, а также функциям, обслуживающим константы.

Глава 5 описывает операторы и конструкции языка, подробно освещая как классические C-подобные операторы: условные, циклические, логические, поразрядные, арифметические, так и специфические для PHP операторы: "точка", "запятая", оператор подавления ошибок, конструкции включения файлов.

Глава 6 посвящена описанию массивов, подробно рассматривает как типы массивов, порядок работы с ними, вспомогательные функции. В главе также рассматривается суперглобальный массив `$_SERVER`, позволяющий взаимодействовать PHP-скрипту с окружением (операционной системой и Web-сервером).

Глава 7 описывает функции, создание собственных функций, их обслуживание, динамические, анонимные функции, функции с переменным количеством параметров и т. п.

Глава 8 посвящена взаимодействию PHP и языка разметки HTML. Львиная доля PHP-скриптов занимается обслуживанием многочисленных форм. В главе описывается син-

таксис элементов формы и типичные случаи обработки на PHP поступающих из форм данных. В этой же главе подробно описываются суперглобальные массивы `$_GET` и `$_POST`.

Глава 9 описывает строковые функции PHP: поиск и замену в строках, преобразование регистра и кодировки, форматный вывод и т. п.

Глава 10 посвящена регулярным выражениям — специализированному мини-языку, позволяющему эффективно работать со строками (осуществлять поиск, подстановку, удаление подстрок).

Глава 11 посвящена функциям, работающим с датой и временем.

Глава 12 описывает математические функции, затрагивая тригонометрические, логарифмические функции, функции общего назначения, а также функции, предназначенные для вычислений с повышенной точностью.

Глава 13 посвящена файлам и каталогам и включает подробные сведения о файловой системе, а также функциях PHP, обслуживающих файловую систему.

Глава 14 посвящена протоколу HTTP, являющемуся основой Web-среды. Язык PHP спроектирован таким образом, чтобы по возможности оградить разработчика от необходимости вникать в низкоуровневые реализации Сети. Однако эффективные Web-приложения без знания протокола HTTP освоить невозможно. В любом случае, необходимо четко понимать, как PHP реализуют те или иные сетевые возможности.

Глава 15 описывает механизм cookie, предназначенный для сеансовой работы клиентов (не поддерживаемой на уровне протокола HTTP).

Глава 16 посвящена сессиям, также предназначенным для реализации сеансовой работы.

Глава 17 описывает процесс буферизации вывода. Все, что выводится PHP-скриптом, немедленно отправляется клиенту и зачастую трудно выполнить какие-то преобразования над уже сформированным HTML-кодом. Данная глава описывает решение этой проблемы путем создания буферов вывода и манипулирования ими.

Глава 18 посвящена средствам шифрования, предоставляемым PHP.

Глава 19 описывает сокеты и библиотеку CURL, позволяющие скрипту самому выступать в роли клиента, обращаясь к удаленным серверам.

Глава 20 описывает настройки PHP-интерпретатора, а также способы управления ими как на уровне Web-сервера, так и отдельного скрипта.

Глава 21 посвящена введению в объектно-ориентированный подход, позволяющий создавать повторно-используемый код и собственные мини-языки, работающие в предметной области разработки.

Глава 22 подробно описывает наследование в объектно-ориентированном подходе PHP.

Глава 23 описывает интерфейсы в объектно-ориентированном подходе PHP.

Глава 24 посвящена обработке ошибок в PHP-скриптах, как при помощи специализированных функций, характерных для структурного подхода, так и при помощи исключений, характерных для объектно-ориентированного подхода.

Глава 25 посвящена механизму отражений, предоставляющему разработчику возможность исследования как пользовательских, так и предопределенных классов. Кроме

этого, отражения позволяют автоматически генерировать документацию иерархии классов по схеме javadoc, применяемой в технологии Java.

Глава 26 описывает предопределенные классы PHP, такие как класс `Dir` доступа к каталогам, классы даты и времени, классы библиотеки SPL.

Глава 27 посвящена базе данных MySQL — наиболее популярной базе данных, используемой совместно с PHP.

Глава 28 описывает взаимодействие PHP и MySQL.

Глава 29 посвящена FTP-протоколу, при помощи которого осуществляется взаимодействие с файловой системой удаленного сервера.

Глава 30 описывает процесс отправки электронных писем из PHP-скрипта, включая письма, содержащие вложения и встроенные изображения.

Глава 31 посвящена созданию динамических изображений при помощи библиотеки GDLib.

On-line-поддержка книги

Разнообразные дополнительные материалы вы можете найти на сайтах IT-студии Soft-Time, руководителями которой являются авторы книги:

- **http://www.softtime.ru** — головной сайт Студии;
- **http://www.softtime.org** — различные проекты Студии;
- **http://www.softtime.biz** — услуги, оказываемые нашей Студией;
- **http://www.softtime.mobi** — вариант портала для мобильных устройств.

Обсудить вопросы, которые могут возникать по мере чтения книги, можно на форуме авторов: **http://www.softtime.ru/forum/**.

Благодарности

Авторы выражают огромную благодарность сотрудникам издательства "БХВ-Петербург", благодаря которым наша рукопись увидела свет.

Мы также безмерно признательны посетителям наших форумов за интересные вопросы и конструктивное обсуждение.



Глава 1

Интернет

По данным за 2007 год количество пользователей глобальной сети Интернет достигло 1 миллиарда человек. Интернет является одним из самых грандиозных инженерных достижений человечества, изменившим как повседневную жизнь людей, так и облик современной цивилизации. В данной главе мы кратко рассмотрим историю развития Интернета и его устройство.

1.1. История развития Интернета

Прототипом Интернета послужила первая глобальная сеть ARPANET, разработка которой финансировалось Управлением перспективного планирования научно-исследовательских работ (Advanced Research Projects Agency, ARPA) Министерства обороны Соединенных Штатов Америки (США).

Цель проекта была двоякой. В идеале, необходимо было создать распределенную глобальную компьютерную сеть, способную продолжать функционировать после ядерной войны. Вывод из строя произвольного количества узлов сети не должен был приводить к выходу из строя сети в целом.

Более приземленная цель состояла в создании сети для обмена файлами и электронной почтой между университетскими учеными, сотрудниками военных ведомств и подрядчиками Министерства обороны США.

Так или иначе, осенью 1968 года ARPA заключило контракт на разработку сети ARPANET с тремя сотрудниками Кембриджского университета — Болтом (Bolt), Беранеком (Beranek) и Ньюменом (Newman), и уже в сентябре 1969 года первый участок сети был запущен в опытную эксплуатацию.

В 1975 году управление сетью было выведено из-под контроля ARPA и поручено управлению связи Министерства обороны США, которое сделало ARPANET частью собственной сети.

Одной из главных особенностей финансирования научных работ государством является то, что все разработки ведутся на деньги налогоплательщиков и не являются интеллектуальной собственностью одной из корпораций, а автоматически становятся достоянием общественности. Любой разработчик может создать программу или аппаратное обеспечение для обмена информацией через сеть. Это с одной стороны привлекло множество энтузиастов, а с другой стороны определило в дальнейшем широкое распространение сетевых технологий по всему миру.

Несмотря на то, что сеть была первоначально построена на выделенных линиях, агентство ARPA финансировало исследования технологий передачи пакетов данных по радиоканалу и каналам спутниковой связи.

В связи выделяют два способа передачи данных: обмен информацией через выделенные линии связи и обмен с коммутацией пакетов. Выделенные линии после установки соединения остаются занятыми, даже если абоненты не обмениваются информацией (так построены телефонные линии). При коммутации пакетов, данные разбиваются на части со стороны одного абонента и собираются из разрозненных частей со стороны второго абонента. В результате по одной линии связи можно передавать данные одновременно от нескольких абонентов. Именно так функционирует современный Интернет, а также IP-телефония (которая по сути использует интернет-соединения для передачи голоса). Коммутация пакетов более сложна в реализации по сравнению с выделенными каналами, однако позволяет более эффективно использовать каналы связи.

Переход на линии связи с коммутацией пактов в сети ARPANET был начат в 1980 году в связи с введением протоколов семейства TCP/IP и закончен в 1983 году.

Так как сеть разрасталась и в ее работе участвовало все больше гражданских лиц, в 1983 году Министерство обороны разделило ARPANET на две связанные сети: гражданский сектор, предназначенный для исследований и обмена открытой информации (по-прежнему носящий название ARPANET), и сеть для военных целей (получившая название MILNET).

> ***Замечание***
>
> Управление сетями было организовано таким образом, что в любой момент их можно было разъединить. Когда в ноябре 1988 года произошло заражение гражданской части сети первым червем, созданным Моррисом, такая возможность помогла предотвратить попадание червя в MILNET. Данная ситуация даже немного обыгрывается в фильме "Терминатор. Восстание машин".

Чтобы заинтересовать университеты в использовании новых протоколов TCP/IP, Министерство обороны стало продавать их программную реализацию по низкой цене, а также профинансировало разработку реализации протоколов TCP/IP для BSD UNIX (популярного варианта UNIX-подобной операционной системы, разработанной в Калифорнийском университете Беркли). В результате протоколы TCP/IP стали доступными более чем в 90% университетских компьютерных лабораториях и факультетах. Сеть ARPANET стала одним из главных магистральных каналов, к которому стали подключаться другие сети, образуя тем самым Всемирную глобальную сеть — Интернет (Internet).

Так как все больше университетов соединяли свои вычислительные возможности в локальные сети, а также подключались к ARPANET, Национальный научный фонд США (National Science Foundation или NSF) начал финансирование проекта сети CSNET (Computer Science NETwork), которая должна была связать все научные компьютерные лаборатории. В 1985 году сетью CSNET были соединены шесть суперкомпьютеров, принадлежавших Национальному научному фонду. В 1986 году NFS выделяет средства на создание новой глобальной магистральной сети NSFNET, которая связала все суперкомпьютерные центры США и объединила их с ARPANET. В том же 1986 году NSF финансирует множество проектов региональных сетей, каждая из которых связала основные научно-исследовательские учреждения в своем регионе.

Замечание

Сеть CSFNET состояла из магистрального канала высокой пропускной способности, охватывающего всю территорию США, к которому подключались местные локальные сети, охватывающие университеты и государственные учреждения.

С этого момента к Интернету стало подключаться множество частных сетей в США и Западной Европе. Темпы роста Интернета составляли более 1000% в год и поддерживались не только государственными учреждениями, к Всемирной сети стало подключаться все больше и больше частных компаний.

В 1991 году фонд NSF и другие государственные учреждения США столкнулись с фактом, что Интернет де-факто перестал быть университетской и научной сетью. Магистральный канал (с пропускной способностью 1,5 Мбит/с), к которому подключались все новые и новые сети, был перегружен. Поэтому с 1991 года правительство США приняло решение позволить приватизировать магистральный канал и арендовать его у частной компании, которая бы самостоятельно заботилась об увеличении его пропускной способности.

В декабре 1991 года три корпорации — IBM, MERIT и MCI — учредили некоммерческую организацию Advanced Network and Services (ANS), которая в 1993 году ввела в строй новый магистральный канал с пропускной способностью 45 Мбит/с. В 1995 году пропускная способность магистрального канала была еще раз увеличена до 155 Мбит/с.

С 1995 года Интернет широко используется в коммерческих целях, некоторые корпорации (например, AT&T и MCI) построили собственные магистральные каналы, которые обмениваются трафиком на условиях партнерских соглашений.

Замечание

Как правило, компания платит другой компании в том случае, если входящий трафик у ее каналов выше, чем исходящий — разница оплачивается погигабайтно. Именно этим определяется частое требование для серверов в соотношениях входящего и исходящего трафиков, зарубежного и российского, входящего в региональное кольцо и не входящего.

Современные магистральные каналы обеспечиваются пропускной способностью в десятки гигабит в секунду. В Российской Федерации имеется несколько магистральных каналов. В качестве примера можно назвать ТрансТелеКом, которым владеет компания "Российские железные дороги". В 2007 году компания владела магистральным каналом протяженностью более 50 тыс. км, обеспечивающим пропускную способность до 50 Гбит/с.

В настоящий момент в США функционирует и развивается исследовательский проект Internet2, позволяющий достигать пропускной способности 100 Гбит/с.

1.2. Принципы работы Интернета

Сама сеть построена на множестве разнородных аппаратных и программных средств, соединенных каналами связи совершенно разной природы: начиная оптоволоконными и медными линиями связи, заканчивая беспроводными и спутниковыми каналами. Объединение таких разнородных элементов стало возможным благодаря применению

протоколов связи — сеть может быть устроена произвольным образом, однако обмениваться с другими сетями она должна пакетами определенного формата.

Протокол — это набор синтаксических и семантических правил, использующихся при обмене данных между двумя компьютерами. В нем оговаривается формат блоков сообщений, описывается реакция компьютера на получение определенного типа сообщений и указываются способы обработки ошибок и других исключительных ситуаций.

Под управлением какой бы операционной системы не работали компьютеры, кто бы ни производил комплектующие, сетевые платы, маршрутизаторы и т. п., если система выполняет предписания протокола, она может стать участником сети.

Передаваемые файлы и сообщения перед отправкой разбиваются на части — *пакеты*. Пакеты от нескольких машин передаются вперемешку от одной сети к другой. Получатель на другом конце соединения собирает разрозненные куски в одно целое.

Для того чтобы обеспечить взаимодействие различных сетей и каналов связи, используется несколько протоколов, которые работают на разных уровнях (рис. 1.1).

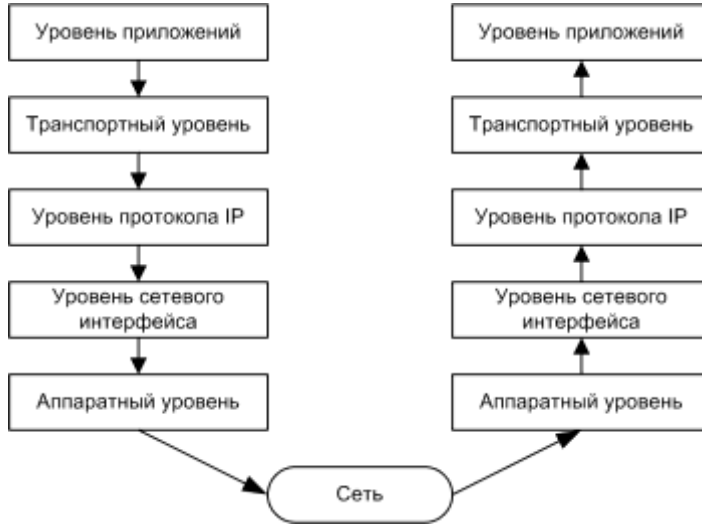


Рис. 1.1. Уровни протоколов Интернета

На *аппаратном уровне* взаимодействие определяется конкретными реализациями сетевого оборудования: это может быть оптоволокно, спутник, медный кабель, реализующий соединение через Ethernet. Задача этого уровня — передавать *сетевые фреймы*, формат которых определяется конкретным оборудованием.

На *уровне сетевого интерфейса* (который иногда называют *канальным уровнем*) протоколы обеспечивают работу сетевого оборудования, в частности платы сетевого интерфейса (реализуется как драйвер устройства). Они принимают *IP-дейтаграммы* и передают их по физической сети.

На *уровне протокола IP* обеспечивается взаимодействие двух узлов сети. Каждый из узлов обозначается уникальным IP-адресом, что позволяет обмениваться IP-дейтаграммами, содержащими в себе пакеты транспортного протокола.

На *транспортном уровне* обеспечивается передача данных между программами. Здесь используются два вида протоколов: UDP, обеспечивающий доставку пакетов, не требующую подтверждения их доставки, и TCP, обеспечивающий гарантированную доставку пакетов. Протокол UDP используется там, где потеря нескольких пакетов не имеет значения (служба точного времени, online-вещание и т. п.). Протокол TCP применяется в том случае, если ни один байт пропасть не должен (обмен файлами, почтовыми сообщениями и т. п.).

Программы, предназначенные для пользователей, работают на *уровне приложений*. Среди наиболее популярных служб прикладного уровня выделяют:

- Web — служба предоставляет доступ к связанным гиперссылками Web-сайтам, содержащим страницы с текстовой информацией и графикой. Для передачи данных в Web используется протокол HTTP;
- электронная почта — программы электронной почты позволяют рассылать копии текстовых сообщений одному или нескольким клиентам. К текстовому сообщению может быть прикреплен один или более файлов произвольного формата. Для работы с электронной почтой используются протоколы SMTP и IMAP;
- пересылка файлов — служба позволяет обмениваться файлами. Для работы с этой службой применяется протокол FTP;
- удаленный доступ к компьютерам — служба позволяет подключиться к удаленной машине и установить с ней интерактивный сеанс связи. Для установки удаленного доступа в основном используется протокол SSH;
- мгновенный обмен сообщениями — позволяет двум машинам устанавливать соединение и обмениваться короткими сообщениями. В основном используется протокол ICQ.

Здесь приведены лишь наиболее популярные службы и протоколы, на самом деле прикладной уровень содержит огромное количество протоколов и служб, начиная с протоколов электронной коммерции, заканчивая специализированными протоколами для обновления программного обеспечения через Интернет.

Таким образом, данные перед пересылкой не только разбиваются на части, но и на каждом уровне упаковываются в соответствующий формат. Для передачи файлов и данных, отправляемых на уровне приложений, открывается соединение. Файл передается на транспортный уровень и разбивается на пакеты (если речь идет о протоколе TCP), далее на уровне протокола IP пакеты могут разбиваться еще раз (или не разбиваться) и упаковываться в IP-дейтаграммы. IP-дейтаграммы, в свою очередь, опять разбиваются и упаковываются в сетевые фреймы. Таким образом, данные на каждом из уровней упаковываются по принципу матрешки (рис. 1.2).
При получении данные извлекаются в обратном порядке и объединяются для получения конечного результата.

> ***Замечание***
>
> На момент написания книги, производилась модернизация корневых DNS-серверов, позволяющая наряду с протоколом версии IPv4 использовать протокол версии IPv6, в котором вместо четырех чисел XXX.YYY.ZZZ.WWW будут использоваться шесть чисел SSS.TTT.XXX.YYY.ZZZ.WWW. Такое расширение необходимо в связи с тем, что уже сейчас ощущается нехватка адресного пространства.

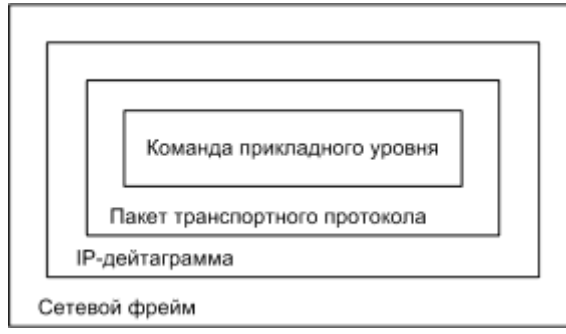


Рис. 1.2. Схема упаковки данных в семействе протоколов TCP/IP

Для того чтобы различать компьютеры в сети, каждому из них назначается IP-адрес, представляющий собой четыре числа (от 0 до 255), например, 127.0.0.1. Использовать IP-адрес для доступа к узлам сети неудобно, т. к. люди запоминают числа хуже названий. Кроме того, для обслуживания каждого сайта потребовался бы выделенный сервер и персональный IP-адрес. Так как сайтов очень много, а серверы достигли мощностей, когда они способны обслуживать сотни, а то и тысячи сайтов, такая ситуация является неприемлемой. В связи с этим в Интернете была введена служба доменных имен DNS, позволяющая установить соответствие между IP-адресом и *доменным именем*.

Например, при вводе в браузер доменного имени **mail.softtime.ru** сначала идет обращение не к серверу с сайтом, а к DNS-серверу, возвращающему IP-адрес для данного доменного имени, при помощи которого можно найти сервер в Интернете.

> ***Замечание***
>
> Назначение сайтам доменных имен позволяет безболезненно менять IP-адреса, а также располагать сайт на нескольких независимых серверах, находящихся в разных точках земного шара (что обеспечивает более быстрый доступ к сайту для участников сети).

Доменное имя представляет собой названия, разделенные точками. В доменном имени **mail.softtime.ru** часть **ru** выступает в качестве домена первого уровня, **softtime.ru** — в качестве домена второго уровня, а **mail.softtime.ru** в качестве домена третьего уровня и т. д. Домены первого уровня являются зарезервированными, часть из них предназначены для национальных сетей, например, **ru** предназначены для сайтов российского сегмента Интернета, **ua** — для украинского, **by** — белорусского и т. д. Часть доменов первого уровня предназначена для сайтов, посвященных определенному виду деятельности. Например, **com** — коммерческие компании, **org** — организации, **net** — сайты, предназначенные для развития Интернета, **gov** — правительственные учреждения, **travel** — сайты, посвященные туризму, и т. п.

> ***Замечание***
>
> Ценовая политика и порядок оформления доменного имени второго уровня определяется отдельно для каждой из зон. Если для регистрации сайтов в зонах **com**, **org** и **net** потребуются только ваши контактные данные, то для регистрации в зоне **ru** регистратор может потребовать паспортные данные. Следует учитывать, что доменные

имена второго уровня не продаются, а сдаются в аренду, если по истечении срока аренды доменное имя не будет оплачено, оно будет доступно для повторной регистрации (спустя 70 дней после окончания срока).

В каждой из зон можно зарегистрировать доменное имя второго уровня, а для доменов второго уровня можно зарегистрировать домены третьего уровня. При получении IP-адреса по доменному имени, корневому DNS-серверу отправляется домен первого уровня, который возвращает IP-адрес DNS-сервера обслуживающего эту зону, следующий DNS-сервер возвращает либо IP-адрес конечного сервера (если это домен второго уровня), либо следующий промежуточный DNS-сервер (если это домен третьего и более высокого уровня). Процедура повторяется до тех пор, пока не будет получен IP-адрес, соответствующий доменному имени.

Современные операционные системы поддерживают многозадачный режим, допускающий одновременное выполнение нескольких программ. Таким образом, данные через сеть получает процесс, запущенный на узле сети с определенным IP-адресом. Однако процессов, нуждающихся в сетевом взаимодействии, может быть запущено несколько, кроме того, они создаются и уничтожаются динамически. В связи с этим при создании сетевых соединений решено было ориентироваться не на процессы, а на функции, которые они реализуют. То есть клиент должен иметь возможность подключаться к серверу, не выясняя, в каком именно процессе реализованы функции сервера. В связи с этим были введены так называемые стандартные *порты* — целые положительные числа, за которыми закрепляется тот или иной протокол. Для протокола HTTP используется 80-й порт. Тот факт, что порт является стандартным, позволяет не указывать его при обращении к сайту, однако он может быть указан явно, например, **http://www.softtime.ru:80/index.php?id_forum=1** вместо **http://www.softtime.ru/forum/index.php?id_forum=1**. Более того, указание порта является обязательным, если сервер использует нестандартный порт, например, 8080 вместо 80. Для многих служб Интернета используются стандартные порты, например, протокол FTP использует порты 20 и 21 (один для передачи команд, другой для передачи данных), служба управления удаленным компьютером SSH закрепляет за собой порт 22, протокол электронной почты SMTP — порт 25, сервер доменных имен DNS — порт 53 и т. д.

Компьютеры в Интернете не равнозначные, часть из них используется как серверы, предназначенные для хранения данных, другие — для доступа к этим данным, третьи обеспечивают пересылку информации между сетями и т. д.

При работе с PHP мы будем взаимодействовать только с протоколами *прикладного уровня* (практически не вмешиваясь в работу более низкоуровневых протоколов), поэтому из всего многообразия компьютеров в Интернете нас будут интересовать серверы и клиентские машины.

Нас в первую очередь будут интересовать даже не компьютеры, а программы, установленные на них. *Сервер*, предоставляющий доступ к данным для клиентов, представляет собой сложную ресурсоемкую программу. Под *клиентом* обычно понимается программа, позволяющая получить доступ к серверу. Например, в качестве клиента Web-сервера выступает браузер, в качестве клиента почтового сервера — почтовый агент и т. п. Сами серверы также могут выступать в качестве клиентов, например, Web-сервер может выступать в качестве клиента базы данных, программа на PHP может выступать в качестве клиента другого Web-сервера.

1.3. Место и роль PHP в Интернете

В предыдущем разделе упоминалось, что PHP работает на прикладном уровне. Это означает, что с его помощью можно разрабатывать любые программы, использующие протоколы прикладного уровня, будь то Web-приложения, программы для отправки или получения почтовых сообщений, взаимодействие с FTP-сервером и т. п.

Web-программирование — это технология, включающая в себя множество компонентов, одним из которых выступает PHP. Прежде чем приступить к его подробному изучению, определим его место и роль в процессе обмена данными между клиентом и сервером. На рис. 1.3 представлена схема, демонстрирующая место и время действия компонентов типичной среды Web-разработки.

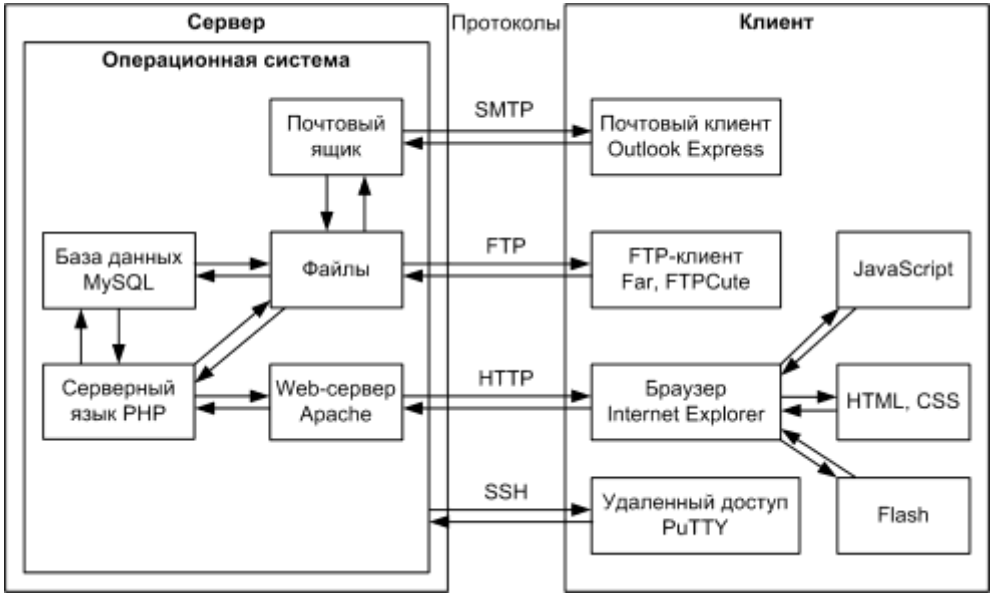


Рис. 1.3. Клиент-серверное взаимодействие

Как видно из схемы, помимо PHP в работе Web-приложений принимает участие множество дополнительных программ и технологий. Все они в той или иной степени будут затрагиваться в данной книге, т. к. PHP — всего лишь один из компонентов Web-технологии; для освоения всего процесса разработки Web-приложения необходимо понимать принципы работы всех компонентов, участвующих в схеме, представленной на рис. 1.3.

Схему взаимодействия клиента и сервера можно условно поделить на три части:

- серверные технологии — эти компоненты работают на сервере, вы не сможете запускать и использовать их на стороне клиента, если только последний сам не выступает в качестве сервера;
- протоколы — при помощи протоколов клиенты и серверы обмениваются информацией;

- клиентские технологии — Web-приложения по своей природе распределенные, часть работы выполняется на сервере, часть — лишь после того, как страница загружена клиентом.

1.3.1. Серверные технологии

UNIX-подобная операционная система

Все программы, будь то на сервере или на клиенте, работают под управлением операционной системы. В Интернете на серверах в качестве серверной операционной системы используется, как правило, UNIX-подобная операционная система (Linux, FreeBSD, Solaris и т. д.). Большинство пользователей Интернета (клиентов Web-сервера), по крайней мере, в Российской Федерации в качестве операционной системы использует Windows, поэтому при изучении Web-технологий основы UNIX будут крайне полезны, т. к. порядок работы в этих двух операционных системах различается достаточно сильно. Знания UNIX-подобной операционной системы не требуются на начальном этапе обучения, многие обходятся без основ UNIX, уже профессионально занимаясь Web-разработкой. Однако изучение этой операционной системы позволяет более уверенно чувствовать себя в мире Web-разработки, т. к. несмотря на кроссплатформенную независимость языка PHP, различие операционных систем Windows и UNIX дает о себе знать (особенно при работе с файловой системой). Кроме того, для многих параметров сервера не предусматривается Web-интерфейс, и ими можно управлять только через удаленный доступ (SSH), при работе через который необходимы глубокие знания команд операционной системы.

Web-сервер

За формирование Web-страниц на сервере несет ответственность Web-сервер. Существует несколько различных Web-серверов, наиболее известные — IIS-сервер операционной системы Windows и Apache, применяющийся главным образом на UNIX-серверах (впрочем, под управлением Windows Apache также прекрасно работает). При использовании PHP мы будем ориентироваться на Web-сервер Apache, как наиболее распространенный. Сервер Apache, помимо того что реализует протокол HTTP и позволяет обрабатывать обращения клиентов, имеет множество модулей, позволяющих преобразовывать URL-адреса, осуществлять переадресацию, подключать языки программирования (PHP, Perl и т. п.), защищать папки и файлы паролем и многое другое. Сервер Apache и его модули имеют множество директив, которые могут применяться как в основном конфигурационном файле, так и в специальном файле .htaccess, позволяющем настраивать работу файлов в отдельных папках. Изучение принципов работы и директив Web-сервера Apache позволят создавать более эффективные Web-приложения, а также решать множество задач в области управления сайтами, которые можно решить только на уровне Web-сервера.

Серверный язык

Следующим компонентом выступает серверный язык, позволяющий формировать динамические страницы. Практически любой серверный язык можно использовать для создания динамических страниц.

Замечание

На заре Интернета, когда язык программирования PHP еще не существовал, авторы данной книги для создания сайтов успешно использовали C++ и даже Fortran.

Однако наибольшее распространение получили специализированные языки, которые либо позволяли легко обрабатывать текст (Perl), либо имели встроенные средства для работы с Web-средой (ASP, PHP, Java). В нашей стране наибольшую популярность приобрел язык программирования PHP, рассматривающийся в данной книге.

Точно так же как и Web-сервер Apache, PHP имеет ядро, где сосредоточены базовые функции, а также множество подключаемых внешних модулей. Наиболее популярные и хорошо зарекомендовавшие себя модули поставляются в официальном дистрибутиве PHP, другие модули можно загрузить в дополнительном архиве или найти на многочисленных сайтах в Интернете. В любом случае язык программирования постоянно перестраивается, некоторые модули устаревают и исключаются из ядра и дистрибутива, другие, получившие популярность, — включаются. Поэтому язык PHP предоставляет собой чрезвычайно гибкий инструмент разработки, подстраивающийся под динамично развивающуюся Web-среду.

PHP — не самый быстрый язык программирования, однако один из самых удобных, что позволяет быстро разрабатывать Web-приложения. В Web-условиях, когда среднее время жизни приложения составляет полгода, а сами приложения постоянно подвергаются модификации, скорость разработки приложений становится решающим фактором. Ранее компьютерное время стоило дороже времени разработчиков (да и программы использовались значительно более долгий срок). В настоящий момент ситуация изменилась кардинально — приобрести быстрый сервер проще и дешевле, чем нанимать 10 программистов на C++, разрабатывающих приложение за полгода, вместо одного PHP-программиста, разрабатывающего приложение за один месяц.

Замечание

Широкая распространенность PHP вовсе не означает, что все остальные серверные языки следует игнорировать — чем больше языков программирования вы освоите, тем более конкурентоспособными вы будете и тем лучше освоите Web-программирование. Помимо PHP при Web-разработке часто используются Perl, семейство языков ASP.NET (VB, C#, ASP), Java, C/C++, Ruby и Python.

Файлы и базы данных

В любом языке программирования и технологии файлы играют значительную роль, в них хранятся как сами программы, так и данные, которые вводятся пользователями, обрабатываются скриптами и архивируются на сервере. Зачастую к серверу одновременно обращается огромное количество пользователей, что создает определенные трудности при записи информации в один и тот же файл, т. к. сразу несколько потоков стремятся получить доступ к файлам, что при игнорировании этой проблемы может приводить к повреждению записанной в него информации.

Для того чтобы каждый раз при разработке очередного Web-приложения не решать проблемы одновременного доступа к файлу, используются базы данных (или, точнее, системы управления базами данных, СУБД). СУБД разрабатываются десятки лет про-

фессионалами своей области с использованием быстрых компилируемых языков (C, C++). Поэтому СУБД в Web-приложениях всегда быстрее файлового доступа. Кроме того, код с использованием базы данных зачастую в несколько раз меньше по объему (а, следовательно, разрабатывается он в более короткие сроки), чем код, использующий файлы. Чем быстрее вы освоите приемы работы с СУБД, тем более быстродействующим и элегантным будет ваш код, и тем быстрее вы сможете его создавать.

Совместно с PHP может использоваться множество баз данных, начиная коммерческими Oracle и MS SQL, заканчивая свободно-распространяемыми MySQL, PostgreeSQL и FireBird. Все СУБД объединяет тот факт, что взаимодействие с ними осуществляется при помощи языка структурированных запросов SQL. Взаимодействие с СУБД сводится к выполнению запросов, построенных при помощи специализированного языка SQL. Язык SQL является общим для всех баз данных — освоив одну из баз данных, вы без труда сможете изучить любую другую.

> ***Замечание***
>
> Следует все же отметить, что, несмотря на стандартизацию языка SQL, разные базы данных имеют различные SQL-диалекты, иногда достаточно сильно отличающиеся друг от друга.

Одной из самых популярных баз данных, применяемых в Web-разработке, является СУБД MySQL, которая заслужила признательность разработчиков благодаря свободному распространению, высокому быстродействию и надежности, а также богатой функциональности.

Электронная почта

Электронная почта появилась задолго до Web-среды и является в настоящий момент неотъемлемой частью Интернета. Пользователи привыкли отправлять и получать почтовые сообщения. Язык программирования PHP имеет средства как для отправки почтовых сообщений, так и для доступа к почтовым ящикам (локальным или расположенным на удаленном сервере).

1.3.2. Клиентские технологии

Серверные и клиентские технологии разделены в пространстве и времени. PHP выполняется на сервере и формирует страницу, содержащую HTML-разметку, JavaScript-код, Flash-ролики, которые отправляются по протоколу HTTP клиенту. Такая страница интерпретируются браузером и отображается клиенту. В момент просмотра страницы клиентом, PHP, Web-сервер Apache уже отработали и отослали страницу. Заставить их среагировать на какие-то действия пользователя можно, только повторно послав запрос серверу. То же самое касается клиентских технологий, которые не выполняются на сервере — они вступают в действие, только когда страница получена клиентом.

Web-браузеры, HTML

Клиенты Web-серверов используют различные операционные системы и браузеры — программы, предназначенные для просмотра Web-страниц. Наиболее популярные среди них на сегодняшний день — Internet Explorer, FireFox и Opera. Одной из особенностей

является тот факт, что все браузеры (а также их различные версии) по-разному интерпретируют язык разметки HTML, каскадные таблицы стилей CSS и клиентский язык JavaScript. Это требует от Web-разработчиков тестирования сайтов в нескольких браузерах (разных версий), а также зачастую отказа от нововведений, которые поддерживаются одними браузерами и не поддерживаются другими.

Если в области клиентских технологий среди производителей браузеров существуют разногласия, то взаимодействие с серверной частью все браузеры поддерживают очень хорошо. Взаимодействие между браузерами и Web-серверами осуществляется при помощи протокола HTTP.

> ***Замечание***
>
> Существуют две версии протокола HTTP — 1.0 и 1.1. В настоящий момент везде (за исключением некоторых промежуточных прокси-серверов) используется протокол HTTP версии 1.1.

Каскадные таблицы стилей CSS и XML

Изначально HTML разрабатывался как язык разметки, т. е. язык, описывающий документ независимо от его внешнего вида. Однако очень скоро он стал применяться для создания дизайна, его конструкции стали использоваться для того, чтобы добиться того или иного внешнего эффекта. Под давлением разработчиков Web-страниц и браузеров в HTML вносились все новые и новые элементы. Начались браузерные войны, когда производители браузеров намеренно вносили дополнительные элементы, не совместимые с другими браузерами.

Для того чтобы разделить структуру и оформление документов, консорциумом W3C, координирующим развитие Web, были введен язык разметки XML и каскадные таблицы стилей CSS, при помощи которых можно оформить XML-код. Язык XML настолько гибок, что позволяет самостоятельно определить нужный XML-формат путем разработки собственного словаря. В настоящий момент уже разработано огромное число словарей XML, позволяющих описывать любую информацию — от химических реакций (CML, Chemical Markup Language) до финансовых данных (OFX, Open Financial Exchange). По сравнению с HTML XML является более гибким форматом; повсеместного применения XML еще не получил, но это лишь вопрос времени.

Каскадные таблицы стилей позволяют наложить дизайн на XML-документ. Пока до окончательного разделения структуры и оформления далеко, XML еще не очень популярен в Web в качестве языка разметки. Однако каскадные таблицы стилей применяются очень интенсивно в HTML. Если требуется создание стильных современных дизайнов, без каскадных таблиц стилей CSS не обойтись.

Flash-ролики

Технология Flash позволяет реализовывать на стороне клиента сложные динамические эффекты в виде Flash-роликов. По сути каждый Flash ролик представляет собой минисайт, позволяющий демонстрировать сложные графические эффекты, реализовывать игры и т. п. Внешний вид Flash-роликов не зависит от браузера и этим выгодно отличается от связки HTML и CSS. К недостаткам Flash можно отнести значительный размер,

необходимость наличия проигрывателя и невозможность индексирования информации роботами поисковых систем.

FTP-клиенты

Протокол HTTP позволяет как загружать файлы на сервер, так и скачивать их с сервера. При помощи серверных языков программирования создаются достаточно сложные Web-интерфейсы, позволяющие работать с файловой системой сервера через браузер. Однако такое взаимодействие не всегда является удобным, зачастую проще получить доступ к папке с файлами сайта на сервере через FTP-протокол и работать с этой папкой так, как если бы она была расположена на клиентской машине (загружать на сервер или скачивать с сервера файлы, создавать и переименовывать папки, выставлять права доступа). В подавляющем большинстве случаев, после того как сайт разработан, файлы, его составляющие, загружаются на удаленный сервер через FTP.

> ***Замечание***
>
> В качестве FTP-клиента часто выступают файловые менеджеры, такие как Far и Total Commander, однако существуют и специализированные FTP-менеджеры, такие как FTPCute, FileZila.

Удаленный доступ к серверу. Протокол SSH

Иногда просто получить доступ к папке на удаленном сервере (как это происходит в случае FTP) не достаточно, необходимо иметь возможность выполнять команды и работать на сервере так, как если бы монитор, клавиатура и мышь были подсоединены непосредственно к серверу. В этом случае в действие вступает протокол SSH. Из-под Windows доступ к удаленному серверу по SSH осуществляется при помощи утилиты PuTTY.



Глава 2

Быстрый старт

Несмотря на то, что PHP является универсальным языком программирования и может применяться для разработки практически любого программного обеспечения, основная его специализация — Web-разработка. Именно с этой целью он и проектировался, поэтому содержит множество инструментов для работы в Интернете.

> ***Замечание***
>
> В данной и последующих главах будем исходить из факта, что интерпретатор PHP и Web-сервер установлены у вас на локальной машине и связаны таким образом, что скрипты, помещенные в файл с расширением php, обрабатываются интерпретатором, и по запросу к Web-серверу в браузере выводится результат работы. Если у вас не настроена связка Web-сервера и PHP, вы можете обратиться к *приложениям 1* и *2*, в которых детально описывается процесс развертывания Web-среды в операционных системах Windows и UNIX.

2.1. Скрипты

Язык программирования PHP считается скриптовым языком, поэтому программы, написанные на нем, называют *скриптами*. Главное отличие традиционных программ от скриптов заключается в том, что скрипты работают только в определенной среде и используют ресурсы данной среды. Программы, будь то компилируемые или интерпретируемые, не зависят от какого-то стороннего программного обеспечения. Например, скриптовый язык программирования JavaScript работает только в Web-браузерах, Visual Basic for Applications — только в среде Microsoft Office. С использованием этих языков программирования невозможно создать программу, работающую без соответствующей среды. В случае PHP в качестве такой среды выступает Web-окружение (Web-сервер, сервер базы данных, почтовый сервер и т. п.).

> ***Замечание***
>
> Впрочем, язык программирования PHP допускает создание программ, работающих независимо от Web-сервера, однако в такой форме он не получил сколько бы то ни было широкого распространения.

Одной из главных особенностей языка программирования PHP является тот факт, что его код может располагаться вперемешку с HTML-кодом. Для того чтобы интерпретатор PHP различал HTML- и PHP-код, последний заключается в специальные теги `<?php` и `?>`, между которыми располагаются конструкции и операторы языка программирования

PHP. В листинге 2.1 приводится классический пример, выводящий в окно браузера при помощи конструкции `echo` фразу `"Hello, world!"` ("Привет, мир!"). Содержимое листинга 2.1 следует поместить в файл с расширением php, например в файл index.php. По умолчанию, файлы с другими расширениями не обрабатываются PHP-интерпретатором, и PHP-код остается необработанным. Впрочем, такое поведение Web-сервера можно изменить *(см. приложение 1)*.

> ***Замечание***
>
> `echo` — это конструкция языка программирования, позволяющая вывести одну или несколько строк в окно браузера. Более подробно конструкция `echo` обсуждается в *разд. 5.2*, посвященном строковым функциям.

Листинг 2.1. Простейший PHP-скрипт, index.php

```
<html>
  <head>
    <title>Пример</title>
  </head>
  <body>
    <?php
       echo "Hello, world!";
    ?>
  </body>
</html>
```

Результатом работы скрипта из листинга 2.1 в окно браузера будет выведена фраза `"Hello, world!"` (рис. 2.1).

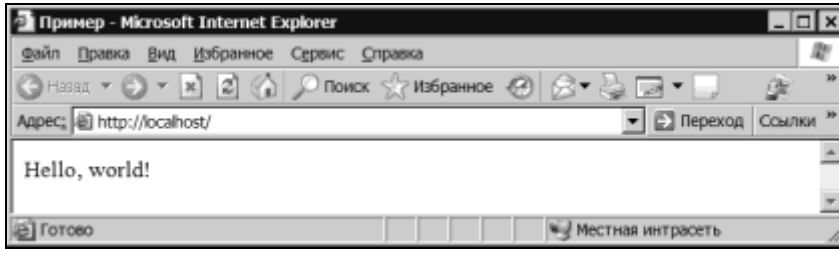


Рис. 2.1. Результат работы скрипта из листинга 2.1

При работе с серверными языками программирования, такими как PHP, следует помнить, что скрипты, расположенные между тегами `<?php` и `?>`, выполняются на сервере, клиенту приходит лишь результат работы PHP-кода, в чем можно легко убедиться, просмотрев исходный код HTML-страницы. В листинге 2.2 приводится результат работы PHP-скрипта из листинга 2.1, который браузер получает от сервера.

Замечание

Для просмотра исходного текста необходимо щелкнуть правой кнопкой мыши по странице и выбрать в контекстном меню соответствующий пункт, название которого зависит от браузера. В Internet Explorer необходимо выбрать пункт **Просмотр HTML-кода**, в Opera — **Исходный текст**, в FireFox — **View Page Source**.

Листинг 2.2. Исходный код HTML-страницы, которую получает браузер

```
<html>
  <head>
    <title>Пример</title>
  </head>
  <body>
    Hello, world!  </body>
</html>
```

Если связка PHP и Web-сервера настроена неправильно или PHP-скрипт был размещен в файле с расширением, отличным от php, например, в файле index.html, то в исходном коде HTML-страницы вместо содержимого листинга 2.2 можно увидеть содержимое листинга 2.1. В этом случае все, что расположено между тегами `<?php` и `?>`, будет рассматриваться браузером как один большой неизвестный ему HTML-тег, а окно браузера будет пустым (рис. 2.2).

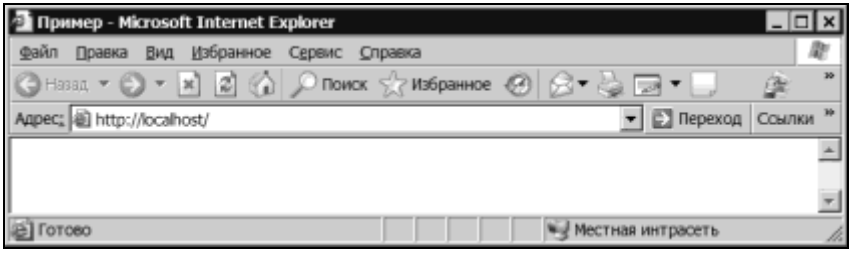


Рис. 2.2. PHP-код не подвергся интерпретации

2.2. Начальные и конечные теги

Как было указано в предыдущем разделе, PHP-скрипт должен быть размещен между начальным тегом `<?php` и конечным тегом `?>` для того, чтобы интерпретатор мог разделить HTML- и PHP-коды. Даже если HTML-код не используется, указание PHP-тегов является обязательным, в противном случае PHP-код будет выведен в окно браузера как есть, без интерпретации. Помимо тегов `<?php` и `?>` PHP поддерживает еще ряд альтернативных тегов, представленных в табл. 2.1.

Замечание

До версии PHP 6.0 поддерживались начальные и конечные теги <% и %>, характерные для ASP-скриптов, включить которые можно было в конфигурационном файле php.ini, назначив директиве asp_tags значение On *(см. главу 17)*. В версии PHP 6.0 данный тип тегов был исключен.

Таблица 2.1. *Варианты PHP-тегов*

Вариант тегов	Описание
<?php ... ?>	Классический вариант PHP-тегов, именно его рекомендуется использовать в повседневной практике
<? ... ?>	Краткий вариант PHP-тегов, работает только в том случае, если в конфигурационном файле php.ini включена (т. е. принимает значение On) директива short_open_tag. По умолчанию эта директива включена, однако все-таки лучше ориентироваться на полный вариант PHP-тегов <?php ... ?>. Последнее особенно актуально при использовании PHP совместно с XML-кодом, который использует теги <?xml ... ?>. При использовании коротких тегов может возникнуть неоднозначность в интерпретации
<script language="php"> ... </script>	Расширенный вариант PHP-тегов, так же как и вариант <?php ... ?> доступный в любой момент без дополнительных настроек конфигурационного файла php.ini. Данный вариант несколько громоздок и распознается не всеми PHP-редакторами, тем не менее, следует быть готовыми встретить и такой вариант в PHP-скриптах
<?= ... ?>	Специальный вид тегов, предназначенных для вывода простого выражения в окно браузера. Конструкция <?= ... ?> эквивалентна <?php echo ... ?>

Отдельно следует отметить конструкцию <?= ... ?>, которую удобно использовать, когда скрипт состоит из одного выражения. Например, скрипт из листинга 2.1 можно было бы переписать так, как это продемонстрировано в листинге 2.3.

Листинг 2.3. Использование тегов <?= ... ?>

```
<html>
  <head>
    <title>Пример</title>
  </head>
  <body>
    <?= "Hello, world!" ?>
  </body>
</html>
```

Следует отметить, что HTML-страница может содержать более одной PHP-вставки. В листинге 2.4 приводится пример, который содержит две вставки: одна задает назва-

ние страницы (в HTML-теге `<title>`), а вторая определяет содержимое страницы (в HTML-теге `<body>`).

> ***ЗАМЕЧАНИЕ***
>
> В листинге 2.4 используется функция `date()`, которая возвращает текущую дату и время. Более подробно эта функция обсуждается в *главе 11*, посвященной функциям даты и времени.

Листинг 2.4. Допускается несколько PHP-вставок в HTML-код

```
<html>
  <head>
    <title><?php echo "Вывод текущей даты" ?></title>
  </head>
  <body>
    <?php
      echo "Текущая дата:<br>";
      echo date(DATE_RSS);
    ?>
  </body>
</html>
```

Результат работы скрипта из листинга 2.4 представлен на рис. 2.3.

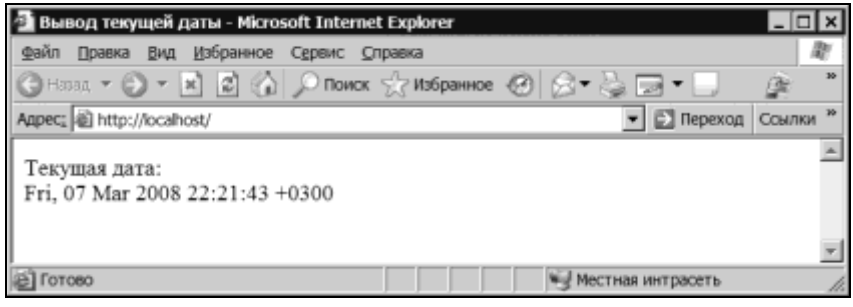


Рис. 2.3. Результат работы скрипта из листинга 2.4

2.3. Использование точки с запятой

Совокупность конструкций языка программирования, завершающуюся точкой с запятой, будем называть *выражением*.

Как видно из листинга 2.4, после строки "Вывод текущей даты" не указывается точка с запятой. Выражение одно, и надобности разделять его между другими выражениями

нет. Однако, как можно видеть во второй вставке, в конце каждой из конструкций `echo` имеется точка с запятой. Если забыть указать этот разделитель, интерпретатор языка программирования PHP посчитает выражение на новой строке продолжением предыдущего и не сможет корректно разобрать скрипт. В результате будет сгенерировано сообщение об ошибке "Parse error: syntax error, unexpected T_ECHO, expecting ',' or ';'" ("Ошибка разбора: синтаксическая ошибка, неожиданно встречена конструкция echo, ожидается либо запятая ',', либо точка с запятой ';'").

Последнее выражение перед завершающим тегом `?>` может не снабжаться точкой с запятой не только в тегах `<?= ... ?>`, но и во всех остальных вариантах, рассмотренных в табл. 2.1. Например, в листинге 2.4 после выражения `echo date(DATE_RSS)` точку с запятой можно не указывать. Однако настоятельно рекомендуется не пользоваться этой особенностью и помещать точку с запятой после каждого выражения, т. к. добавление новых операторов может привести к появлению трудно улавливаемых ошибок.

Переводы строк никак не влияют на интерпретацию скрипта, выражение может быть разбито на несколько строк — интерпретатор PHP будет считать, что выражение закончено лишь после того, как обнаружит точку с запятой или завершающий тег `?>`. В листингах 2.5 и 2.6 представлены два скрипта, аналогичные по своей функциональности.

Листинг 2.5. Использование точки с запятой

```
<?php
  echo 5 + 5;
  echo 5 - 2;
  echo "Hello, world!";
?>
```

Листинг 2.6. Альтернативная запись скрипта из листинга 2.5

```
<?php
echo(5
     +
     5); echo(5-
     2); echo("Hello, world!"
     );
?>
```

Замечание

Следует избегать конструкций, подобной той, которая приведена в листинге 2.6. Понятно написанный код — залог успеха, т. к. такой код намного проще и приятнее отлаживать. А искусство программиста, как известно, состоит не в безошибочном написании кода (ошибки делают все), а в умении его отлаживать.

2.4. Составные выражения. Фигурные скобки

Фигурные скобки позволяют объединить несколько выражений в группу, которую обычно называют *составным выражением* (листинг 2.7).

Листинг 2.7. Составное выражение

```
<?php
  {
    echo 5 + 5;
    echo 5 - 2;
    echo "Hello, world!";
  }
?>
```

Само по себе составное выражение практически никогда не используется, основное его предназначение — работа совместно с условными операторами, операторами цикла и т. п., которые подробно рассматриваются в *главе 5*.

Составное выражение может быть расположено в нескольких PHP-вставках. В листинге 2.8 приводится пример двух составных выражений, которые разбиты несколькими PHP-вставками. Задача скрипта сводится к случайному выводу в окно браузера либо слова "Истина" зеленого цвета, либо слова "Ложь" красного цвета. Без использования фигурных скобок оператор `if` распространял бы свое действие только на одно выражение, использование составного выражения позволяет распространить его действие на несколько простых выражений.

> ***ЗАМЕЧАНИЕ***
>
> Логический оператор `if` рассматривается в *главе 5*, а функция `rand()`, генерирующая случайное число из заданного интервала, — в *главе 12*.

Листинг 2.8. Составное выражение может располагаться в нескольких PHP-вставках

```
<?php
  if (rand(0,1))
  {
    ?>
    <div style='color:green'>
    <?php
      echo "Истина";
    ?>
    </div>
    <?php
  }
  else
  {
```

```
    ?>
    <div style='color:red'>
    <?php
      echo "Ложь";
    ?>
    </div>
    <?php
  }
?>
```

Как видно из листинга 2.8, составное выражение в любой момент может быть прервано тегами `<?php` и `?>`, а затем продолжено. Впрочем, существуют исключения, например, составное выражение, применяемое для формирования класса, нельзя разбивать тегами `<?php` и `?>`. Возможно, в будущих версиях языка это ограничение будет устранено.

2.5. Комментарии

Код современных языков программирования является достаточно удобным для восприятия человеком по сравнению с машинными кодами, ассемблером или первыми языками программирования высокого уровня. Тем не менее, конструкции языка продиктованы архитектурой компьютера и, создавая программы, разработчик волей-неволей использует компьютерную, а не человеческую логику. Это зачастую приводит к созданию достаточно сложных построений, которые нуждаются в объяснении на обычном языке. Для вставки таких пояснений в код предназначены *комментарии*.

PHP предоставляет несколько способов для вставки комментариев, варианты которых представлены в табл. 2.2.

***Таблица 2.2.** Варианты комментариев*

Комментарий	Описание
`// ...`	Комментарий в стиле языка C++, начинающийся с символа двух слешей `//` и заканчивающийся переводом строки
`# ...`	Комментарий в стиле скриптовых языков UNIX, начинающийся с символа диеза `#` и заканчивающийся переводом строки
`/* ... */`	Если два предыдущих комментария ограничены лишь одной строкой, то комментарий в стиле языка C `/* ... */` является многострочным

В листинге 2.9 демонстрируется использование всех трех видов комментариев из табл. 2.2.

Листинг 2.9. Комментарии

```
<?php
  echo "Hello"; // это комментарий
  echo "Hello"; # и это комментарий
```

```
  /*
    а это многострочный
    комментарий
  */
?>
```

Естественно, что комментарии PHP действуют только внутри тегов ограничителей `<?php ... ?>`. То есть если символы комментариев будут находиться вне тегов-ограничителей, то они, как и любой текст, будут отображены браузером (листинг 2.10).

Листинг 2.10. Комментарии действуют только внутри тегов `<?php` и `?>`

```
<?php
  echo "Hello<br>"; // нормальный комментарий
?>
// этот текст отобразится браузером.
<!-- Этот текст не будет отображен браузером, поскольку заключен между
символами, являющимися комментариями  HTML. Однако он может быть просмотрен
в исходном коде HTML-страницы -->
```

Результат работы скрипта из листинга 2.10 представлен на рис. 2.4.

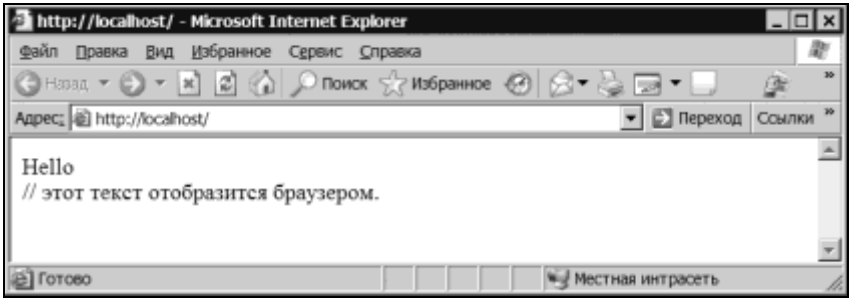


Рис. 2.4. Результат работы скрипта из листинга 2.10

Комментарии можно вставлять не только в конце выражения после точки с запятой, но и посередине. Так скрипт, представленный в листинге 2.11, является вполне корректным.

Листинг 2.11. Комментарий в списке аргументов функции

```
<?php
  echo strstr( // эту функцию мы рассмотрим позднее
              "Hello, world", "H");
?>
```



Глава 3

Переменные и типы данных

Ключевым объектом практически любого языка программирования является переменная. Под *переменной* в общем случае понимается именованная область памяти. В этой области может храниться либо строка, либо число, которыми можно манипулировать при помощи имени переменной, которое мы далее для простоты будем называть просто переменной. То, что хранится в области памяти, будем называть *значением переменной*.

3.1. Объявление переменной. Оператор =

В PHP переменные начинаются со знака доллара (`$`), за которым может следовать любое количество буквенно-цифровых символов и символов подчеркивания, но первый символ не может быть цифрой. Таким образом, допустимы следующие имена переменных: `$n`, `$n1`, `$user_func_5` и т. д. В отличие от ключевых слов, имена переменных в PHP чувствительны к регистру, т. е. переменные `$user`, `$User` и `$USER` являются различными (листинг 3.1).

Листинг 3.1. Зависимость переменных от регистра

```
<?php
  $user = "Владимир";
  $User = "Дмитрий";
  $USER = "Юрий";
  echo $user; // Владимир
  echo $User; // Дмитрий
  echo $USER; // Юрий
?>
```

Как видно из листинга 3.1, для присвоения значения переменной необходимо воспользоваться оператором присвоения =, который позволяет инициализировать переменную.

При объявлении числовых значений в качестве разделителя целого значения и дробной части выступает точка (листинг 3.2).

Листинг 3.2. Объявление числовых переменных

```
<?php
  $number = 1;
  $var = 3.14;
?>
```

Допускается инициализация одним значением сразу несколько переменных за счет того, что оператор "равно" = возвращает результат присвоения. В листинге 3.3 переменным `$num`, `$number` и `$var` присваивается значение 1 в одну строку за счет использования оператора "равно" = в цепочке.

Листинг 3.3. Инициализация трех переменных одним значением

```
<?php
  $num = $number = $var = 1;
?>
```

3.2. Типы данных

Язык PHP является слабо типизированным, т. е. переменные языка не требуют строгого задания типа при их объявлении, а в ходе выполнения программы тип переменной может быть практически всегда изменен неявным образом без специальных преобразований. Например, переменная, объявленная строкой, может использоваться далее в арифметических операциях, выступать как логическая переменная, а в конце ей в качестве значения может быть присвоен объект. Все это позволяет разработчику практически не задумываться о типах данных. Тем не менее, некоторые ограничения на типы переменных все-таки накладываются, поэтому о типах переменных все же следует иметь представление. В табл. 3.1 представлены типы переменных, которые поддерживаются PHP.

Таблица 3.1. *Типы переменных PHP*

Тип переменной	Описание
`boolean`	Логический тип, способный принимать лишь два значения: `true` (истина) и `false` (ложь). Более подробно данный тип обсуждается в *главе 5*
`integer`	Целое число, разрядность которого зависит от архитектуры компьютера. Если разрядность операционной системы составляет 32 бита, то число может принимать значения от –2 147 483 648 до 2 147 483 647. Если разрядность составляет 64 бита, то число может принимать значение от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
`int64`	Целое число, разрядность которого не зависит от архитектуры компьютера и всегда равна 64 битам. Таким образом, переменные данного типа могут принимать значения от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

Таблица 3.1 *(окончание)*

Тип переменной	Описание
`double` (или `float`)	Вещественное число, минимально возможное значение которого составляет $\pm1.7\times10^{-308}$, а максимально возможное $\pm1.7\times10^{308}$
`string`	Строковый тип, может хранить строку произвольного объема
`array`	Массив — это объединение нескольких однотипных переменных под одним именем. Обращаться к отдельным переменным можно при помощи индекса или ключа массива. Более подробно массивы обсуждаются в *главе 6*
`object`	Объект — это конструкция, объединяющая несколько разнотипных переменных и методы их обработки. Более подробно объекты обсуждаются в *главах 21—26*
`resource`	Дескриптор, позволяющий оперировать тем или иным ресурсом, доступ к которому осуществляется при помощи библиотечных функций. Дескрипторы применяются при работе с файлами, базами данных, динамическими изображениями и т. д. Более подробно дескрипторы будут рассмотрены в соответствующих главах
`NULL`	Специальный тип, который сигнализирует о том, что переменная не была инициализирована

3.3. Целые числа

Целые числа являются наиболее распространенными в программировании. Это связано с тем, что большинство прикладных задач носит арифметический характер, а также с тем, что это наиболее быстродействующий тип данных. Если для обработки плавающих чисел используется математический сопроцессор с достаточно сложными алгоритмами их обработки, то оперировать целыми числами процессор может напрямую.

> ***Замечание***
>
> В отличие от других языков программирования переполнение (т. е. выход значения за допустимые границы) в PHP не бывает; если полученное значение не убирается в `integer`, для него автоматически выбирается больший тип данных (`int64` или `double`).

Язык PHP является C-подобным, поэтому целочисленная переменная может быть объявлена несколькими способами (листинг 3.4).

Листинг 3.4. Объявление целочисленных переменных

```
<?php
  $num = 1234; // десятичное число
  $num = +123; // десятичное число
  $num = -123; // отрицательное число
  $num = 0123; // восьмеричное число (эквивалентно 83)
  $num = 0x1A; // шестнадцатеричное число (эквивалентно 26)
?>
```

Целое положительное число объявляется, как правило, без указания плюса перед ним (впрочем, указанием символа плюса (+) не приводит к ошибке). Для объявления отрицательного числа перед ним размещается символ минуса (-).

По умолчанию считается, что числа задаются в десятичной систем счисления, однако PHP позволяет объявлять переменные в восьмеричной и шестнадцатеричной системах счисления. В восьмеричной системе счисления основанием служит число 8, а для выражения всех чисел используются цифры от 0 до 7. Число 8 в восьмеричной системе счисления имеет то же значение, что число 10 в десятичной. Шестнадцатеричная система счисления (основанием служит число 16) использует цифры от 0 до 9 и буквы английского алфавита от A до F, означающие шестнадцатеричные "цифры" 10, 11, 12, 13, 14 и 15. Числа, объявленные в восьмеричной системе счисления, предваряются префиксом 0, в шестнадцатеричной системе — префиксом 0x.

ЗАМЕЧАНИЕ

Может показаться, что восьмеричные и шестнадцатеричные числа являются избыточным наследием языка C, однако, как будет показано дальше, они применяются и в Web-разработке, например, при задании прав доступа к файлам и папкам.

3.4. Вещественные числа

Вещественные числа (`float` или `double`) позволяют сохранять данные в огромном интервале, за пределы которого прикладные программы не выходят практически никогда. Различают две формы записи вещественного числа: стандартную и научную (листинг 3.5).

ЗАМЕЧАНИЕ

При выводе под число с плавающей точкой отводится 12 символов, это значение может быть изменено при помощи директивы `precision` в конфигурационном файле php.ini *(см. главу 20)*.

Листинг 3.5. Объявление вещественных чисел

```
<?php
  // Объявление положительного вещественного числа
  // Стандартная запись
  $var = 1.23456;
  // Объявление отрицательного вещественного числа
  // Стандартная запись
  $var = +1.23456;
  // Объявление положительного вещественного числа больше единицы
  // Научная запись
  $var = 1.23456E2; // 123.456
  // Объявление положительного вещественного числа больше единицы
  // Научная запись
```

```
  $var = 1.23456E+2; // 123.456
  // Объявление положительного вещественного числа меньше единицы
  // Научная запись
  $var = 1.23456E-3; // 0.00123456
  // Объявление отрицательного вещественного числа меньше 0 и больше -1
  // Научная запись
  $var = -1.23456e-2; // -0.0123456
?>
```

Научная запись помимо мантиссы (целой и дробной частей) содержит порядок, который начинается со строчной или прописной буквы E и целого положительного (или отрицательного) числа. Так запись `1.2e-3` эквивалентна произведению 1.2×10^{-3} (или 0.0012). Вещественное число `1.1e+2` (или `1.1e2`) эквивалентно 1.1×10^{2} (или 110.0).

Вещественные числа преобразуются в компьютерное представление с небольшими потерями. Это приводит к тому, что при длительных преобразованиях могут накапливаться ошибки вычисления, например число 1.3 может принимать форму 1.29999999... Это связано с тем, что некоторые дроби в десятичной системе счисления невозможно представить конечным числом цифр. Так дробь 1/3 в десятичной форме принимает периодическую форму 0.33333333..., и часть цифр приходится отбрасывать. Такое поведение вещественных чисел следует учитывать в программах, особенно при сравнении их (чисел) друг с другом *(см. главу 5)*.

3.5. Строки

Строки могут принимать произвольный размер, однако не превышающий объем памяти, отводимый под PHP-скрипт. Этот размер, равный, как правило, 8 или 16 Мбайт, определяется при помощи директивы `memory_limit` конфигурационного файла php.ini. Объявление строк осуществляется при помощи кавычек, использование которых подробно обсуждается в следующем разделе.

3.6. Кавычки

Как уже было продемонстрировано в предыдущей главе, строки и строковые переменные формируются путем заключения той или иной фразы в кавычки. До сих пор использовались только двойные кавычки, однако в PHP имеется возможность использовать несколько кавычек, каждый вид которых имеет собственные особенности (табл. 3.2).

Таблица 3.2. *Типы кавычек*

Кавычки	Описание
`" ... "`	Двойные кавычки — вместо переменных PHP в этих кавычках подставляются их значения

Таблица 3.2 *(окончание)*

Кавычки	Описание
' ... '	Одиночные кавычки — вместо переменных PHP не подставляются их значения, символ $ отображается как есть
` ... `	Обратные кавычки — значение, заключенное в такие кавычки, рассматривается как системная команда, вместо такой системной команды возвращается результат выполнения команды

В языках программирования обычно поддерживают два варианта кавычек: одиночные и двойные, для того чтобы при необходимости применять одиночные кавычки для обрамления двойных (листинг 3.6).

Листинг 3.6. Двойные кавычки

```
<?php
  echo "Переменная принимает значение '345'";
?>
```

А двойных — для обрамления одиночных (листинг 3.7).

Листинг 3.7. Одинарные кавычки

```
<?php
  echo 'Проект "Бездна" - самый дорогой проект в истории...';
?>
```

В PHP функциональность кавычек была расширена (вернее, заимствована из Perl). Так, если поместить в двойные кавычки переменную, то ее значение будет подставлено в текст (листинг 3.8). Такая подстановка называется *интерполяцией*.

Листинг 3.8. Подстановка переменной

```
<?php
  $str = 123;
  echo "Значение переменной - $str"; // Значение переменной - 123
?>
```

Иногда требуется подавить такое поведение переменных, для этого применяется экранирование символа $ обратным слешем, как это показано в листинге 3.9.

Листинг 3.9. Экранирование символа $

```
<?php
  $str = 123;
  echo "Значение переменной - \$str"; // Значение переменной - $str
?>
```

Точно так же можно применять *экранирование* для размещения двойных кавычек в строке, обрамленных двойными же кавычками (листинг 3.10).

Листинг 3.10. Экранирование двойных кавычек

```
<?php
  echo "Проект \"Бездна\" - самый дорогой проект в истории...";
?>
```

Применение обратного слеша с рядом других символов интерпретируется особым образом. Список наиболее часто используемых символов и соответствующие им значения представлены в табл. 3.3.

Таблица 3.3. *Специальные символы и их значение*

Значение	Смысл
\n	Перевод строки
\r	Возврат каретки
\t	Символ табуляции
\\	Обратный слеш
\"	Двойная кавычка
\'	Одинарная кавычка

Размещение переменных и специальных символов (за исключением \') в одиночных кавычках не приводит к их специальной интерпретации (листинг 3.11).

Листинг 3.11. В одиночных кавычках переменные не интерполируются

```
<?php
  $str = 123;
  echo 'Значение переменной - $str'; // Значение переменной - $str
?>
```

Иногда при размещении переменной внутри строки требуется точно указать границы переменной. Для этого имя переменной, значение которой следует подставить в строку, обрамляют фигурными скобками (листинг 3.12).

Замечание

Интерполяция элементов массивов более подробно рассматривается в *разд. 6.4*.

Листинг 3.12. Применение фигурных скобок для указания границ переменных

```
<?php
  // Выставляем уровень тревожности интерпретатора
  error_reporting(E_ALL & ~E_NOTICE & ~E_STRICT);
```

```
  // Подставляем переменную $text в строки, обрамленные двойными кавычками
  $text = "Паро";
  echo "Едет $textвоз";      // Неправильно. Выведет только "Едет"
  echo "Плывет $textход";    // Неправильно. Выведет только "Плывет"
  echo "Едет {$text}воз";    // Выведет "Едет Паровоз"
  echo "Плывет {$text}ход";  // Выведет "Плывет Пароход"
?>
```

В первом случае PHP, встретив знак доллара (`$`), ищет пробел и считает переменными последовательности `$textвоз` и `$textход` (которые рассматриваются как неинициализированные переменные), во втором случае мы явно указываем границы переменных.

PHP унаследовал от Perl третий вид кавычек — так называемые обратные кавычки (`` ` ``). Заключенные в них строки воспринимаются как команды операционной системы, которые выполняются, а все, что команда печатает в стандартное устройство вывода, возвращается скрипту (листинг 3.13).

Листинг 3.13. Использование обратных кавычек

```
<?php
  // echo `ls -l`; // UNIX
  echo `dir`;      // Windows
?>
```

Результат работы скрипта

```
 Том в устройстве E имеет метку MEDIUM
 Серийный номер тома: EC68-EF90

 Содержимое папки E:\main

25.03.2004  20:41    <DIR>          .
25.03.2004  20:41    <DIR>          ..
23.10.2004  00:19               411 config.php
24.07.2004  12:43                54 index.php
03.07.2004  12:55             1 815 main.php
18.09.2003  19:43             2 789 top.php
25.03.2004  20:42    <DIR>          images
11.04.2004  16:30    <DIR>          Projects
25.03.2004  20:43    <DIR>          admin
30.03.2004  22:17    <DIR>          scripts
04.04.2004  12:27    <DIR>          Books
25.04.2004  12:52    <DIR>          tools
12.06.2004  09:46    <DIR>          test
26.09.2004  12:53    <DIR>          Site
               4  файлов          5 069 байт
              10 папок  18 157 846 528 байт свободно
```

3.7. Оператор <<<

Объявить строку можно, так же не прибегая к кавычкам. Для этого используется последовательность <<<. Сразу после данной последовательности размещается метка, конец оператора обозначается повторным вхождением метки (листинг 3.14).

> ***Замечание***
>
> Метка является зависимой от регистра. Традиционно вводится при помощи прописных букв, несмотря на то, что использование символов в нижнем регистре не запрещается.

Листинг 3.14. Использование последовательности <<<

```
<?php
$str = <<< HTML_END
Здесь располагается любой текст, до тех пор, пока не встретится
метка, можно писать все, что угодно
HTML_END;
echo $str;
?>
```

Важно отметить, что после первой метки `HTML_END` не должно быть никаких пробелов, лишь перевод строки, как, впрочем, и перед последней меткой `HTML_END`. Последовательность <<< традиционно применяют для ввода объемного текста, который неудобно вводить при помощи традиционных кавычек.

> ***Замечание***
>
> Следует осторожно использовать последовательность <<< для создания строк, т. к. допустить ошибку при формировании метки достаточно просто, например, добавив лишний пробел, а обнаружить ее бывает не легко.

3.8. Обращение к неинициализированной переменной. Замечания (Notice)

В отличие от большинства языков программирования, в PHP переменная создается сразу же при первом обращении. В ранних версиях языка это поощряло создание множества трудноулавливаемых ошибок и дыр в безопасности. Поэтому разработчиками принято решение ужесточить требование к переменным. Переменная по-прежнему создается при первом обращении к ней, однако, если она не была инициализирована при помощи оператора присваивания =, генерируется замечание (Notice). В листинге 3.15 приводится обращение к неинициализированной переменной `$user`, которое приводит к генерации замечания.

Листинг 3.15. Обращение к неинициализированной переменной `$user`

```
<?php
  echo $user;
?>
```

В результате обращения к неинициализированной переменной `$user` генерируется замечание "Notice: Undefined variable: user" ("Замечание: неопределенная переменная user") (рис. 3.1).

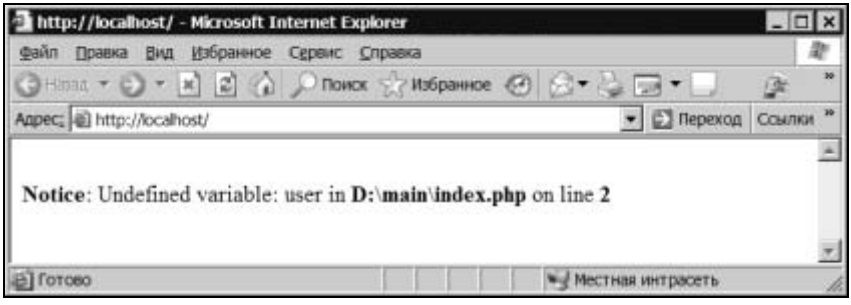


Рис. 3.1. Замечание, генерирующееся при обращении к неопределенной переменной

Такое поведение скрипта не всегда удобно, и любому разработчику предоставляется возможность самостоятельно настроить реакцию интерпретатора PHP. За чувствительность PHP к таким ситуациям несет ответственность директива `error_reporting` конфигурационного файла php.ini.

Замечание

Более подробно уровни обработки ошибок рассматриваются в *разд. 24.1*.

При настройке PHP можно создать файл php.ini на основе заготовки php.ini-dist, которая больше подходит для рабочих серверов, или файла php.ini-recommended, которым рекомендуется пользоваться разработчику. В файле php.ini-dist значение директивы `error_reporting` выставлено в `E_ALL & ~E_NOTICE`, означающее, что выводятся все сообщения об ошибках за исключением замечаний:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

В файле php.ini-recommended директива `error_reporting` выводит все возможные сообщения:

```
error_reporting  =  E_ALL
```

Следует отметить, что, начиная с PHP 6.0, в значение `E_ALL` добавлено значение `E_STRICT` (которое ранее туда не входило). Сообщения `E_STRICT` генерируются, когда интерпретатором PHP замечается неудачное использование той или иной конструкции языка, не являющееся запрещенным, однако не рекомендуемое к использованию. Эти сообщения могут быть крайне навязчивыми, особенно при работе с большим объемом

старого кода, однако в любой момент они могут быть отключены исправлением директивы `error_reporting` следующим образом:

```
error_reporting  =  E_ALL & ~E_NOTICE & ~E_STRICT
```

Иногда доступ к конфигурационному файлу php.ini затруднен, в этом случае можно управлять директивой `error_reporting` при помощи функции `error_reporting()`. Скрипт в листинге 3.16 не выдает никаких сообщений, несмотря на то, что имеется обращение к неинициализированной переменной.

> ***ЗАМЕЧАНИЕ***
>
> Функция `error_reporting()` более подробно рассматривается в *главе 24*.

Листинг 3.16. Управление режимом вывода сообщений при помощи `error_reporting()`

```
<?php
  // Выставляем уровень тревожности интерпретатора
  error_reporting(E_ALL & ~E_NOTICE & ~E_STRICT);
  // Обращаемся к неинициализированной переменной
  echo $user;
?>
```

> ***ЗАМЕЧАНИЕ***
>
> Язык программирования PHP предоставляет разработчику возможность подавлять вывод замечаний, предупреждений и сообщений об ошибках, не прибегая к изменению значения директивы `error_reporting`. Можно подавить вывод для конкретного выражения при помощи оператора подавления ошибок `@`, который рассматривается в *разд. 5.14*.

3.9. Специальный тип *NULL*

Специальный тип `NULL` предназначен для пометки неинициализированной переменной. Если интерпретатор PHP встречает в выражении неинициализированную переменную (как это описывается в *разд. 3.8*), она получает тип `NULL`. Переменная также получает данный тип, если она инициализируется константой типа `NULL` (листинг 3.17) или уничтожается при помощи конструкции `unset()` *(см. разд. 3.11)*.

> ***ЗАМЕЧАНИЕ***
>
> Константа `NULL` не зависит от регистра. Константы более подробно обсуждаются в *главе 4*.

Листинг 3.17. Использование константы `NULL`

```
<?php
  $user = NULL;
?>
```

Следует отметить, что при инициализации переменной при помощи константы NULL и последующем обращении к переменной в выражениях не происходит генерации замечания "Notice: Undefined variable". Такое поведение несколько нелогично, т. к. во всех остальных случаях, когда переменная принимает значение NULL, генерация замечания происходит. Возможно, это будет исправлено в последующих версиях PHP.

3.10. Логический тип

Переменные логического типа boolean принимают только два значения: TRUE (истина) или FALSE (ложь). В листинге 3.18 приводится объявление логической переменной, принимающей значение TRUE.

> ***Замечание***
>
> Константы TRUE и FALSE не зависят от регистра. Константы более подробно обсуждаются в *главе 4*.

Листинг 3.18. Объявление переменной логического типа

```
<?php
  $bool = TRUE;
?>
```

Переменные логического типа интенсивно используются совместно с операторами сравнения и цикла, которые более подробно обсуждаются в *главе 5*.

3.11. Уничтожение переменной. Конструкция *unset()*

Переменная может быть уничтожена при помощи конструкции unset(), которая имеет следующий синтаксис:

```
unset($var [, $var1 [, ...]])
```

В круглых скобках конструкции unset() указывается либо одна, либо несколько разделенных запятой переменных, которые подлежат уничтожению. После вызова конструкции unset() память, выделенная под значение переменной, возвращается системе, а самой переменной присваивается значение NULL. В листинге 3.19 демонстрируется использование конструкции unset().

> ***Замечание***
>
> Конструкция unset() может применяться ко всем типам данных, включая массивы и объекты.

Листинг 3.19. Использование конструкции `unset()`

```
<?php
  // Объявляем переменные
  $user = "Юрий";
  $number = 123;
  // Уничтожаем переменные
  unset($user, $number);
  // Попытка обращения к несуществующей переменной
  echo $user; // Генерируется "Notice: Undefined variable"
?>
```

Уничтожение ненужных переменных может быть полезно, когда скрипт оперирует объемными данными (например, содержимым файлов), и их размер грозит превысить объем памяти, выделяемой на один скрипт *(см. разд. 3.5)*.

3.12. Проверка существования переменной. Конструкции *isset()* и *empty()*

Как видно из предыдущих разделов, в произвольной точке скрипта переменная может оказаться неинициализированной или быть уничтоженной к этому времени. Для проверки существования переменной используется конструкция `isset()`, которая имеет следующий синтаксис:

```
isset($var [, $var1 [, ...]])
```

В круглых скобках конструкции `isset()` указывается либо одна, либо несколько разделенных запятой переменных. Если все переменные существуют, конструкция возвращает `TRUE`, если хотя бы одна переменная не существует — возвращается `FALSE`.

В листинге 3.20 демонстрируется использование конструкции `isset()`.

> ***Замечание***
>
> В листинге 3.20 используется оператор сравнения `if`, который выполняет следующие за ним операторы, если ему передано значение `TRUE`, и игнорирует, если значение `FALSE`. Более подробно синтаксис оператора `if` обсуждается в *главе 5*.

Листинг 3.20. Использование конструкции `isset()`

```
<?php
  // Объявляем пустую переменную
  $str = '';

  if (isset($str)) // TRUE
    echo 'Переменная $str существует<br>';
```

```
  // Помечаем переменную $str как неинициализированную
  $str = NULL;
  if (isset($str)) // FALSE
    echo 'Переменная $str существует<br>';

  // Инициализируем переменные $a и $b
  $a = "variable";
  $b = "another variable";

  // Проверяем существование переменных
  if (isset($a))     // TRUE
    echo 'Переменная $a существует<br>';
  if (isset($a, $b)) // TRUE
    echo 'Переменные $a и $b существуют<br>';

  // Уничтожаем переменную $a
  unset ($a);

  // Проверяем существование переменных
  if (isset($a))     // FALSE
    echo 'Переменная $a существует<br>';
  if (isset($a, $b)) // FALSE
    echo 'Переменные $a и $b существуют<br>';
?>
```

Как видно из листинга 3.20, пустая строка не эквивалентна неинициализированной переменной (т. е. переменной, принимающей значение `NULL`). Для проверки, является ли строка пустой, предназначена специальная конструкция `empty()`, которая имеет следующий синтаксис:

```
empty($var)
```

В отличие от конструкции `isset()`, конструкция `empty()` принимает в качестве параметра лишь одну переменную `$var` и возвращает `TRUE`, если переменная равна пустой строке `""`, нулю `0`, символу нуля в строке `"0"`, `NULL`, `FALSE`, пустому массиву `array()`, неинициализированной переменной (случай, впрочем, аналогичный `NULL`). Во всех остальных случаях возвращается `FALSE`.

В листинге 3.21 демонстрируется использование конструкции `empty()`.

Листинг 3.21. Использование конструкции `empty()`

```
<?php
  // Объявляем пустую переменную
  $str = '';
```

```
  // Проверяем существование переменной
  if (isset($str)) // TRUE
    echo 'Переменная $str существует<br>';

  // Проверяем, не пустая ли переменная
  if (empty($str)) // TRUE
    echo 'Переменная $str пустая<br>';
?>
```

3.13. Определение типа переменной

PHP предоставляет гибкие средства, позволяющие определить, к какому типу данных принадлежит переменная. Это бывает полезно, учитывая, что ряд функций (особенно функции для работы с массивами и объектами) крайне чувствительны к типу данных. В табл. 3.4 приводится список функций, позволяющих определять принадлежность переменной к тому или иному типу.

ЗАМЕЧАНИЕ

Часть функций из табл. 3.4 описывается также в *разд. 12.8*.

Таблица 3.4. *Функции определения типа переменной*

Функция	Описание
`get_defined_vars()`	Возвращает массив с именами всех объявленных переменных
`get_resource_type($handle)`	Возвращает тип дескриптора `$handle`
`gettype($var)`	Возвращает тип переменной `$var`. Возвращаемое значение может принимать одно из значений: `"boolean"`, `"integer"`, `"double"` (и для `double` и для `float`), `"string"`, `"array"`, `"object"`, `"resource"`, `"NULL"` и `"unknown type"`. Описание типов приводится в табл. 3.1
`is_array($var)`	Возвращает `TRUE`, если `$var` является массивом, в противном случае возвращается `FALSE`
`is_bool($var)`	Возвращает `TRUE`, если `$var` является переменной логического типа, в противном случае возвращается `FALSE`
`is_callable($var)`	Возвращает `TRUE`, если `$var` является функцией, в противном случае возвращается значение `FALSE`. Дело в том, что ряд функций принимает в качестве параметра названия пользовательских функций, к которым они обращаются по мере выполнения. Функция `is_callable()` проверяет, соответствует ли переменной `$var` какая-либо из существующих функций. Более подробно функции обсуждаются в *главе 7*

Таблица 3.4 *(окончание)*

Функция	Описание
`is_double($var)`	Возвращается `TRUE`, если `$var` является переменной типа `double` (или `float`), в противном случае возвращается `FALSE`
`is_float($var)`	Синоним функции `is_double()`
`is_int($var)`	Возвращается `TRUE`, если `$var` является целочисленной переменной типа `int`, в противном случае возвращается `FALSE`
`is_integer($var)`	Синоним для функции `is_int()`
`is_long($var)`	Синоним для функции `is_long()`
`is_null($var)`	Возвращает `TRUE`, если `$var` равняется `NULL`, в противном случае возвращается `FALSE`
`is_numeric($var)`	Возвращает `TRUE`, если `$var` является переменной числового типа (`int`, `float` или `double`), в противном случае возвращается `FALSE`
`is_object($var)`	Возвращает `TRUE`, если `$var` является объектом, в противном случае возвращается `FALSE`
`is_real($var)`	Синоним для функции `is_float()`
`is_resource($var)`	Возвращает `TRUE`, если `$var` является дескриптором, в противном случае возвращается `FALSE`
`is_scalar($var)`	Возвращает `TRUE`, если `$var` является скаляром (`int`, `float`, `double`, `string` или `boolean`), в противном случае возвращается `FALSE` (`array`, `object` или `resource`)
`is_string($var)`	Возвращает `TRUE`, если `$var` является переменной строкового типа `string`, в противном случае возвращается `FALSE`

Рассмотрим более подробно функции, представленные в табл. 3.4. В первую очередь следует отменить функцию `get_defined_vars()`, которая позволяет получить массив всех объявленных в скрипте переменных. Так как массивы рассматриваются только в *главе 6*, отложим подробное обсуждение этой функции до этого времени.

Универсальной функцией, позволяющей определить тип переменной, является функция `gettype()`, которая получает в качестве аргумента переменную и возвращает ее тип (листинг 3.22).

Листинг 3.22. Использование функции `gettype()`

```
<?php
  // Объявляем целую переменную
  $number = 123;
```

```
  echo gettype($number); // integer

  // Объявляем логическую переменную
  $flag = TRUE;
  echo gettype($flag); // boolean

  // Объявляем строковую переменную
  $str = '';
  echo gettype($str); // string

  // Объявляем вещественную переменную
  $var = 1.7;
  echo gettype($var); // double

  // Уничтожаем переменную $var
  unset($var);
  echo gettype($var); // NULL
?>
```

Для каждого из типов, представленных в табл. 3.1, предназначена функция, начинающаяся с префикса `is_`, которая возвращает `TRUE`, если переменная принадлежит заданному типу. В листинге 3.23 демонстрируется использование функции `is_int()`, определяющей, принадлежит переменная целому типу `int` или нет.

Листинг 3.23. Использование функции `is_int()`

```
<?php
  // Объявляем целую переменную
  $number = 123;
  if (is_int($number)) // TRUE
    echo "Переменная $number является целочисленной<br>";

  // Объявляем строковую переменную
  $str = '123';
  if (is_int($str)) // FALSE
    echo "Переменная $str является целочисленной<br>";
?>
```

Результатом работы скрипта из листинга 3.23 будет лишь одна строка, сообщающая, что только переменная `$number` является целочисленной. Переменная `$str`, несмотря на то, что содержит число, имеет строковый тип, поэтому функция `is_int()` возвращает для нее `FALSE`.

В отличие от других языков программирования, где типы `float` и `double` имеют разный размер допустимых значений, в PHP между ними не существует различий, в чем можно легко убедиться при помощи скрипта, представленного в листинге 3.24.

Листинг 3.24. Использование функций `is_float()` и `is_double()`

```
<?php
  // Объявляем переменную типа "float"
  $float = 123.24;
  if (is_float($float)) // TRUE
    echo "Переменная $float имеет тип float<br>";
  if (is_double($float)) // TRUE
    echo "Переменная $float имеет тип double<br>";

  // Объявляем переменную типа "double"
  $double = 123.24e307;
  if (is_float($double)) // TRUE
    echo "Переменная $double имеет тип float<br>";
  if (is_double($double)) // TRUE
    echo "Переменная $double имеет тип double<br>";
?>
```

Все проверки из листинга 3.24 возвращают `TRUE`. Такой же особенностью обладает функция `gettype()`, которая возвращает значение `"double"` для всех вещественных чисел, не зависимо от того, имеют они тип `float` или `double` (листинг 3.25).

Листинг 3.25. Для вещественных чисел `gettype()` всегда возвращает `"double"`

```
<?php
  // Объявляем переменную типа "float"
  $float = 123.24;
  echo gettype($float); // double

  // Объявляем переменную типа "double"
  $double = 123.24e307;
  echo gettype($double); // double
?>
```

Использование специализированных функций, таких как `is_int()` или `is_double()`, не всегда удобно в таком слабо типизированном языке программирования, как PHP. Как будет показано в следующем разделе, числа в строках зачастую могут быть автоматически приведены к одному из числовых типов, поэтому при проверках удобнее воспользоваться обобщенной функций `is_numeric()`, которая возвращает `TRUE`, если переменная является числом или строкой, содержащей число (листинг 3.26).

Листинг 3.26. Использование функции `is_numeric()`

```
<?php
  // Объявляем вещественную переменную
  $float = 123.24;
  if (is_numeric($float)) // TRUE
    echo 'Переменная $float является числом<br>';

  // Объявляем целочисленную переменную
  $number = 0;
  if (is_numeric($number)) // TRUE
    echo 'Переменная $number является числом<br>';

  // Объявляем строковую переменную,
  // содержащую число
  $str = '12';
  if (is_numeric($str)) // TRUE
    echo 'Переменная $str является числом<br>';

  // Объявляем пустую строковую переменную
  $emptystr = '';
  if (is_numeric($emptystr)) // FALSE
    echo 'Переменная $emptystr является числом<br>';

  // Объявляем строковую переменную,
  // содержащую неправильное число
  $wrongstr = '12q';
  if (is_numeric($wrongstr)) // FALSE
    echo 'Переменная $wrongstr является числом<br>';

  // Объявляем логическую переменную
  $flag = TRUE;
  if (is_numeric($flag)) // FALSE
    echo 'Переменная $flag является числом<br>';
?>
```

Как видно из листинга 3.26, функция `is_numeric()` возвращает `TRUE` только для чисел и строк, содержащих числа, во всех остальных случаях возвращается `FALSE`.

Функция `is_resource()` позволяет определить, является ли переменная дескриптором. Эта функция часто используется совместно с функцией `get_resource_type()`, возвращающей тип дескриптора. В листинге 3.27 приводится пример установки соединения с СУБД MySQL при помощи функции `mysql_connect()`. Забегая вперед, сообщим, что функция `mysql_connect()` возвращает дескриптор соединения в случае успешной установки соединения и `FALSE` в противном случае.

Замечание

Подробнее СУБД MySQL и порядок работы с ней обсуждаются в *главах 27* и *28*.

Листинг 3.27. Использование функций `is_resource()` и `get_resource_type()`

```
<?php
  // Устанавливаем соединение с СУБД MySQL
  $dbcnx = mysql_connect("localhost", "root", "");

  // Проверяем, является ли переменная $dbcnx дескриптором соединения
  if (is_resource($dbcnx))
  {
    // Да, переменная $dbcnx является дескриптором.
    // Выводим тип дескриптора в окно браузера
    echo get_resource_type($dbcnx); // mysql link
  }
?>
```

Если функция `mysql_connect()` успешно устанавливает соединение с СУБД MySQL, то переменная `$dbcnx` становится дескриптором (в противном случае переменная получит значение `FALSE` и будет отнесена к логическому типу). Если теперь передать этот дескриптор функции `get_resource_type()`, то она возвратит его тип в виде строки `"mysql link"`.

3.14. Неявное приведение типов

Все языки программирования можно условно поделить на сильно типизированные и слабо типизированные. В сильно типизированных языках программирования при объявлении переменной, как правило, следует указывать ее тип, а присвоение переменной одного типа переменной другого типа приводит к ошибке. В слабо типизированных языках программирования (к ним относится и PHP) не требуется явное указание типа переменной, а попытка использования переменной в контексте, где ожидается переменная другого типа, приведет к тому, что PHP попытается автоматически (неявно) преобразовать переменную к нужному типу. Например, если строка содержит число и используется в арифметическом выражении, то она автоматически будет приведена к числовому типу (листинг 3.28).

Листинг 3.28. Автоматическое преобразование строки в число

```
<?php
  $str = "12.6";
  $number = 3 + $str;
  echo $number; // 15.6
?>
```

Важно отметить, что строка может содержать помимо числа любые другие символы, интерпретатор PHP постарается извлечь из начала строки наиболее полное значение, соответствующее числу. Если извлечь число из строки не удается, ее значение рассматривается как нулевое (листинг 3.29).

Листинг 3.29. Преобразование сложных строк в число

```
<?php
  echo "12wet56.7" + 10;      // 12 + 10 = 22
  echo "<br>";
  echo 14 + "четырнадцать";  // 14 + 0 = 14
?>
```

Аналогичным образом число автоматически преобразуется в строку там, где ожидается строковая переменная (например, при выводе в окно браузера).

Если ожидается логический тип (например, в условных операторах), числа, равные нулю, пустая строка, строка, содержащая `"0"`, пустые массивы и объекты, а также `NULL` автоматически приводятся к значению `FALSE`, все остальные переменные рассматриваются как `TRUE` (листинг 3.30).

Листинг 3.30. Преобразование к логическому типу

```
<?php
  // Объявляем нулевое вещественное число
  $float = 0.0;
  if ($float) // FALSE
    echo 'Переменная $float рассматривается как TRUE';

  $str = "Hello world!";
  if ($str) // TRUE
    echo 'Переменная $str рассматривается как TRUE';
?>
```

Дескрипторы всегда рассматриваются как `TRUE`, поэтому для проверки корректности выделения дескриптора вместо функции `is_resource()`, рассмотренной в *разд. 3.13*, очень часто в операторе `if` используют сам дескриптор (листинг 3.31).

Листинг 3.31. Преобразование дескриптора к логическому типу

```
<?php
  // Получаем дескриптор соединения с СУБД MySQL
  $dbcnx = mysql_connect("localhost", "root", "");

  // Проверяем корректность выделения дескриптора
```

```
  if ($dbcnx)
  {
    // Дескриптор успешно получен, осуществляем дальнейшие действия
    ...
  }
?>
```

При преобразовании логического типа к строке, `TRUE` превращается в `"1"`, а `FALSE` в пустую строку `""` (листинг 3.32). Преобразование логического типа к числу приводит к превращению `TRUE` в 1, а `FALSE` в 0.

Листинг 3.32. Преобразование логического типа к строке

```
<?php
  echo TRUE; // "1"
  echo FALSE // ""
?>
```

3.15. Явное приведение типов

Помимо неявного преобразования типов, разработчик может явно потребовать от интерпретатора PHP преобразовать ту или иную переменную к одному из поддерживаемых типов. Для такого преобразования предусмотрено несколько механизмов. Первый из них заключается в использовании круглых скобок, в которых указывается тип, к которому следует привести переменную (листинг 3.33).

> ***ЗАМЕЧАНИЕ***
>
> Следует отметить, что при приведении вещественных чисел к целому, не происходит округления, дробная часть просто отбрасывается. Для округления числа следует воспользоваться специальной функцией `round()`, синтаксис которой рассматривается в *главе 12*.

Листинг 3.33. Преобразование типа `float` к `int`

```
<?php
  // Объявляем вещественную переменную
  $float = 4.863;

  // Преобразуем переменную $float к целому типу
  $number = (int)$float;

  echo $number; // 4
?>
```

В табл. 3.5 приводятся все возможные варианты использования оператора круглых скобок.

Таблица 3.5. *Явное приведение типа при помощи оператора круглых скобок*

Преобразование	Описание
`$var = (int) $var;`	Приведение к целому типу `int`
`$var = (integer) $var;`	Приведение к целому типу `int`
`$var = (bool) $var;`	Приведение к логическому типу `boolean`
`$var = (boolean) $var;`	Приведение к логическому типу `boolean`
`$var = (float) $var;`	Приведение к вещественному типу `double`
`$var = (double) $var;`	Приведение к вещественному типу `double`
`$var = (real) $var;`	Приведение к вещественному типу `double`
`$var = (string) $var;`	Приведение к строковому типу `string`
`$var = (binary) $var;`	Приведение к бинарной строке
`$var = (array) $var;`	Приведение к массиву
`$var = (object) $var;`	Приведение к объекту

Обычно явного преобразования типов не требуется, однако это может быть полезным. В листинге 3.34 приводится пример определения четности числа: проверяемое число два раза делится на 2, при этом один раз приводится к целому, а второй к вещественному значению. Полученные результаты вычитаются друг из друга, если число четное, результат будет равен нулю, который автоматически рассматривается в контексте оператора `if` как `FALSE`, если число нечетное, то результат будет равен `0.5`, что рассматривается как `TRUE`.

Листинг 3.34. Определение четности числа

```
<?php
  // Объявляем число
  $number = 15;

  $flag = (float)($number/2) - (int)($number/2);

  // Определяем статус числа
  if ($flag) // TRUE, т. к. $flag == 0.5
    echo "Число $number нечетное";
  else
    echo "Число $number четное";
?>
```

Преобразование к бинарной строке (`binary`) введено в PHP, начиная с версии 5.2.1, и может быть заменено префиксом `b` перед кавычками: `b"строка"`. В листинге 3.35 оба преобразования эквивалентны.

Листинг 3.35. Приведение к бинарной строке

```
<?php
  $str = (binary) "1234.5";
  $str = b"1234.5";
?>
```

Помимо оператора круглых скобок, PHP предоставляет ряд специальных функций, позволяющих осуществить преобразование типа переменной (табл. 3.6).

Таблица 3.6. *Функции преобразования типа переменных*

Функция	Описание
`settype($var, $type)`	Преобразует переменную `$var` к типу, указанному в параметре `$type`, который может принимать одно из следующих значений: `"boolean"`, `"bool"`, `"integer"`, `"int"`, `"float"`, `"string"`, `"array"`, `"object"` и `"null"`. Функция возвращает `TRUE`, если преобразование было успешно осуществлено, и `FALSE` в противном случае
`floatval($var)`	Преобразует переменную `$var` к вещественному типу `float`
`doubleval($var)`	Синоним для функции `floatval()`
`intval($var [, $base])`	Преобразует переменную `$var` к вещественному типу `int` по основанию `$base`
`strval($var)`	Преобразует переменную `$var` к строковому типу `string`

Функция `settype()` является универсальной функцией преобразования типов и принимает в качестве первого параметра переменную, которую необходимо подвергнуть преобразованию, а в качестве второго параметра — название типа, который должна иметь переменная после преобразования. В листинге 3.36 демонстрируется использование функции `settype()`.

Листинг 3.36. Использование функции `settype()`

```
<?php
  // Объявляем строковую переменную
  $str = "5wet";
  // Приводим строку к целому числу
  settype($str, "integer");
  echo $str; // 5
```

```
  // Перевод строки в браузере
  echo "<br>";

  // Объявляем логическую переменную
  $flag = TRUE;
  // Приводим логическую переменную к строке
  settype($flag, "string");
  echo $flag; // 1
?>
```

Однако использование функции `settype()` не очень удобно из-за необходимости явного указания типа и того факта, что функция возвращает логическое значение, а не результат преобразования. В реальной практике, если в явном преобразовании возникает необходимость, гораздо удобнее пользоваться специализированными функциями `floatval()`, `doubleval()`, `intval()` и `strval()`. В листинге 3.37 на примере функции `intval()` демонстрируется использование функций данного класса.

> ***Замечание***
>
> Если функции `intval()` передается число, которое превышает размеры типа `int`, возвращается максимально возможное для данного типа значение — 2 147 483 647. Если в силу каких-либо обстоятельств переменная не может быть преобразована в целый тип, функция возвращает 0.

Листинг 3.37. Использование функции `intval()`

```
<?php
  echo intval(42);                      // 42
  echo intval(4.2);                     // 4
  echo intval('42');                    // 42
  echo intval('+42');                   // 42
  echo intval('-42');                   // -42
  echo intval(042);                     // 34
  echo intval('042');                   // 42
  echo intval(1e10);                    // 2147483647
  echo intval('1e10');                  // 1
  echo intval(0x1A);                    // 26
  echo intval(42000000);                // 42000000
  echo intval(420000000000000000000);   // 2147483647
  echo intval('420000000000000000000'); // 2147483647
  echo intval(42, 8);                   // 42
  echo intval('42', 8);                 // 34
?>
```

Функция `intval()` имеет второй необязательный параметр, который позволяет задать основание числа в первом параметре функции. Так если этот необязательный

параметр принимает значение 8, то ожидается, что в первом параметре передается восьмеричное число (попытки использовать числа 8 и 9 приведут к тому, что функция возвратит 0), если в качестве второго параметра передается значение 16, то в первом параметре будет ожидаться шестнадцатеричное число. Функция `intval()` всегда возвращает результат в десятичной системе счисления.

3.16. Динамические переменные

В заключение главы рассмотрим еще один способ создания переменных, он состоит в использовании двойного символа `$$`, позволяющего создавать переменные произвольного типа. Иногда название переменной невозможно определить заранее, и оно должно определяться по мере выполнения скрипта — в этом случае прибегают к *динамическим переменным* (листинг 3.38).

> ***Замечание***
>
> Не следует злоупотреблять динамическим формированием имени переменной, т. к. это приводит к снижению читабельности кода и значительно усложняет его отладку и сопровождение.

Листинг 3.38. Создание динамической переменной

```
<?php
  $id_menu = 3;
  $str = "active$id_menu"; // "active3"
  $$str = 1; // $active3 = 1;
  if (isset($active3))
    echo "Переменная \${$str} существует и равна $active3";
?>
```

Результатом работы скрипта из листинга 3.38 будет следующая строка:

```
Переменная $active3 существует и равна 1
```

Значение переменной, следующей после первого знака `$`, воспринимается как имя для новой переменной. В результате, если строка `$str` имеет значение `"active3"`, то имя новой переменной будет `$active3`.

Еще один способ создания динамических переменных заключается в использовании функции `eval()`. Эта функция принимает в качестве параметра строку с PHP-кодом и выполняет ее (листинг 3.39).

Листинг 3.39. Использование функции `eval()`

```
<?php
  $code = '$str = "Hello, world!<br>";
                  echo $str;';
```

```
  eval($code);
  echo $str;
?>
```

В результате выполнения скрипта из листинга 3.39 в окно браузера будут выведены следующие строки:

```
Hello, world!
Hello, world!
```

Таким образом, переменная `$str`, которая определена в строке `$code`, доступна не только в пределах данной строки, но и после выполнения функции `eval()`. Именно на этом эффекте основано формирование динамических переменных, например восьмеричных чисел, определяющих права доступа (листинг 3.40).

Листинг 3.40. Динамическое формирование значения переменных

```
<?php
  $number = 755;
  eval("\$mode = 0$number;");
  if (isset($mode)) echo $mode; // 493
?>
```

Код в листинге 3.40 эквивалентен определению переменной `$mode` в скрипите следующим образом: `$mode = 0755`. При помощи функции `eval()` можно динамически формировать не только значения переменных, но и сами переменные. Код в листинге 3.41 по результату полностью эквивалентен скрипту из листинга 3.38.

Листинг 3.41. Динамическое формирование имени переменной

```
<?php
  $id_menu = 3;
  eval("\$active$id_menu = 1;");
  if (isset($active3))
    echo "Переменная \$active3 существует и равна $active3";
?>
```



Глава 4

Константы

Константы отличаются от переменных тем, что их значение не может изменяться по мере выполнения скрипта. Кроме того, константы, в отличие от переменных, не могут принимать неопределенное значение `NULL`.

4.1. Объявление константы. Функция *define()*

Объявление константы осуществляется при помощи функции `define()`, которая имеет следующий синтаксис:

```
define($name, $value [, $case_insensitive])
```

Функция принимает в качестве первого параметра *`$name`* строку с именем константы, в качестве второго параметра *`$value`* передается значение константы. Третий необязательный параметр *`$case_insensitive`* определяет чувствительность константы к регистру. По умолчанию, константы точно так же, как и переменные, зависят от регистра, т. е. константы с именами `CONSTANT` и `constant` считаются разными, однако, если при создании константы в качестве третьего параметра *`$case_insensitive`* передать значение `TRUE`, новая константа не будет зависеть от регистра. В случае успешного создания константы функция `define()` возвращает `TRUE`, в противном случае возвращается значение `FALSE`.

В листинге 4.1 приводится пример создания двух констант: `NUMBER`, принимающей значение 1, и `VALUE`, принимающей строковое значение "Hello, world!". Следует помнить, что в качестве значения константы может выступать лишь скалярная величина, т. е. либо логическое (`boolean`), либо целочисленное (`integer`), либо вещественное (`float`), либо строковое (`string`) значение. Массивы (`array`), объекты (`object`) и дескрипторы (`resource`) не могут выступать в качестве значения константы.

> ***Замечание***
>
> Традиционно, имена констант записываются литерами верхнего регистра, хотя допускается создание констант с именами, содержащими литеры произвольного регистра. Эта традиция связана с тем, что в большинстве языков высокого уровня константы и переменные никак не отличаются друг от друга (переменные не снабжаются начальным символом `$`, в отличие от PHP), поэтому правила хорошего тона предписывают снабжать переменные и константы именами в разных регистрах, чтобы улучшить читабельность программы.

Листинг 4.1. Создание констант. Использование функции `define()`

```
<?php
  define("NUMBER", 1);
  define("VALUE", "Hello, world!");
  echo NUMBER; // 1
  echo VALUE;  // Hello, world!
  echo Number; // Notice: Use of undefined
               // constant Number - assumed 'Number'
?>
```

Как видно из листинга 4.1, обращаться к константам можно по их имени, при этом, в отличие от констант, не требуется указывать символ `$` перед именем константы. Если осуществляется попытка обратиться к несуществующей константе, при включенном режиме отображения сообщений (Notice) генерируется замечание "Notice: Use of undefined constant". В листинге 4.1 осуществляется попытка обратиться к неопределенной константе `Number`, что вызывает генерацию соответствующего замечания. Помимо замечания в окно браузера выводится строка "Number", в которую преобразуется неопределенная константа (рис. 4.1).

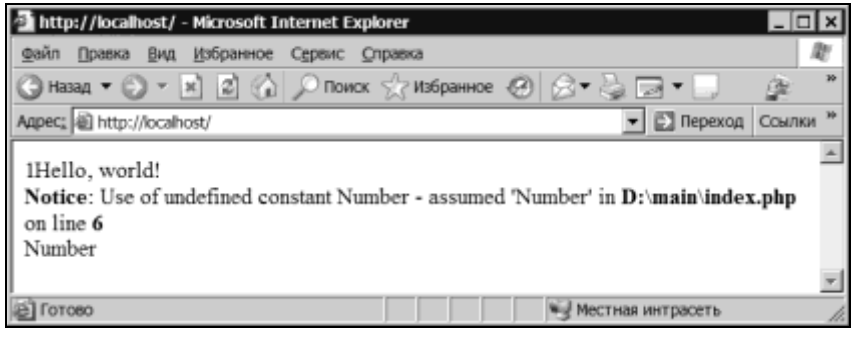


Рис. 4.1. Попытка обращения к неопределенной константе `Number`

Если необходима константа, не зависящая от регистра, то при ее создании, функции `define()` в качестве третьего параметра необходимо передать значение `TRUE` (листинг 4.2).

Замечание

Если в качестве третьего параметра функции `define()` передать значение `FALSE`, то константы будут зависеть от регистра, однако обычно третий параметр просто не указывают, и он получает значение `FALSE` по умолчанию (листинг 4.1).

Листинг 4.2. Создание констант, имя которых не зависит от регистра

```
<?php
  define("VALUE", "Hello, world!", TRUE);
  echo VALUE;
  echo Value;
  echo VaLuE;
?>
```

Следует отметить, что PHP имеет ряд предопределенных констант, которые затрагивались в предыдущей главе, таких как `NULL`, `TRUE` и `FALSE`. Все они не зависят от регистра, поэтому в программах можно встретить различные их записи, например, `TRUE`, `true`, `True` и т. п.

Интересно отметить поведение функции `define()` при попытке переопределения уже существующей константы. В этом случае функция возвращает значение `FALSE`, поэтому всегда можно проверить успешность объявления константы при помощи оператора `if` (листинг 4.3).

> ***Замечание***
>
> Условный оператор `if` более подробно рассматривается в *главе 5*.

Листинг 4.3. Попытка переопределения константы

```
<?php
  if (define("VALUE", "Hello, world!"))
    echo "Константа VALUE получила значение \"Hello, world!\"<br>";
  if (define("VALUE", 1))
    echo "Константа VALUE получила значение 1<br>";
  echo VALUE;
?>
```

Если в конфигурационном файле php.ini директивой `error_reporting` разрешается вывод замечаний, то при попытке повторного определения константы генерируется замечание "Notice: Constant value already defined" (рис. 4.2).

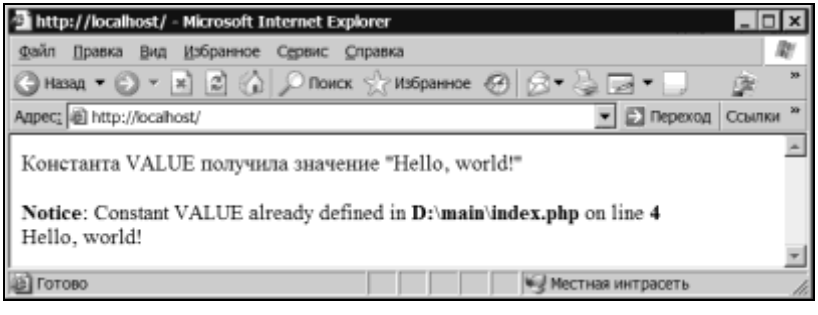


Рис. 4.2. Попытка переопределения константы

Как видно из результатов, при первом вызове функции `define()`, создающем константу `VALUE`, функция возвращает `TRUE`. Повторный вызов функции для создания уже существующей константы возвращает `FALSE`, при этом у константы остается старое значение.

Сложности возникают, когда в дело вступает третий параметр функции `define()`. Если оба вызова функции в качестве третьего параметра используют либо `TRUE`, либо `FALSE`, поведение функции остается таким же, как было описано ранее. Если же один из вызовов функции `difine()` в качестве третьего параметра имеет значение `FALSE`, а второй `TRUE`, то создаются две константы с двумя разными значениями (листинг 4.4).

> ***ЗАМЕЧАНИЕ***
>
> Создание двух констант с одинаковыми именами, одно из которых зависит от регистра, а другое нет, больше похоже на ошибку, чем на особенность (фичу) языка программирования, поэтому не рекомендуется основывать код на эксплуатации этой особенности. Возможно, в следующих версиях PHP эта ситуация будет исправлена.

Листинг 4.4. Две разные константы с одни и тем же именем

```
<?php
  if (define("VALUE", "Hello world!"))
    echo "Константа VALUE получила значение \"Hello, world!\"<br>";
  if (define("VALUE", 1, TRUE))
    echo "Константа VALUE получила значение 1<br>";
  echo VALUE;
  echo Value;
?>
```

Результат работы скрипта из листинга 4.4 представлен на рис. 4.3. Константа `VALUE`, зависящая от регистра, как бы перекрывает своим значением константу `Value` (`VaLuE`, `value` и т. п.), не зависящую от регистра.

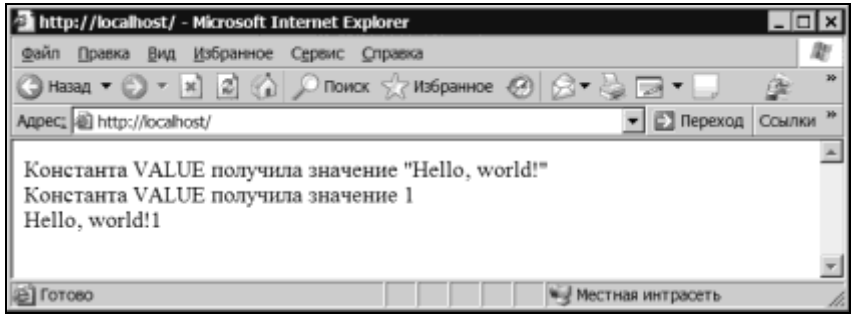


Рис. 4.3. Две разные константы с одним и тем же именем

4.2. Функции для работы с константами

Помимо функции `define()`, рассмотренной в предыдущем разделе, существуют дополнительные функции для работы с константами. Их полный список представлен в табл. 4.1.

Таблица 4.1. *Функции для работы с константами*

Функция	Описание
`define($name, $value [, $case_insensitive] )`	Создает константу с именем `$name` и значением `$value`. Если необязательный параметр `$case_insensitive` принимает значение `TRUE`, имя константы не зависит от регистра. Функция возвращает `TRUE` при успешном создании константы и `FALSE` в противном случае
`defined($name)`	Проверяет, существует ли константа с именем `$name`, и возвращает `TRUE`, если константа существует, и `FALSE` в противном случае
`constant($name)`	Возвращает значение константы с именем `$name`
`get_defined_constants ([$categorize])`	Возвращает список пользовательских и предопределенных констант в виде массива. Если необязательный параметр `$categorize` не указан или принимает значение `FALSE`, функция возвращает одномерный массив, если параметр принимает значение `TRUE`, возвращается двумерный массив, константы в котором разбиты по категориям (пользовательские константы, константы расширений, ядра PHP и т. п.)

4.3. Константы с динамическими именами. Функция *constant()*

В списке функций, представленных в табл. 4.1, особый интерес вызывает функция `constant()`, которая возвращает значение константы. В листинге 4.5 обе строки, выводящие значение константы `VALUE` при помощи прямого обращения и функции `constant()`, эквивалентны.

> ***Замечание***
>
> Если константа не существует, то функция `constant()` генерирует не замечание (Notice), а предупреждение (Warning).

Листинг 4.5. Использование функции `constant()`

```
<?php
  // Определяем константу VALUE
  define("VALUE", 1);
```

```
  // Прямое обращение к константе
  echo VALUE; // 1
  // Получение значение константы через функцию constant()
  echo constant("VALUE"); // 1
?>
```

Обычно используют прямое обращение к константе, однако функция `constant()` может быть полезна особенно в том случае, если имя константы формируется динамически и не может быть жестко вбито в тело скрипта. В листинге 4.6 при помощи функции `rand()` *(см. главу 12)* генерируется случайное число `$index` от 1 до 10, которое в дальнейшем используется для формирования константы. В результате имя константы всякий раз может принимать одно из десяти значений. Получить значение такой константы можно только при помощи функции `constant()`.

Листинг 4.6. Константа с динамическим именем

```
<?php
  // Формируем случайное число от 1 до 10
  $index = rand(1, 10);
  // Формируем имя константы
  $name = "VALUE{$index}";

  // Определяем константу с динамическим именем
  define($name, 1);

  // Получаем значение константы
  echo constant($name);
?>
```

4.4. Проверка существования константы

Для проверки существования константы предназначена функция `defined()`, которая в качестве единственного параметра принимает строку с именем константы и возвращает `TRUE`, если константа существует, и `FALSE` в противном случае. В листинге 4.7 приводится наиболее типичное использование этой функции.

Листинг 4.7. Использование функции `defined()`

```
<?php
  // Определяем константу
  define("VALUE", 1);

  // Если константа существует, выводим ее значение
  if (defined("VALUE")) echo VALUE; // 1
?>
```

4.5. Предопределенные константы

Помимо констант, определяемых разработчиком, существует ряд предопределенных констант, значение которым выставляет интерпретатор PHP. Список предопределенных констант представлен в табл. 4.2.

Замечание

Классы и соответствующие предопределенные константы более подробно рассматриваются в *главах 20—26*.

Таблица 4.2. *Предопределенные константы*

Константа	Описание
`__LINE__`	Текущая строка в файле
`__FILE__`	Полный путь и имя текущего файла
`__FUNCTION__`	Имя функции
`__CLASS__`	Имя класса
`__METHOD__`	Имя метода класса

Помимо представленных в табл. 4.2 констант, которые доступны всегда, существует огромное количество предопределенных констант, объявленных в различных расширениях. Привести их полный список не представляется возможным, кроме того, их набор зависит от подключенных в данный момент расширений и может меняться. В любом случае получить полный список доступных в текущий момент констант можно при помощи функции `get_defined_constants()` (см. табл. 4.1).

В листинге 4.8 приводится пример использования предопределенных констант. Константы `__LINE__` и `__FILE__` удобно использовать для вывода сообщений об ошибках.

Листинг 4.8. Использование предопределенных констант

```
<?php
  echo "Имя файла ".__FILE__."<br>";
  echo "Строка ".__LINE__;
?>
```

Результат работы скрипта из листинга 4.8 может выглядеть так, как это представлено на рис. 4.4.



Рис. 4.4. Использование предопределенных констант



Глава 5

Операторы и конструкции языка

Если переменные и константы можно рассматривать как своеобразные "существительные" языка программирования, то операторы и конструкции относятся к "глаголам". С некоторыми операторами и конструкциями, такими как оператор инициализации переменной =, оператор создания строки <<<, конструкции для работы с переменными `isset()`, `unset()` и `empty()`, мы познакомились в предыдущих главах. В данной главе будут рассмотрены остальные операторы и конструкции языка PHP.

> ***Замечание***
>
> В данной главе не рассматривается оператор цикла `foreach()`, обсуждение которого отложено до *главы 6*, посвященной массивам.

5.1. Объединение строк. Оператор "точка"

Оператор "точка" позволяет объединять строки. В листинге 5.1 приводится пример, в котором при помощи точки объединяются две строки и число в одну целую строку. Число при этом автоматически приводится к строке.

> ***Замечание***
>
> Во многих языках программирования для объединения строк используется оператор +. В PHP это не так, при использовании строк в выражении с оператором + строка автоматически приводится к числовому значению. Если строка не содержит ничего похожего на число, она приводится к нулевому значению.

Листинг 5.1. Использование оператора "точка"

```
<?php
  $number = 216;
  echo "Сегодня " . $number . " участников"; // Сегодня 216 участников
?>
```

Скрипт из листинга 5.1 можно записать в альтернативной форме, путем интерполяции переменной, а не объединения строк (листинг 5.2).

Листинг 5.2. Альтернативная запись

```
<?php
  $number = 216;
   echo "Сегодня $number участников";
   echo "Сегодня {$number} участников";
?>
```

Помимо оператора точки (.) существует оператор .=, который предназначен для сокращения конструкции `$str = $str . $newstring` до `$str .= $newstring`. Скрипт из листинга 5.1 с учетом оператора .= можно переписать так, как это представлено в листинге 5.3.

Листинг 5.3. Использование оператора .=

```
<?php
  $number = 216;
  $str = "Сегодня ";
  $str .= $number;
  $str .= " участников";
  echo $str;
?>
```

5.2. Конструкция *echo*. Оператор "запятая"

Конструкция `echo` предназначена для вывода переменных в окно браузера и уже многократно использовалась в предыдущих главах. Конструкция имеет следующий синтаксис:

```
echo($str1 [, str2..])
```

Может принимать один или более параметров, разделенных запятой, которые она выводит в окно браузера. В листинге 5.1 приводился пример, в котором три строки объединялись в одну перед выводом их конструкцией `echo`. Однако эту операцию можно осуществить средствами конструкции, просто перечислив все параметры через запятую после ключевого слова `echo` (листинг 5.4).

Листинг 5.4. Вывод нескольких строк при помощи конструкции `echo`

```
<?php
  $number = 216;
  echo "Сегодня " , $number, " участников"; // Сегодня 216 участников
?>
```

Параметры конструкции `echo` могут помещаться в необязательные круглые скобки (листинг 5.5).

Листинг 5.5. Использование круглых скобок в конструкции `echo`

```
<?php
  $number = 216;
  echo ("Сегодня " , $number, " участников"); // Сегодня 216 участников
?>
```

Часто вместо конструкции `echo` используют строковую функцию `print()`, которая также позволяет выводить переменные в окно браузера. Однако функция `print()` принимает только один параметр, поэтому в ней нельзя приводить параметры через запятую. Кроме того, функция возвращает `TRUE`, в случае успешного вывода переменной и `FALSE` в противном случае. Конструкция `echo` не возвращает ничего (листинг 5.6).

ЗАМЕЧАНИЕ

Строковые функции подробно рассматриваются в *главе 9*.

Листинг 5.6. Использование строковой функции `print()`

```
<?php
  $number = 216;
  print("Сегодня " . $number . " участников"); // Сегодня 216 участников
  echo print("Hello, world!");    // Hello, world!
  // print echo "Hello, world!";  // Ошибка
?>
```

5.3. Арифметические операторы

Выполнение операций над числами производится при помощи арифметических операторов, которые перечислены в табл. 5.1.

Таблица 5.1. *Арифметические операторы*

Оператор	Операция
+	Сложение
*	Умножение
-	Вычитание
/	Деление
%	Деление по модулю
++	Увеличение на единицу (инкремент)
--	Уменьшение на единицу (декремент)

Операторы сложения, вычитания, умножения и деления используются по правилам, принятым в арифметике: сначала выполняются операторы умножения и деления и лишь затем операторы сложения и вычитания. Для того чтобы изменить такой порядок, следует прибегать к группированию чисел при помощи круглых скобок (листинг 5.7).

> ***Замечание***
>
> Традиционно, до и после арифметического оператора помещаются пробелы, т. к. это позволяет увеличить читабельность программы. Однако это необязательное требование, так выражение в листинге 5.7 может быть записано следующим образом: `(5+3)*(6+7)`.

Листинг 5.7. Использование арифметических операторов

```
<?php
  echo (5 + 3) * (6 + 7); // 104
?>
```

В отличие от многих других C-подобных языков программирования, если при операции деления целого числа на целое получается дробное число, то результат автоматически приводится к вещественному типу (листинг 5.8).

> ***Замечание***
>
> По умолчанию для дробного числа выводится только 12 значащих цифр (не считая точки), однако это значение можно изменить, исправив значение директивы `precision` в конфигурационном файле php.ini. Так, если изменить значение до 4, то вместо дроби 1.66666666667 будет выведено число 1.667. Изменив значение директивы до 20, можно наблюдать ошибки вычисления, накапливающиеся в конце дроби — 1.6666666666666667407.

Листинг 5.8. Использование оператора деления /

```
<?php
  $number = 5 / 3;
  echo $number; // 1.66666666667
?>
```

Для того чтобы получить целое значение, потребуется явно привести результат деления к целому типу либо при помощи конструкции `(int)`, либо при помощи функции `intval()` (листинг 5.9).

Листинг 5.9. Получение целочисленного результата деления

```
<?php
  $number = (int)(5 / 3);
  echo $number; // 1
?>
```

Как видно из листинга 5.9, дробная часть при приведении к целому числу отбрасывается.

Для того чтобы выяснить, делится ли число без остатка на другое число, можно воспользоваться оператором деления по модулю `%` (листинг 5.10).

Листинг 5.10. Получение остатка от деления

```
<?php
  echo 5 % 3; // 2
  echo 6 % 3; // 0
?>
```

При помощи оператора `%` удобно проверять четность числа: если при делении на 2 имеется остаток — число нечетное, если оператор `%` вернет 0 — число четное (листинг 5.11).

Листинг 5.11. Проверка числа на четность

```
<?php
  $number = 5317;
  if ($number % 2) echo "Число четное";
  else echo "Число нечетное";
?>
```

Помимо операторов, приведенных в табл. 5.1, можно также использовать "сокращенную запись" арифметических операторов (табл. 5.2). То есть выражение `$a = $a + $b` в сокращенной форме запишется как `$a += $b`, что означает "присвоить операнду левой части сумму значений левого и правого операндов". Аналогичные действия допустимы для всех приведенных выше операторов.

Таблица 5.2. *Сокращенные операторы*

Сокращенная запись	Полная запись
`$number += $var;`	`$number = $number + $var;`
`$number *= $var;`	`$number = $number * $var;`
`$number -= $var;`	`$number = $number - $var;`
`$number /= $var;`	`$number = $number / $var;`
`$number %= $var;`	`$number = $number % $var;`
`$number .= $var;`	`$number = $number . $var;`

Замечание

В зависимости от количества участвующих в операции операндов, операции подразделяют на унарные и бинарные. Унарная операция работает с одним операндом, бинарная — с двумя. Все арифметические операции, кроме операций инкремента и декремента, являются бинарными. Существует также условная операция, в которой используются три операнда *(см. разд. 5.8)*.

Операторы инкремента (++) и декремента (--) подразделяют на префиксные и постфиксные. При префиксной операции, например инкремента, увеличение значения операнда на 1 происходит до того, как возвращается значение. Соответственно, при постфиксной — после. Таким образом, операторы инкремента и декремента могут применяться в двух формах — префиксной и постфиксной (листинг 5.12). Для удобства далее в примерах будет говориться только об операторе инкремента — все, что справедливо для него, справедливо также и для оператора декремента.

Листинг 5.12. Префиксная и постфиксная записи операторов инкремента

```
<?php
  $var = 1;
  // префиксная форма:
  ++$var;
  // постфиксная форма:
  $var++;
?>
```

В приведенном примере различие между префиксной и постфиксной формами записи несущественно — оба варианта эквивалентны. Различие обнаруживается, если полученное значение используется в вычислениях так, как показано в листинге 5.13.

Листинг 5.13. Использование оператора инкремента

```
<?php
  // случай 1 — префиксная форма:
  $var = 1;
  echo ++$var;  // 2
  // случай 2 — постфиксная форма:
  $var = 1;
  echo $var++;  // 1
?>
```

В первом случае (префиксная запись) на выходе получается значение 2, поскольку префиксный оператор инкремента сначала выполняет инкрементирование, а лишь затем возвращает полученное значение. Во втором случае в качестве результата получается единица. Следует учитывать это различие, поскольку из-за невнимательности при работе с операциями инкремента и декремента проистекает достаточно много ошибок, найти которые в объемном коде не всегда тривиально.

Операции инкремента и декремента могут применяться только к переменным, т. е. запись `++1` является неверной, как и запись `++($a + $b)`.

Операторы `++` и `--` применимы не только к целочисленным переменным. В листинге 5.14 приведен пример кода, в котором операция инкремента применяется к переменной типа `string`.

Листинг 5.14. Применение операции инкремента к строке

```
<?php
  $var = "aaa";
  echo ++$var;
?>
```

Результатом выполнения этого кода будет строка `aab`, т. к. буква `b` является следующей за `a` буквой алфавита.

5.4. Поразрядные операторы

Группа поразрядных операторов предназначена для выполнения поразрядных операций, так же как и арифметические операторы, обрабатывает числовые значения (табл. 5.3). Поразрядные операторы работают с двоичными данными.

Таблица 5.3. *Поразрядные операторы*

Оператор	Операция
`&`	Поразрядное пересечение — И (AND)
`|`	Поразрядное объединение — ИЛИ (OR)
`^`	Поразрядное исключающее ИЛИ (XOR)
`~`	Поразрядное отрицание (NOT)
`<<`	Сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда
`>>`	Сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда

Замечание

Использование побитовых операторов редко требуется при разработке Web-сайтов, однако они лежат в основе любых вычислений и интенсивно используются в криптографии и компьютерной графике.

Рассмотрим поразрядные операторы более подробно. Оператор `&` соответствует побитовому И (пересечение). При использовании битовых операторов следует помнить, что все операции выполняются над числами в двоичном представлении. Пусть имеются два числа — 6 и 10, применение к ним оператора `&` дает 2 (листинг 5.15).

Листинг 5.15. Использование оператора &

```
<?php
  echo 6 & 10; // 2
?>
```

Этот, казалось бы, парадоксальный результат можно легко объяснить, если перевести цифры 6 и 10 в двоичное представление, в котором они равны 0110 и 1010 соответственно. Сложение разрядов при использовании оператора `&` происходит согласно правилам, представленным в табл. 5.4; учитывая их, можно легко получить результат, который равен 0010 (в десятичной системе — 2).

```
0110 (6)
1010 (10)
0010 (2)
```

Таблица 5.4. *Правила сложения разрядов при использовании оператора* `&`

Операнд/результат	Значение			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	1	0	0	0

В листинге 5.16 приведен более сложный пример — применение оператора `&` для чисел 113 и 45.

Листинг 5.16. Использование оператора `&`

```
<?php
  echo 113 & 45; // 33
?>
```

Для наглядности запишем складываемые числа в двоичном представлении:

```
1110001 (113)
0101101 (45)
0100001 (33)
```

Оператор `|` соответствует побитовому ИЛИ (пересечение). В табл. 5.5 приведены правила сложения разрядов при использовании оператора `|`.

Таблица 5.5. *Правила сложения разрядов при использовании оператора* `|`

Операнд/результат	Значение			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	1	1	1	0

Применяя операцию `|` к числам, рассмотренным в предыдущем разделе, получим следующий результат (листинг 5.17).

Листинг 5.17. Использование оператора |

```
<?php
  echo 6 | 10;    // 14
  echo "<br>";    // Перевод строки
  echo 113 | 45;  // 125
?>
```

Распишем числа в двоичном представлении, чтобы действие оператора было более понятно:

```
0110 (6)
1010 (10)
1110 (14)
```

Для второй пары чисел:

```
1110001 (113)
0101101 (45)
1111101 (125)
```

Оператор ^ соответствует побитовому исключающему ИЛИ (XOR). В табл. 5.6 приведены правила сложения разрядов при использовании оператора ^.

Таблица 5.6. *Правила сложения разрядов при использовании оператора* ^

Операнд/результат	Значение			
Первый операнд	1	1	0	0
Второй операнд	1	0	1	0
Результат	0	1	1	0

В листинге 5.18 приведен пример использования данного оператора применительно к числам 6, 10 и 113, 45.

Листинг 5.18. Использование оператора ^

```
<?php
  echo 6 ^ 10;    // 12
  echo "<br>";    // Перевод строки
  echo 113 ^ 45;  // 92
?>
```

Распишем числа в двоичном представлении, чтобы действие оператора было более понятно:

```
0110 (6)
1010 (10)
1100 (12)
```

Для второй пары чисел:

```
1110001 (113)
0101101 (45)
1011100 (92)
```

Оператор ~ является побитовой инверсией и заменяет все 0 на 1 и наоборот. В листинге 5.19 демонстрируется использование данного оператора.

Листинг 5.19. Использование оператора ~

```
<?php
  echo ~45;     // -46
  echo "<br>";
  echo ~92;     // -93
?>
```

Такие числа являются следствием того, что все побитовые операции производятся над целыми числами, знак указывает первый символ: если он равен 0, число положительное, если он равен 1 — отрицательное. В двоичных числах ведущие нули обычно не указывают, но т. к. операция ~ производит их инвертирование — они заменяются единицами, приводя к отрицательным значениям. Так, операция `~45` в двоичном представлении выглядит следующим образом:

```
00000000000000000000000000101101 (45)
11111111111111111111111111010010 (-45)
```

Для `~92`:

```
00000000000000000000000001011100 (92)
11111111111111111111111110100011 (-92)
```

Оператор << сдвигает влево все биты первого операнда на число позиций, указанных во втором операнде (листинг 5.20). Сдвиг на отрицательное значение приводит к нулевому результату.

Листинг 5.20. Использование оператора <<

```
<?php
  echo 6 << 1;   // 12
  echo "<br>";
  echo 6 << 2;   // 24
  echo "<br>";
  echo 6 << 10;  // 6144
  echo "<br>";
  echo 6 << -1;  // 0
  echo "<br>";
  echo 6 << 31;  // 0
?>
```

В двоичном представлении эти операции выглядят следующим образом:

- `6 << 1`:

```
0110 (6)
1100 (12)
```

- `6 << 2`:

```
00110 (6)
11000 (24)
```

- `6 << 10`:

```
0000000000110 (6)
1100000000000 (6144)
```

Как видно из листинга 5.20, к нулевому результату приводит не только сдвиг на отрицательное число позиций, но и сдвиг на 31-й позиции, т. к. это приводит к тому, что все 1 уходят за границу 32-битного числа и все разряды принимают значение 0.

```
00000000000000000000000000000000 (0)
```

Оператор >> сдвигает вправо все биты первого операнда на число позиций, указанных во втором операнде. Сдвиг на отрицательное значение приводит к нулевому результату (листинг 5.21).

Листинг 5.21. Использование оператора >>

```
<?php
  echo 6144 >> 10; // 6
  echo "<br>";
  echo 45 >> 2;    // 11
  echo "<br>";
  echo 45 > 2;     // 1
?>
```

В двоичном представлении эти операции выглядят следующим образом:

- `6144 >> 10`:

```
1100000000000 (6144)
0000000000110 (6)
```

- `45 >> 2`:

```
101101 (45)
001011 (11)
```

Как видно из листинга 5.21, при использовании операторов сдвига очень легко ошибиться, указав только один знак > (оператор "больше", который рассматривается в *разд. 5.5*). В этом случае интерпретатор PHP вернет либо `TRUE`, либо `FALSE`, которые будут приведены к 1 и 0 соответственно. За такими опечатками следует следить очень внимательно, т. к. PHP не генерирует предупреждений, в то же время результат будет отличаться от ожидаемого.

Точно так же как и арифметические операторы, поразрядные операторы поддерживают сокращенную запись, позволяющую сократить выражения вида `$number = $number & $var` до `$number &= $var`. В табл. 5.7 приводятся сокращенные формы побитовых операторов.

Таблица 5.7. *Сокращенные операторы*

Сокращенная запись	Полная запись
`$number &= $var;`	`$number = $number & $var;`
`$number |= $var;`	`$number = $number | $var;`
`$number ^= $var;`	`$number = $number ^ $var;`
`$number <<= $var;`	`$number = $number << $var;`
`$number >>= $var;`	`$number = $number >> $var;`

5.5. Операторы сравнения

PHP предоставляет разработчику следующие операторы сравнения: меньше (<), меньше или равно (<=), больше (>), больше или равно (>=), равно (==), не равно (!=). В дополнение к двум последним операторам введены оператор эквивалентности (===) и оператор неэквивалентности (!===). Все эти операторы сравнивают заданные значения и возвращают `TRUE` (истина) или `FALSE` (ложь). Список логических операторов и их описание приводится в табл. 5.8.

Таблица 5.8. *Операторы сравнения*

Оператор	Описание
`$x < $y`	Оператор "меньше", возвращает `TRUE`, если переменная `$x` имеет значение строго меньше, чем значение `$y`
`$x <= $y`	Оператор "меньше или равно", возвращает `TRUE`, если переменная `$x` имеет значение меньше или равное `$y`
`$x > $y`	Оператор "больше", возвращает `TRUE`, если переменная `$x` имеет значение строго больше, чем значение `$y`
`$x >= $y`	Оператор "больше или равно", возвращает `TRUE`, если переменная `$x` имеет значение больше или равное `$y`
`$x == $y`	Оператор равенства, возвращает `TRUE`, если значение переменной `$x` равно `$y`
`$x != $y`	Оператор неравенства, возвращает `TRUE`, если значение переменной `$x` не равно `$y`
`$x === $y`	Оператор эквивалентности, возвращает `TRUE`, если значение и тип переменной `$x` равны `$y`
`$x !== $y`	Оператор неэквивалентности, возвращает `TRUE`, если либо значение, либо тип переменной `$x` не соответствует переменной `$y`

Пример использования операторов сравнения приведен в листинге 5.22.

Замечание

Сами по себе операторы сравнения практически не используются, главное их предназначение — это совместная работа с оператором `if`, который подробно рассматривается в *разд. 5.6*.

Листинг 5.22. Использование операторов сравнения

```
<?php
  echo 1 > 0;              // true
  echo 1 > 1;              // false
  echo 1 >= 1;             // true
  echo 1 < 0;              // false
  echo 1 < 1;              // false
  echo 1 <= 1;             // true
  echo 1 == 0;             // false
  echo 1 == 1;             // true
  echo 1 != 0;             // true
  echo 1 != 1;             // false
  echo 0 == "4bfj";        // false
  echo 0 == "";            // true
  echo 0 == "hello world"; // true
  echo 0 == NULL;          // true
?>
```

Замечание

Ранее уже отмечалось, что строки автоматически приводятся интерпретатором PHP к числам. Если строки содержат что-то похожее на число, например, `"5"`, `"3.14"` или `"4bfj"`, то они автоматически приводятся к числу, `5`, `3.14` и `4` соответственно. Строки, не содержащие чисел, например, `"Hello world"` автоматически приводятся к 0.

В комментариях листинга 5.22 приводятся значения `TRUE` или `FALSE`, реально в окно браузера выводится либо значение 1 для `TRUE`, либо пустая строка для `FALSE`. Последние примеры в листинге 5.52 показывают, как ведут себя операторы, если одним из операндов выступает строка. Даже если значение 0 сравнивается с `NULL` при помощи оператора равенства `==`, возвращается значение истины (`TRUE`). Однако такое поведение является неудобным в ряде ситуаций — иногда требуется проверить, равно ли значение переменной пустой строке `""`, `0` или `NULL`. Для решения этой задачи предназначены операторы эквивалентности `===` и неэквивалентности `!==`, они возвращают `TRUE` только в том случае, если значения операндов в точности соответствуют друг другу (листинг 5.23).

Листинг 5.23. Использование операторов эквивалентности

```
<?php
  echo 0 === 0;    // true
  echo 0 === "";   // false
  echo 0 === NULL; // false
  echo 1 != "1";   // false
  echo 1 !== "1";  // true
?>
```

Операции == и != имеют более низкий приоритет по сравнению с остальными. В силу этого, к примеру, выражение ($x < $a == $b < $x) является истинным только тогда, когда $x принадлежит интервалу от $a до $b. Это происходит потому, что сначала вычисляется ($x < $a), потом ($x < $b), и только затем к результатам применяется операция сравнения на равенство ==.

В заключение раздела приведем таблицы сравнения различных величин при помощи операторов равенства == (табл. 5.9) и эквивалентности === (табл. 5.10).

ЗАМЕЧАНИЕ

В табл. 5.9 и 5.10 "T" обозначает TRUE, а "F", соответственно, FALSE.

Таблица 5.9. *Сравнение величин при помощи оператора равенства* ==

	TRUE	FALSE	1	0	–1	"1"	"0"	"–1"	NULL	array()	"php"
TRUE	T	F	T	F	T	T	F	T	F	F	T
FALSE	F	T	F	T	F	F	T	F	T	T	F
1	T	F	T	F	F	T	F	F	F	F	F
0	F	T	F	T	F	F	T	F	T	F	F
–1	T	F	F	F	T	F	F	T	F	F	F
"1"	T	F	T	F	F	T	F	F	F	F	F
"0"	F	T	F	T	F	F	T	F	F	F	F
"–1"	T	F	F	F	T	F	F	T	F	F	F
NULL	F	T	F	T	F	F	F	F	T	T	F
array()	F	N	F	F	F	F	F	F	F	T	F
"php"	T	F	F	T	F	F	F	F	F	F	T

Таблица 5.10. *Сравнение величин при помощи оператора эквивалентности* ===

	TRUE	FALSE	1	0	–1	"1"	"0"	"–1"	NULL	array()	"php"
TRUE	T	F	F	F	F	F	F	F	F	F	F
FALSE	F	T	F	F	F	F	F	F	F	F	F
1	F	F	T	F	F	F	F	F	F	F	F
0	F	F	F	T	F	F	F	F	F	F	F
–1	F	F	F	F	T	F	F	F	F	F	F
"1"	F	F	F	F	F	T	F	F	F	F	F
"0"	F	F	F	F	F	F	T	F	F	F	F
"–1"	F	F	F	F	F	F	F	T	F	F	F
NULL	F	F	F	F	F	F	F	F	T	F	F
array ()	F	F	F	F	F	F	F	F	F	T	F
"php"	F	F	F	F	F	F	F	F	F	F	T

5.6. Условный оператор *if*

Условный оператор `if` имеет следующий синтаксис:

`if (`*`условие`*`)` *`оператор1`* `else` *`оператор2`*

В качестве аргумента *`условие`* оператор `if` принимает логическую переменную или выражение, возвращающее логическое значение. Если оно истинно, то выполняется *`оператор1`*. В противном случае выполняется *`оператор2`* (листинг 5.24).

Листинг 5.24. Использование условного оператора

```
<?php
  $flag = TRUE; // Истина
  if ($flag)
  {
    echo "<p>Переменная flag имеет значение TRUE</p>";
  }
  else
  {
    echo "<p>Переменная flag имеет значение FALSE</p>";
  }
?>
```

Оператор `if` проверяет условие `$flag`, и, если его значение `TRUE`, выполняется код в фигурных скобках, следующий за `if`, а если `FALSE` — код после ключевого слова `else`.

Если блок `else` не требуется, его можно опустить (листинг 5.25). Кроме этого, *`оператор1`* и *`оператор2`* необязательно должны быть составными операторами в фигурных скобках; если необходимо выполнить только один оператор, фигурные скобки можно опустить.

Листинг 5.25. Сокращенная запись оператора `if`

```
<?php
  $number = 4;
  if ($number == 4) echo "Число равно 4";
?>
```

Проверка дополнительных условий возможна также при помощи оператора `elseif`:

```
if (условие)
{
  операторы;
}
elseif (условие)
{
  операторы;
}
else
{
  операторы;
}
```

Оператор `if` может включать сколько угодно блоков `elseif`, но ключевое слово `else` в каждом `if` может присутствовать только в одном экземпляре. Как правило, в конструкциях `if...elseif...else` оператор `else` определяет, что нужно делать, если никакие другие условия не являются истинными.

В листинге 5.26 приводится пример использования конструкции `if...elseif` для проверки значения переменной.

Листинг 5.26. Использование конструкции `if...elseif`

```
<?php
  if ($char == 'a')
  {
     echo "Передан первый символ алфавита";
  }
  elseif ($char == 'b')
  {
     echo "Передан второй символ алфавита";
```

```
  }
  elseif ($char == 'c')
  {
     echo "Передан третий символ алфавита";
  }
  else
  {
     echo "Символ не входит в список первых трех символов";
  }
?>
```

PHP предоставляет также возможность альтернативного синтаксиса условного оператора — без фигурных скобок. В этом случае операторы `if`, `else` и `elseif` заканчиваются двоеточием, а сама конструкция `if` завершается обязательным ключевым словом `endif`.

В следующем примере первая таблица помещается на страницу, если только `$hdd` равно "Maxtor", а вторая — если "Seagate". Наличие оператора `endif` в этом случае обязательно, т. к. фигурная скобка, обозначающая конец блока `if`, отсутствует (листинг 5.27).

Листинг 5.27. Использование альтернативного синтаксиса оператора `if`

```
<?php
  if ($hdd == "Maxtor"):
    echo "<table>";
    echo " <caption> Maxtor </caption>";
    echo "</table>";
  elseif ($HDD == "Seagate"):
    echo "<table>";
    echo " <caption> Seagate </caption>";
    echo "</table>";
  else :
    echo "<table>";
    echo " <caption> Неизвестно </caption>";
    echo "</table>";
  endif;
?>
```

5.7. Логические операторы

В предыдущих разделах и главах в операторе `if` фигурировало лишь одно условие. Однако в повседневной практике зачастую требуется использовать несколько условий, объединенных логическими операторами. Логические операторы очень похожи на би-

нарные операторы, только вместо чисел они оперируют переменными логического типа. В табл. 5.11 приводится список логических операторов.

Таблица 5.11. *Логические операторы*

Оператор	Описание
`$x && $y`	Логическое И, возвращает `TRUE`, если оба операнда, `$x` и `$y`, равны `TRUE`, в противном случае возвращается `FALSE`
`$x and $y`	Логическое И, отличающееся от оператора `&&` меньшим приоритетом
`$x || $y`	Логическое ИЛИ, возвращает `TRUE`, если хотя бы один из операндов, `$x` и `$y`, равен `TRUE`; если оба операнда равны `FALSE`, оператор возвращает `FALSE`
`$x or $y`	Логическое ИЛИ, отличающееся от оператора `||` меньшим приоритетом
`! $x`	Возвращает либо `TRUE`, если `$x` равен `FALSE`, либо `FALSE`, если `$x` равен `TRUE`

Логическое И записывается как `&&`. Оператор возвращает `TRUE`, если оба операнда равны `TRUE`, и `FALSE` в любом другом случае. Пример использования оператора `&&` приведен в листинге 5.28.

> ***Замечание***
>
> Переменную типа `boolean`, принимающую только два значения `TRUE` и `FALSE` (или 1 и 0), часто называют флагом. Причем флаг считается установленным, если переменная принимает значение `TRUE`, и сброшенным, если переменная принимает значение `FALSE`.

Листинг 5.28. Использование оператора `&&` (логическое И)

```
<?php
  $flag1 = TRUE; // Истина
  $flag2 = TRUE; // Истина
  if ($flag1 && $flag2) echo "<p>Оба флага истинны</p>";
?>
```

Код, представленный в листинге 5.28, эквивалентен по результату коду, представленному в листинге 5.29.

Листинг 5.29. Альтернативное представление двойного условия

```
<?php
  $flag1 = TRUE; // Истина
  $flag2 = TRUE; // Истина
  if ($flag1)
  {
```

```
    if ($flag2)
    {
      echo "<p>Оба флага истинны</p>";
    }
  }
?>
```

Зачастую проще прибегнуть к двум операторам `if`, чем к двойному условию (см. листинг 5.28), т. к. совершить ошибку в нем проще. Однако некоторые конструкции, например, с участием блока `else`, наиболее эффективны только при использовании двойного условия (листинг 5.30).

Листинг 5.30. Двойное условие совместно с блоком `else`

```
<?php
  $flag1 = TRUE; // Истина
  $flag2 = TRUE; // Истина
  if ($flag1 && $flag2) // и $flag1, и $flag2 равны TRUE
  {
    echo "<p>Условие: TRUE (Оба флага истинны)</p>";
  }
  else
  {
    echo "<p>Условие: FALSE (Один из флагов ложен)</p>";
  }
?>
```

Скрипт, представленный в листинге 5.30, выведет в окно браузера фразу:

```
Условие: TRUE (Оба флага истинны)
```

Для того чтобы воспроизвести этот результат при помощи двух блоков `if`, потребуется после каждого из них поместить блок `else`, в котором будет продублирована вторая фраза (листинг 5.31), что чрезвычайно неудобно, особенно, если по мере разработки приложения потребуется изменить ее.

> ***Замечание***
>
> Следует всеми силами стараться избегать дублирующего кода: обнаруженная в нем ошибка или потребность модернизации потребует искать дублирующие блоки по всему коду — пара из них рано или поздно останется непоправленной, приводя к трудно локализуемым ошибкам.

Листинг 5.31. Неудачное дублирование кода

```
<?php
  $flag1 = TRUE; // Истина
  $flag2 = TRUE; // Истина
```

```
  if ($flag1)
  {
    if ($flag2)
    {
      echo "<p>Оба флага истины</p>";
    }
    else echo "<p>Условие: FALSE (Один из флагов ложен)</p>";
  }
  else  echo "<p>Условие: FALSE (Один из флагов ложен)</p>";
?>
```

По аналогии с битовым оператором ИЛИ (`|`) существует логическое ИЛИ, которое обозначается двумя вертикальными чертами — `||`. Заменим в листинге 5.30 оператор `&&` (И) на `||` (ИЛИ) (листинг 5.32).

Листинг 5.32. Использование оператора `||` (логическое ИЛИ)

```
<?php
  $flag1 = TRUE;  // Истина
  $flag2 = FALSE; // Ложь
  if ($flag1 || $flag2) // TRUE
  {
    echo "<p>Условие: TRUE (Один из флагов истинен)</p>";
  }
  else
  {
    echo "<p>Условие: false (Оба флага ложны)</p>";
  }
?>
```

Оператор `||` возвращает `TRUE`, если хотя бы один из его операндов равен `TRUE`, и `FALSE`, если оба операнда равны `FALSE`. В результате работы скрипта из листинга 5.32 получаем:

```
Условие: TRUE (Один из флагов истинен)
```

Операторы `&&` и `||` имеют альтернативу в виде операторов `and` и `or`, обладающих более низким приоритетом. То есть если в арифметических операциях сначала выполняется умножение и деление и лишь затем сложение и вычитание, то в логических операторах сначала выполняются `&&` и `||` и лишь затем `and` и `or`. В листинге 5.33 демонстрируется использование оператора `or`.

Листинг 5.33. Использование оператора `or` (логическое ИЛИ)

```
<?php
  $flag1 = TRUE;        // Истина
  $flag2 = FALSE;       // Ложь
```

```
  if ($flag1 or $flag2) // TRUE
  {
    echo "<p>Условие: TRUE (Один из флагов истинен)</p>";
  }
  else
  {
    echo "<p>Условие: FALSE (Оба флага ложны)</p>";
  }
?>
```

Особенность логических операторов `||` и `or` заключается в том, что если первый операнд равен `TRUE`, то в вычислении второго операнда отсутствует надобность — не зависимо от того, будет он равен `TRUE` или `FALSE`, логическое выражение все равно примет значение `TRUE`. Поэтому существует практика для функций, возвращающих в качестве значения `TRUE` или значение, которое автоматически приводится к `TRUE`, осуществлять проверку корректности выполнения при помощи операторов `||` или `or` (листинг 5.34).

> ***Замечание***
>
> Функция `exit()` останавливает работу скрипта и выводит сообщение, переданное ей в качестве параметра, в окно браузера.

Листинг 5.34. Проверка корректности выполнения функции `mysql_query()`

```
<?php
  mysql_query("SELECT VERSION()") or exit("Ошибка");
  // Дальнейшие операторы
  ...
?>
```

Если функция `mysql_query()` выполняется успешно и возвращает значение `TRUE`, то в вычислении второго операнда `or` нет надобности, функция `exit()` не выполняется — скрипт продолжает работу. Однако если функция `mysql_query()` возвращает `FALSE`, то, для того чтобы вычислить логическое выражение `or`, необходимо получить результат от второго операнда: неизбежно выполняется функция `exit()`, сообщающая о возникшей ошибке.

Следует отметить, что функция `mysql_query()` возвращает дескриптор, и если он требуется для дальнейших действий, выражение проверки может выглядеть так, как это демонстрируется в листинге 5.35.

Листинг 5.35. Получение дескриптора

```
<?php
  $ver = mysql_query("SELECT VERSION()") or exit("Ошибка");
  // Дальнейшие операторы
  ...
?>
```

Именно тут и проявляется необходимость двух версий операторов ИЛИ с разным приоритетом (`or` и `||`). Оператор `or` имеет меньший приоритет, чем оператор "равно" `=`, сначала выполняется приравнивание результата функции `mysql_query()` и лишь затем сравнение. Если бы вместо оператора `or` использовался оператор `||`, имеющий больший приоритет, чем `=`, сначала бы осуществлялось сравнение, а затем результат этого сравнения присваивался бы переменной `$ver`, в итоге она содержала бы не дескриптор результирующей таблицы, а значение `TRUE`. Таким образом, если для проверки корректности выполнения функции `mysql_query()` использовался бы оператор `||`, необходимо было бы воспользоваться дополнительными кавычками (листинг 5.36).

Листинг 5.36. Проверка корректности выполнения функции при помощи ||

```
<?php
  ($ver = mysql_query("SELECT VERSION()")) || exit("Ошибка");
  // Дальнейшие операторы
  ...
?>
```

В любом случае, лучше избегать проверок, основанных на операторах `||` и `or`. Вместо этого лучше использовать оператор `if`, в этом случае скрипты получаются пусть и несколько более объемными, зато более читабельными.

Стиль программирования, приведенный в листингах 5.34—5.36, был заимствован из Perl. В нем такие конструкции выглядят естественно и гармонично увязаны с остальными конструкциями языка. В PHP, который рекомендует более строгий стиль, подходящий для промышленной разработки, стиль программирования, основанный на эксплуатации приоритета операторов, не приветствуется.

Иногда бывает необходимо проверить условие безальтернативно на ложность или истинность. В этом случае в применении ключевого слова `else` и следующих за ним операторов нет необходимости. Допустим, функция `func()` возвращает истину или ложь, и в зависимости от этого завершается работа функции успешно или неудачно. Например, функция `mysql_query($query)` возвращает `TRUE`, если запрос к базе данных MySQL, размещенный в строке `$query`, успешно выполнился, и `FALSE` — в противном случае. Если требуется среагировать только на удачное обращение, скрипт может выглядеть так, как это представлено в листинге 5.37.

Листинг 5.37. Выводим сообщение, если работа функции успешно завершена

```
<?php
  if (mysql_query($query))
  {
    echo "<p>Данные успешно занесены в базу данных.<p>";
  }
?>
```

Если же необходимо среагировать только в том случае, если работа функции завершилась неудачно, код может выглядеть так, как это представлено в листинге 5.38.

Листинг 5.38. Выводим сообщение, если работа функции прервана из-за ошибки

```
<?php
  if (mysql_query($query))
  {}
  else
  {
    echo "<p>Данные не были занесены в базу данных.<p>";
  }
?>
```

Пустой составной оператор `{}` вполне допускается синтаксисом языка. Однако способ, продемонстрированный в листинге 5.38, не является элегантным. Здесь лучше воспользоваться оператором отрицания `!`, применение которого к переменной меняет ее значение с `TRUE` на `FALSE` и наоборот (листинг 5.39).

Листинг 5.39. Использование оператора отрицания !

```
<?php
  if (!mysql_query($query))
  {
    echo "<p>Данные не были занесены в базу данных.<p>";
  }
?>
```

Операторы *оператор1* и *оператор2* в условном операторе, в свою очередь, также могут быть условными, что позволяет организовывать цепочки проверок любой степени вложенности. В этих цепочках каждый условный оператор, в свою очередь, может быть как полным, так и сокращенным. В связи с этим возможны ошибки неоднозначного сопоставления `if` и `else`. Синтаксис языка предполагает, что при вложенных условных операторах каждое `else` соответствует ближайшему `if`. В качестве такого неоднозначного примера можно привести конструкцию, представленную в листинге 5.40.

Листинг 5.40. Пример неоднозначной записи сокращенного условного оператора

```
<?php
  $x = 1;
  $y = 1;
  if ($x == 1) // первый условный оператор
    if ($y == 1) echo "x == 1 and y == 1"; // второй условный оператор
    else echo("x != 1");
?>
```

При `$x`, равном 1, и `$y`, равном 1, совершенно справедливо печатается фраза `"x == 1 and y == 1"`. Однако фраза `"x != 1"` будет напечатана при `$x`, равном 1, и при `$y`, не равном 1, т. к. `else` соответствует ближайшему `if`. Первый условный оператор, где проверяется условие `$x == 1`, является сокращенным, и в качестве *оператор1* включает полный условный оператор, где проверяется условие `$y == 1`. Таким образом, проверка этого условия выполняется только при `$x`, равном 1. Простым правильным решением этой задачи является применение фигурных скобок, т. е. построение составного оператора. В этом случае, ограничивается область действия второго условного оператора, и он становится неполным. Тем самым первый оператор превращается в полный условный (листинг 5.41).

Листинг 5.41. Использование фигурных скобок

```
<?php
  $x = 1;
  $y = 1;
  if ($x == 1)
  {
    if ($y == 1) echo("x == 1 and y == 1");
  }
    else echo("x != 1");
?>
```

5.8. Условный оператор *x ? y : z*

PHP так же, как и C++, Java, предоставляет возможность заменять блоки `if...else` условной операцией, которая имеет следующий синтаксис:

выражение1 `?` *выражение2* `:` *выражение3*

Первым вычисляется значение *выражение1*. Если оно истинно, то вычисляется значение *выражение2*, которое и становится результатом. Если *выражение1* ложно, то в качестве результата берется *выражение3*. Классическим примером условной операции является получение абсолютного значения переменной (листинг 5.42).

Листинг 5.42. Использование условного оператора

```
<?php
  $x = -17;
  $x = $x < 0 ? -$x : $x;
  echo $x; // 17
?>
```

Если переменная `$x` оказывается меньше нуля — у нее меняется знак, если переменная оказывается больше нуля, она возвращается без изменений. Основное назначение условного оператора — сократить конструкцию `if` до одной строки, если это не приводит к снижению читабельности.

Начиная с PHP 6.0, допускается не указывать средний параметр условного оператора. Такой синтаксис позволяет в одну строку проверять, инициализирована ли переменная, и если она не инициализирована, присваивать ей начальное значение. В листинге 5.43 проверяется, имеет ли переменная `$x` значение, отличное от `0`, `NULL`, пустой строки (и вообще всего, что может рассматриваться как `FALSE`). Если переменная инициализирована — ее значение остается неизменным, если нет — ей присваивается единица.

Листинг 5.43. Средний параметр в условной конструкции не обязателен

```
<?php
  $x = $x ?: 1;
  echo $x; // 1
?>
```

Замечание

Несмотря на то, что условная операция позволяет записать текст в гораздо более компактной форме, по возможности ее следует избегать, т. к. код, созданный с ее помощью, менее читабельный, чем код, организованный посредством оператора `if`.

5.9. Переключатель *switch*

Переключатель `switch` предоставляет более удобные средства для организации множественного выбора, чем оператор `elseif`. Синтаксис данного оператора таков:

```
switch(выражение)  // переключающее выражение
{
  case значение1:  // выражение 1
    операторы;     // блок операторов
    break;
  case значение2:   // выражение 2
    операторы;
    break;
  default:
    операторы;
}
```

Управляющая структура `switch` передает управление тому из помеченных операторов `case`, для которого значение константного выражения совпадает со значением переключающего выражения. Сначала анализируется переключающее *выражение* и осуществляется переход к той ветви программы, для которой его значение совпадает с выражением в операторе `case`. Далее следует выполнение оператора или группы операторов до тех пор, пока не встретится ключевое слово `break`, которым обозначается выход из конструкции `switch`. Если же значение переключающего выражения не совпадает ни с одним из константных выражений, то выполняется переход к оператору, помеченному меткой `default`. В каждом переключателе может быть не более од-

ной метки `default`. Ключевые слова `break` и `default` не являются обязательными, и их можно опускать.

В листинге 5.44 в зависимости от того, какое значение принимает переменная `$answer`, в окно браузера выводится та или иная фраза. Если переменная `$answer` принимает значение "yes", выводится фраза "Продолжаем работу!", если переменная принимает значение "no", выводится фраза "Завершаем работу". Если `$answer` принимает любое другое значение, управление передается блоку `default` и выводится фраза "Некорректный ввод".

Листинг 5.44. Пример использования оператора `switch`

```
<?php
  $answer = "yes";
  switch($answer)
  {
    case "yes":
      echo "Продолжаем работу!";
      break;
    case "no":
      echo "Завершаем работу";
      break;
    default:
      echo "Некорректный ввод";
  }
?>
```

Операторы в блоке `case` могут быть заключены в необязательные фигурные скобки. Конструкции в листингах 5.44 и 5.45 являются эквивалентными.

Листинг 5.45. Альтернативная запись оператора `switch`

```
<?php
  $answer = "yes";
  switch($answer)
  {
    case "yes":
    {
      echo "Продолжаем работу!";
      break;
    }
    case "no":
    {
```

```
      echo "Завершаем работу";
      break;
    }
    default:
      echo "Некорректный ввод";
  }
?>
```

Если в операторе `switch` пропущен оператор `break`, то на экран будут выведены сразу два сообщения. В листинге 5.46 выводятся названия нечетных целых десятичных цифр от 1 до 9, не меньше введенной.

Листинг 5.46. Вывод чисел

```
<?php
  switch($number)
  {
    case 1:
      echo "один ";
    case 2: case 3:
      echo "три ";
    case 4: case 5:
      echo "пять ";
    case 6: case 7:
      echo "семь ";
    case 8: case 9:
      echo "девять ";
      break;
    default:
      echo ("Это либо не число, либо число больше 9 или меньше 1");
  }
?>
```

Таким образом, если переменная `$number` примет значение 2, то результат работы скрипта может выглядеть следующим образом:

```
три пять семь девять
```

Оператор `switch` переключает скрипт на позицию `case 2`, пропуская позицию `case 1`, и далее скрипт выполняется до ближайшего оператора `break`.

Помимо синтаксиса, представленного в начале раздела, PHP позволяет использовать альтернативный синтаксис без фигурных скобок с применением ключевых слов `switch:` и `endswitch`. В листинге 5.47 представлена альтернативная запись оператора `switch` для скрипта из листинга 5.44.

Листинг 5.47. Альтернативная форма оператора `switch`

```
<?php
  $answer = "yes";
  switch($answer):
    case "yes":
      echo "Продолжаем работу!";
      break;
    case "no":
      echo "Завершаем работу";
      break;
    default:
      echo "Некорректный ввод";
  endswitch;
?>
```

Конструкция `switch` может быть представлена при помощи оператора `if...elseif`. Так, скрипт из листинга 5.47 можно переписать следующим образом (листинг 5.48).

Листинг 5.48. Оператор `switch` можно заменить оператором `if...elseif`

```
<?php
  $answer = "yes";
  if ($answer == "yes")
  {
    echo "Продолжаем работу!";
  }
  elseif ($answer == "no")
  {
    echo "Завершаем работу";
  }
  else
  {
    echo "Некорректный ввод";
  }
?>
```

На первый взгляд может показаться, что оператор `if...elseif` более гибкий, т. к. позволяет оперировать сложными условиями с применением логических операторов, например, так, как это демонстрируется в листинге 5.49.

Листинг 5.49. Использование сложных логических условий

```
<?php
  $number = 120;
  if ($number > 0 && $number <= 10)
    echo $number." меньше 10 и больше 0";
  elseif ($number > 10 && $number <= 100)
    echo $number." меньше 100 и больше 10";
  elseif ($number > 100 && $number <= 1000)
    echo $number." меньше 1000 и больше 100";
  else
    echo $number." больше 1000 или меньше 0";
?>
```

Однако оператор `switch` позволяет реализовывать точно такие же построения, как это продемонстрировано в листинге 5.49, благодаря тому, что логическое выражение можно сравнивать со значением `TRUE` или `FALSE`. В листинге 5.50 приводится скрипт, аналогичный скрипту 5.49 по функциональности.

Листинг 5.50. Использование сложных логических выражений в `switch`

```
<?php
  $number = 120;
  switch(TRUE)
  {
    case ($number > 0 && $number <= 10):
      echo $number." меньше 10 и больше 0";
      break;
    case ($number > 10 && $number <= 100):
      echo $number." меньше 100 и больше 10";
      break;
    case ($number > 100 && $number <= 1000):
      echo $number." меньше 1000 и больше 100";
      break;
    default:
      echo $number." больше 1000 или меньше 0";
      break;
  }
?>
```

В заключение следует заметить, что вместо традиционного оператора `break`, прерывающего выполнение оператора, когда его цели достигнуты, можно использовать ключевое слово `continue` (листинг 5.51). В рамках оператора `switch` эти два оператора

эквивалентны, разница между ними имеет значение только в циклах, которым посвящены *разд. 5.10—5.12*.

> ***Замечание***
>
> Не следует заменять `break` на `continue` только из-за того, что это допускает синтаксис, уже многие десятилетия во всех C-подобных языках программирования совместно с оператором `switch` используется `break`. Отклонение от этого правила потребует от большинства программистов дополнительных усилий на разбор кода и, следовательно, приведет к уменьшению его читабельности.

Листинг 5.51. Оператор `break` может быть заменен оператором `continue`

```
<?php
  $number = 1;
  switch(true)
  {
    case ($number > 0 && $number <= 10):
      echo $number." меньше 10 и больше 0";
      continue;
    case ($number > 10 && $number <= 100):
      echo $number." меньше 100 и больше 10";
      continue;
    case ($number > 100 && $number <= 1000):
      echo $number." меньше 1000 и больше 100";
      continue;
    default:
      echo $number." больше 1000 или меньше 0";
      continue;
  }
?>
```

5.10. Цикл *while*

Оператор `while` называется оператором цикла с предусловием (листинг 5.52). Синтаксис оператора выглядит следующим образом:

```
while (условие)
{
  операторы;
}
```

При входе в цикл вычисляется *условие*, и если его значение истинно (`TRUE`), выполняется тело цикла. После завершения выполнения операторов, выражение-условие вычисляется повторно, и если оно по-прежнему равно `TRUE`, операторы тела цикла выполняются повторно. Это происходит до тех пор, пока значение выражения-условия

не станет ложным (FALSE). Тело цикла не обязательно должно быть заключено в фигурные скобки, если требуется выполнить только один оператор — они могут быть опущены.

> ***Замечание***
>
> Необходимо заботиться о том, чтобы рано или поздно *условие* принимало значение FALSE, иначе произойдет образование бесконечного цикла, и приложение зависнет.

Листинг 5.52. Использование оператора `while`

```
<?php
  $do_while = TRUE;
  $i = 1;
  while($do_while)
  {
    echo $i."<br>";
    $i++;
    // Условие выхода из цикла
    if ($i > 5) $do_while = FALSE;
  }
?>
```

Результатом выполнения кода из листинга 5.52 являются числа от 1 до 5, расположенные в столбик:

```
1
2
3
4
5
```

Тело цикла, расположенное между фигурными скобками, выполняется до тех пор, пока флаг $do_while не примет значение FALSE. При каждой итерации цикла значение счетчика $i увеличивается на единицу при помощи оператора инкремента $i++. Как только значение переменной $i достигает 6 и становится больше 5, переменная $do_while принимает значение FALSE и на следующей итерации оператор while прекращает свою работу.

Изменение значения флага $do_while на FALSE не приводит к немедленному выходу из цикла, цикл прекращает свою работу только после того, как начинается новая итерация (листинг 5.53).

Листинг 5.53. Цикл `while`

```
<?php
  $do_while = TRUE;
  $i = 0;
```

```
  while ($do_while)
  {
    $i++;
    // Условие выхода из цикла
    if ($i > 5) $do_while = FALSE;
    echo $i."<br>";
  }
?>
```

Результатом работы скрипта из листинга 5.53 будет колонка цифр от 1 до 6:

```
1
2
3
4
5
6
```

Тем не менее, часто требуется досрочно прекратить выполнение цикла. Для этого применяется уже упоминавшийся оператор `break`. При обнаружении этого оператора цикл прекращает свою работу немедленно (листинг 5.54).

Листинг 5.54. Использование оператора `break`

```
<?php
  $i = 0;
  while (TRUE)
  {
    $i++;
    // Условие выхода из цикла
    if ($i > 5) break;
    echo $i."<br>";
  }
?>
```

Результатом работы скрипта из листинга 5.54 является колонка цифр от 1 до 5. Когда `$i` становится больше 5, последний оператор `echo` в цикле просто не выполняется, т. к. `break` переводит текущий поток программы на первый оператор после цикла `while`.

Иногда требуется досрочно прекратить текущую итерацию и перейти сразу к началу следующей. Для этого применяется оператор `continue` (листинг 5.55).

> ***Замечание***
>
> Оператор `switch` можно рассматривать как цикл, всегда выполняющийся один раз, поэтому в нем операторы `break` и `continue` эквивалентны по своему действию.

Листинг 5.55. Пример оператора `continue`

```
<?php
  $i = 0;
  while (TRUE)
  {
    $i++;
    // Досрочно прекращаем текущую итерацию
    if ($i < 4) continue;
    // Условие выхода из цикла
    if ($i > 5) break;
    echo $i."<br>";
  }
?>
```

Результатом выполнения этого примера будут выведенные в столбик цифры:

```
4
5
```

Во вложенных циклах операторы `break` по умолчанию действуют каждый на своем уровне вложения (листинг 5.56).

Листинг 5.56. Использование оператора `break` во вложенном цикле

```
<?php
  $i = $j = 0;
  while (TRUE)
  {
    while (TRUE)
    {
      $i++;
      if ($i > 5) break;
      echo $i."<br>";
    }
    $i = 0;
    $j++;
    if ($j > 5)
    {
      break;
    }
  }
?>
```

Скрипт из листинга 5.56 выводит пять раз подряд последовательность из пяти чисел от 1 до 5. Однако оператор `break` может управлять не только своим циклом, но и внешним циклом, для этого следует указать его номер. По умолчанию, если передать `break` значение 1, его действия будут относиться к текущему циклу. В листинге 5.57 действия операторов `break` аналогичны скрипту 5.56.

Листинг 5.57. Указания номера цикла в операторе `break`

```
<?php
  $i = $j = 0;
  while (TRUE)
  {
    while (TRUE)
    {
      $i++;
      if ($i > 5) break 1;
      echo $i."<br>";
    }
    $i = 0;
    $j++;
    if ($j > 5)
    {
      break 1;
    }
  }
?>
```

Если оператору `break`, расположенному во внутреннем цикле, передать номер 2, то вместо внутреннего цикла он будет прерывать внешний цикл, и скрипт выведет последовательность от 1 до 5 только один раз вместо пяти. В листинге 5.58 приводится пример тройного вложенного цикла, который прерывается из самого внутреннего цикла.

Листинг 5.58. Прерывание внешнего цикла из внутреннего

```
<?php
  $i = 0;
  while (TRUE)
  {
    while (TRUE)
    {
      while (TRUE)
      {
        $i++;
```

```
        if ($i > 5) break 3;
        echo $i."<br>";
      }
    }
  }
?>
```

Цикл `while` может иметь более замысловатую условную конструкцию. Например, цикл, представленный в листинге 5.59, выводит столбик цифр от 4 до 1.

Листинг 5.59. Использование оператора декремента совместно с циклом `while`

```
<?php
  $number = 5;
  while (--$number)
  {
     echo $number."<br>";
  }
?>
```

Если использовать постфиксную форму оператора декремента `--`, как это представлено в листинге 5.60, выводится столбик цифр от 4 до 0.

Листинг 5.60. Использование оператора постфиксной формы оператора декремента

```
<?php
  $number = 5;
  while ($number--)
  {
     echo $number."<br>";
  }
?>
```

Работа циклов из листингов 5.59 и 5.60 основана на том факте, что любое число больше нуля преобразуется к значению `TRUE` (истина), а значение 0 к `FALSE` (ложь). Как только значение `$number` принимает значение 0 — цикл прекращает свою работу. В листинге 5.59 используется префиксная форма декремента, в которой от числа сначала вычитается единица, и лишь затем возвращается значение, поэтому цифра 0 не выводится. В листинге 5.60 по достижении `$number` значения 1 в операторе `while` сначала возвращается значение 1, и лишь затем из `$number` вычитается единица, приравнивая его значение 0. Цикл выводит значение 0 и останавливает работу лишь на следующей итерации.

Как и в случае операторов `if` и `switch`, оператор `while` поддерживает синтаксис без фигурных скобок, с использованием ключевых слов `while` и `endwhile`. В листинге 5.61

приводится скрипт из листинга 5.60, модифицированный в соответствии с альтернативным синтаксисом.

Листинг 5.61. Альтернативный синтаксис оператора `while`

```
<?php
  $number = 5;
  while ($number--):
     echo $number."<br>";
  endwhile;
?>
```

5.11. Цикл *do ... while*

Цикл `do...while` в отличие от цикла `while` проверяет условие выполнения цикла не в начале итерации, а в конце и имеет следующий синтаксис:

```
do
{
 операторы;
} while (условие);
```

Такой цикл всегда будет выполнен хотя бы один раз. После выполнения тела цикла вычисляется выражение *условие*, и, если оно истинно (`TRUE`), вновь выполняется тело цикла. В листинге 5.62 ноль всегда будет добавлен в список, независимо от значения условия `(++$i <= 5)`.

> ***ЗАМЕЧАНИЕ***
>
> При использовании цикла `do...while` также допускается применение операторов `break` и `continue`, порядок использования которых рассмотрен в *разд. 5.10*.

Листинг 5.62. Использование оператора `do...while`

```
<?php
  $i = 0;
  do
  {
    echo $i."<br>";
  } while (++$i <= 5);
?>
```

Результатом выполнения этого примера будут выведенные в столбик числа от 0 до 5.

В отличие от остальных операторов цикла, а также операторов `if` и `switch`, данный вид цикла не поддерживает альтернативный синтаксис.

5.12. Цикл *for*

Итерационный цикл `for` имеет следующий синтаксис:

```
for (начало; условие; тело)
{
  операторы;
}
```

Здесь оператор *начало* (инициализация цикла) — последовательность определений и выражений, разделяемая запятыми. Все выражения, входящие в инициализацию, вычисляются только один раз при входе в цикл. Как правило, здесь устанавливаются начальные значения счетчиков и параметров цикла. Смысл выражения-условия (*условие*) такой же, как и у циклов с пред- и постусловиями — цикл выполняется до тех пор, пока *условие* истинно (`TRUE`), и прекращает свою работу, когда оно ложно (`FALSE`). При отсутствии выражения-условия предполагается, что его значение всегда истинно. Выражения *тело* вычисляются в конце каждой итерации после выполнения тела цикла.

> ***Замечание***
>
> При использовании цикла `for` также допускается применение операторов `break` и `continue`, порядок использования которых рассмотрен в *разд. 5.11*.

В листинге 5.63 при помощи цикла выводятся цифры от 0 до 4.

Листинг 5.63. Использование цикла `for`

```
<?php
  for ($i = 0; $i < 5; $i++)
  {
    echo $i."<br>";
  }
?>
```

Здесь значение `$i` в самом начале работы цикла принимает значение, равное нулю (`$i = 0`), на каждой итерации цикла значение переменной `$i` увеличивается на единицу при помощи оператора инкремента (`$i++`). Цикл выполняет свою работу до тех пор, пока значение `$i` меньше 5. Результат работы скрипта из листинга 5.63 выглядит следующим образом:

```
0
1
2
3
4
```

В листинге 5.64 приводится пример вывода цифр от 5 до 1 при помощи цикла с декрементом.

Листинг 5.64. Использование декремента в цикле `for`

```
<?php
  for ($i = 5; $i; $i--)
  {
    echo $i."<br>";
  }
?>
```

Результат работы цикла из листинга 5.64 выглядит следующим образом:

```
5
4
3
2
1
```

Переменная `$i` инициируется значением 5, на каждой итерации цикла это значение уменьшается на единицу при помощи оператора декремента (`$i--`). Интересно отметить, что в качестве условия выступает сама переменная `$i`; как только ее значение достигает 0, цикл прекращает свою работу.

Рассмотренные варианты использования цикла `for` являются традиционными, но не единственными, так цикл `for` допускает отсутствие тела цикла, т. к. все вычисления могут быть выполнены в выражении `body`. Цикл в листинге 5.64 может быть переписан так, как это представлено в листинге 5.65.

ЗАМЕЧАНИЕ

Использование конструкции `echo` в цикле `for` не допускается.

Листинг 5.65. Альтернативное использование цикла `for`

```
<?php
  for ($i = 5; $i; print $i, print "<br>", $i--);
?>
```

Здесь вместо оператора декремента `$i--` используется последовательность операторов, разделенных запятой — `print $i, print "<br>", $i--`. Такая форма записи вполне допускается синтаксисом PHP (и любого другого C-подобного языка программирования), но не приветствуется, т. к. здорово уменьшает читабельность кода.

Допускается использование нескольких операторов и в блоке инициализации (листинг 5.66).

ЗАМЕЧАНИЕ

Код в листинге 5.66 является ярким примером того, как не следует программировать. Здесь он приведен лишь для демонстрации особенностей работы цикла `for`.

Листинг 5.66. Использование запятых для разделения операторов

```
<?php
  for ($i = 6, $j = 0;
       $i != $j;
       print $i." - ".$j, print "<br>", $i--, $j++);
?>
```

Результат работы цикла из листинга 5.66 выглядит следующим образом:

```
6 - 0
5 - 1
4 - 2
```

Переменная `$i` инициируется значением 6, а переменная `$j` — значением 0. За одну итерацию значение `$i` уменьшается на единицу, а значение `$j` увеличивается. Как только они становятся равны (это происходит, когда их значения достигают 3) — цикл прекращает свою работу. На каждой из итераций выводится значение цикла.

До сих пор рассматривались скрипты, которые увеличивают или уменьшают значения счетчиков лишь на единицу. Однако циклы вполне допускают использование в качестве шага и других значений. Например, в цикле, который представлен в листинге 5.67, выводятся значения от 0 до 100 с интервалом в 5.

Листинг 5.67. Использование шага, равного 5

```
<?php
  for ($i = 0; $i <= 100; $i += 5) echo $i."<br>";
?>
```

Выражения *начало*, *условие* и *тело* не обязательно должны присутствовать. Так, в листинге 5.68 приводится пример цикла `for`, в котором отсутствуют выражения *начало* и *тело*.

Листинг 5.68. Отсутствие выражений *начало* и *тело*

```
<?php
  $do_for = true;
  $i = 0;
  for (; $do_for; )
  {
     $i++;
     echo $i."<br>";
     if ($i > 5) $do_for = false;
  }
?>
```

Код в листинге 5.68 может быть переписан следующим образом (листинг 5.69).

Замечание

Такое использование цикла `for` не является типичным; если число циклов заранее не известно, традиционно используется цикл `while`.

Листинг 5.69. Отсутствие выражений в цикле `for`

```
<?php
  $i = 0;
  for (;;)
  {
     $i++;
     echo $i."<br>";
     if ($i > 5) break;
  }
?>
```

Циклы `for`, как и любые другие циклы, могут быть вложенными друг в друга. Одним из классических примеров на вложенные циклы является программа для нахождения простых чисел (листинг 5.70).

Замечание

Напомним, что простыми называются числа, которые делятся только на 1 и на самих себя.

Листинг 5.70. Программа нахождения простых чисел

```
<?php
  for ($i = 2; $i < 100; $i++)
  {
    for ($j = 2; $j < $i; $j++)
    {
      if (($i%$j)!=0) continue;
      else
      {
        $flag = true;
        break;
      }
    }
    if (!$flag) echo $i." ";
    $flag = false;
  }
?>
```

Результат:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Скрипт в листинге 5.70 реализован в виде двух вложенных циклов, в которых осуществляются перебор и проверка остатка от деления пары чисел. Первое число изменяется от 2 до 100, а второе — от 2 до значения первого числа. Если остаток от деления не равен нулю, то по оператору `continue` осуществляется продолжение внутреннего цикла, поскольку этот оператор, как вы помните, предписывает программе перейти на следующую итерацию цикла. Если же остаток от деления равен нулю, то происходит выход из внутреннего цикла по оператору `break`. При этом логическая переменная `$flag`, в которую устанавливается признак деления, принимает значение `true`. По окончании внутреннего цикла производится анализ логической переменной и вывод простого числа.

Как и в случае операторов `if` и `switch`, оператор `for` поддерживает синтаксис без фигурных скобок, с использованием ключевых слов `for` и `endfor`. В листинге 5.71 приводится скрипт из листинга 5.63, модифицированный в соответствии с альтернативным синтаксисом.

Листинг 5.71. Альтернативный синтаксис оператора `for`

```
<?php
  for($i = 0; $i < 5; $i++):
    echo $i."<br>";
  endfor;
?>
```

5.13. Включение файлов

Язык программирования PHP предоставляет в распоряжение разработчика четыре конструкции, позволяющие включать в скрипт другой скрипт или файл. Список конструкций и их описание приводится в табл. 5.12.

Таблица 5.12. *Конструкции включения*

Конструкция	Описание
`include("index.php")`	Включает файл index.php, в случае отсутствия файла index.php выводится предупреждение, при этом скрипт продолжает работу
`include_once("index.php")`	Включает файл index.php только один раз, сколько бы вызов конструкции не встречалось. В случае отсутствия файла index.php выводит предупреждение, при этом скрипт продолжает работу
`require("index.php")`	Включает файл index.php, в случае отсутствия файла index.php выводится предупреждение и прекращается работа скрипта

Таблица 5.12 *(окончание)*

Конструкция	Описание
`require_once("index.php")`	Включает файл index.php только один раз, сколько бы вызовов конструкции не встречалось. В случае отсутствия файла index.php выводит предупреждение и прекращается работа скрипта

Конструкции `include()` и `require()` позволяют включить файл в PHP-скрипт. Обе принимают единственный аргумент — путь к включаемому файлу, и результатом их действия является подстановка содержимого файла в место их вызова в исходном скрипте. Если в качестве включаемого скрипта выступает PHP-скрипт, то сначала происходит его подстановка в исходный скрипт, а затем интерпретация результирующего скрипта (листинг 5.72).

> ***Замечание***
>
> Круглые скобки, в которые заключается путь к файлу, не являются обязательными.

Листинг 5.72. Использование инструкции `include`

```
<?php
  echo "first<br>";
  include("second.php");
  echo "first<br>";
?>
```

Пусть файл second.php содержит код, представленный в листинге 5.73.

Листинг 5.73. Файл second.php (вариант 1)

```
<?php
  echo "second<br>";
?>
<h3>Текст не обязательно должен выводиться оператором echo</h3><br>
```

Результат вызова скрипта, приведенного в листинге 5.73, представлен на рис. 5.1.

Выполнение включаемого скрипта second.php можно прекратить досрочно, объявив во включаемом файле оператор `return`.

Листинг 5.74. Файл second.php (вариант 2)

```
<?php
  return;
  echo "second";
?>
<h3>Текст не обязательно должен выводиться оператором echo</h3><br>
```

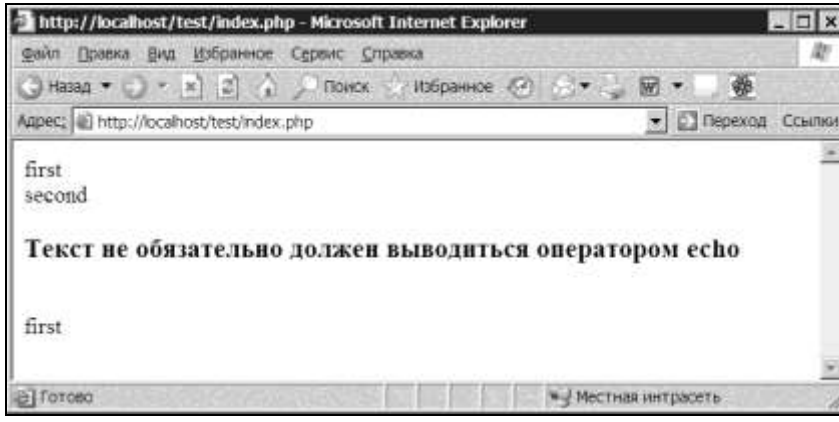


Рис. 5.1. Результат работы скрипта с инструкцией `include`

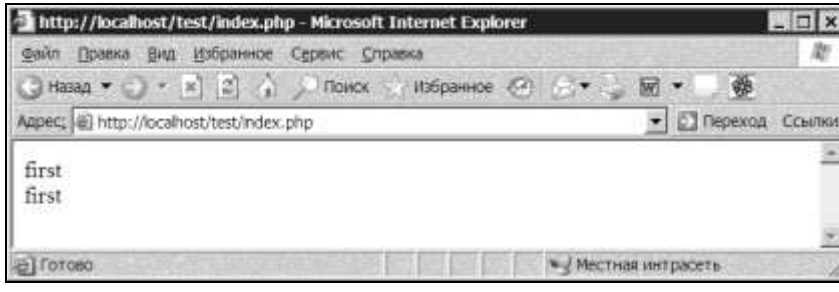


Рис. 5.2. Досрочное прекращение выполняемого файла

Так после исправления содержимого файла second.php из листинга 5.73 на содержимое, представленное в листинге 5.74, результат работы скрипта будет выглядеть так, как это представлено на рис. 5.2.

Вызов оператора `return` приводит к немедленному выходу из файла, а все последующие за `return` операторы игнорируются. Однако внешний скрипт, вызывающий оператор `include`, свою работу не прекращает.

Оператор `return` может возвращать значение. В листинге 5.75 приводится содержимое файла second.php, который вызывает оператор `return`, передавая ему в качестве параметра строку "Hello, world!".

> ***ЗАМЕЧАНИЕ***
>
> В качестве возвращаемого значения могут выступать не только строки, но и числа, логические переменные и даже объекты и массивы.

Листинг 5.75. Оператор `return` может возвращать значение

```
<?php
  return "Hello, world!";
?>
```

Если теперь включить такой файл в другой файл, никаких видимых изменений не последует, строка "Hello, world!" не будет включена в главный файл. Однако, если конструкцию `include` (или любую другую из табл. 5.11) присвоить переменной, последняя будет инициализирована строкой "Hello, world!" (листинг 5.76).

Листинг 5.76. Переменная получает значение, возвращаемое оператором `return`

```
<?php
  $return = include "second.php";
  echo $return; // Hello, world!
?>
```

В качестве пути к файлу может быть указан сетевой путь к файлу. В этом случае следует помнить, что функция `include()` вернет файл в том виде, в котором она получает его от Web-сервера, т. е. не следует ждать PHP-код — в сетевом варианте функции `include()` будет получен лишь результат работы скрипта, но не его код.

Включаемый функцией `include()` файл не обязательно должен быть PHP-скриптом; к этой функции часто прибегают для включения объемных текстовых вставок, которые могли бы затруднить чтение кода.

Как было упомянуто в начале раздела, помимо функции `include()` существует функция `require()`, выполняющая аналогичные действия. Различия в этих функциях заключается в их реакции на отсутствие включаемого файла. Если в случае функции `include()` включаемый файл отсутствует, то реакцией на это является единственно вывод в окно браузера соответствующего предупреждения. Отсутствие файла по пути, который передается в качестве аргумента функции `require()`, приводит к остановке скрипта.

> ***Замечание***
>
> Для обеих функций существуют аналоги с суффиксом _once: `include_once()` и `require_once()`, позволяющие включить файл в документ только один раз, не зависимо от того, сколько попыток включения предпринимается. Это удобно использовать при вложенных включениях, во избежание ошибок при повторном включении файлов, содержащих объявления функций.

При первом знакомстве с операторами `include()` и `require()` возникает соблазн передавать путь к файлам через GET-параметры, которые затем передавать непосредственно операторам `include()` или `require()`. Однако такой подход таит в себе опасность, т. к. злоумышленник получает возможность выполнения произвольного PHP-кода на сервере. Рассмотрим уязвимость более подробно, для этого создадим файл index.php с содержимым, представленным в листинге 5.77.

> ***Замечание***
>
> Работа с GET-параметрами более подробно рассматривается в *главе 8*.

Листинг 5.77. Передача GET-параметра оператору `include()`

```
<?php
  include($_GET['path']);
?>
```

Теперь, создав множество файлов req1.php, req2.php, ..., req3.php, мы получаем возможность динамически подключать их, просто указывая их имена в GET-параметре **path** строки запроса, например, **http://www.site.ru/index.php?path=req2.php**. Однако злоумышленник в качестве пути к файлу может подставить свой путь, в том числе и сетевой, например, **http://www.site.ru/index.php?path=http://www.xaker.ru/q.php**. Если файл имеет расширение php, ничего страшного не произойдет, будет подставлен результат выполнения PHP-файла, т. е. HTML-код, при помощи которого деструктивные действия невозможны. Однако если вместо PHP-файла будет передан txt-файл с PHP-инструкциями — они будут выполнены. Причем принудительное добавление расширения файла, как это представлено в листинге 5.78, не устраняет уязвимости.

Листинг 5.78. Принудительное назначение расширения включаемого файла

```
<?php
  include($_GET['path'].".php");
?>
```

В этом случае злоумышленник может создать PHP-скрипт, который вместо HTML-кода генерирует текст другого PHP-скрипта, или назначить на своем сервере в качестве MIME-типа PHP-файлов текстовый тип `text/plain`.

Часто возникает задача запретить прямое обращение к включаемым файлам. Для этого удобно воспользоваться константами *(см. главу 4)*. Достаточно в главном файле определить константу при помощи функции `define()`, а во включаемых файлах проверять ее существование при помощи функции `defined()`. В листинге 5.79 приводится главный файл, который определяет константу `SECOND` и включает файл second.php.

Листинг 5.79. Использование инструкции `include`

```
<?php
  define("SECOND", 1);
  echo "first<br>";
  include("second.php");
  echo "first<br>";
?>
```

Файл second.php, содержимое которого представлено в листинге 5.80, проверяет, определена ли константа `SECOND`, и если константа не определена, прекращает свою работу вызовом оператора `return`.

Листинг 5.80. Файл second.php

```
<?php
  if (!defined("SECOND ")) return;
  echo "second";
?>
<h3>Текст не обязательно должен выводиться оператором echo</h3><br>
```

В любой момент, можно узнать, какие файлы включает скрипт, воспользовавшись функцией `get_included_files()`, которая не принимает параметров и возвращает массив с именами включаемых файлов (листинг 5.81).

ЗАМЕЧАНИЕ

Массивы более подробно рассматриваются в *главе 6*.

Листинг 5.81. Список включаемых файлов

```
<?php
  // Включаем файл
  include "second.php";
  // Получаем массив включенных файлов
  $arr = get_included_files();
  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 5.81 будет следующий дамп массива:

```
Array
(
    [0] => D:\main\index.php
    [1] => D:\main\second.php
)
```

Первый элемент массива содержит абсолютный путь к главному файлу index.php, тогда как второй содержит путь к включаемому файлу second.php.

5.14. Подавление вывода ошибок. Оператор @

Язык программирования PHP содержит гибкую систему вывода ошибок, замечаний и предупреждений *(см. разд. 3.8)*. Однако вывод сообщений об ошибках и предупреждения не всегда являются желательными, особенно, когда их незапланированный

вывод (например, временное отсутствие связи с удаленным сервером) может нарушить дизайн Web-приложения. В этом случае прибегают к специальному оператору `@`, который будучи расположенным перед тем или иным выражением подавляет вывод ошибок и предупреждений в окно браузера.

В листинге 5.82 приводится инициализация переменной при помощи условного оператора *(см. разд. 5.8)*, если переменная `$x` имеет значение, отличное от 0, оно остается неизменным, в противном случае ей присваивается значение 1.

Листинг 5.82. Инициализация переменной `$x`

```
<?php
  $x = $x ?: 1;
  echo $x;
?>
```

Если директива `error_reporting` в конфигурационном файле php.ini имеет значение `E_ALL`, то результат выполнения скрипта из листинга 5.82 может выглядеть так, как это представлено на рис. 5.3.

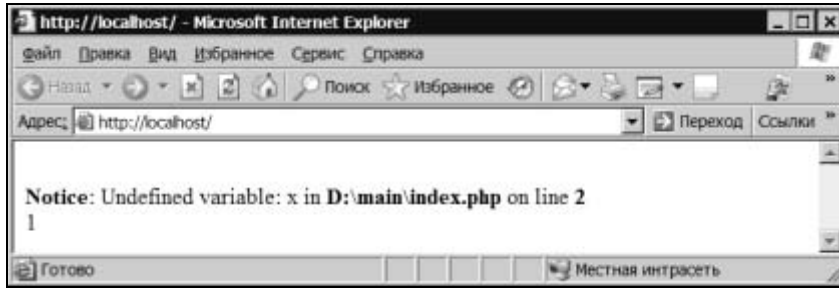


Рис. 5.3. Вывод замечания Notice из-за неинициализированной переменной `$x`

Для того чтобы подавить вывод замечания, не изменяя значения директивы `error_reporting`, достаточно разместить перед первым параметром условного оператора символ `@` (листинг 5.83).

Листинг 5.83. Подавление вывода замечания Notice

```
<?php
  $x = @$x ?: 1;
  echo $x;
?>
```

На рис. 5.4 приводится результат выполнения скрипта из листинга 5.83 (директива `error_reporting` имеет значение `E_ALL`).

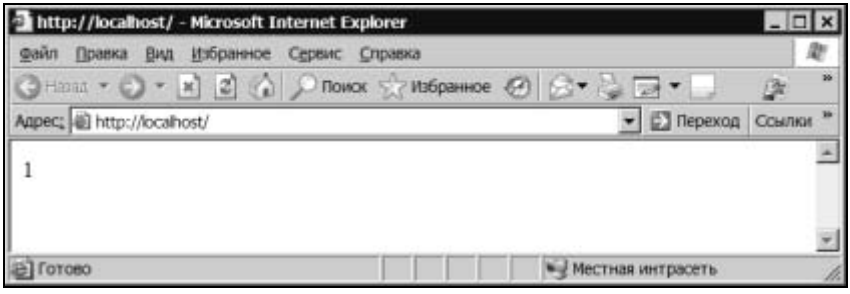


Рис. 5.4. Подавление вывода предупреждения

5.15. Приоритет выполнения операторов

В заключение приведем таблицу приоритета выполнения операторов (табл. 5.13). Операторы с более низким приоритетом расположены в таблице ниже, с более высоким — выше. Операторы, расположенные на одной строке, имеют одинаковый приоритет, и выполняется в первую очередь тот из операторов, который встречается в выражении первым.

ЗАМЕЧАНИЕ

В данной главе были рассмотрены не все операторы, часть из них уместнее рассмотреть совместно с классами, массивами и функциями.

Таблица 5.13. *Приоритет выполнения операторов*

Оператор	Ассоциативность
`new`	Неассоциативная
`[]`	Левая
`++ --`	Неассоциативная
`~ (int) (float) (string) (array) (object) @`	Неассоциативная
`instanceof`	Неассоциативная
`!`	Правая
`* / %`	Левая
`+ - .`	Левая
`<< >>`	Левая
`< <= > >=`	Неассоциативная
`== != === !==`	Неассоциативная
`&`	Левая
`^`	Левая

Таблица 5.13 *(окончание)*

Оператор	Ассоциативность
`|`	Левая
`&&`	Левая
`||`	Левая
`? :`	Левая
`= += -= *= /= .= %= &= |= ^= <<= >>=`	Правая
`and`	Левая
`xor`	Левая
`or`	Левая
`,`	Левая

Левая ассоциативность подразумевает, что выражение вычисляется слева направо, правая ассоциативность, соответственно, подразумевает противоположный порядок (листинг 5.84).

> ***Замечание***
>
> Несмотря на то, что оператор `!` имеет более высокий приоритет, чем `=`, PHP в выражении `if (!$a = foo())` присваивает переменной `$a` результат выполнения функции `foo()` и лишь затем применяет оператор `!`.

Листинг 5.84. Ассоциативность операторов

```
<?php
  $a = 3 * 3 % 5; // (3 * 3) % 5 = 4
  $a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

  $a = 1;
  $b = 2;
  $a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```



Глава 6

Массивы

Массивы являются одной из основных и часто встречающихся структур для хранения данных. По определению, массив представляет собой индексированную совокупность переменных одного типа. Каждая переменная или элемент массива имеет свой *индекс*, т. е. все элементы массива последовательно пронумерованы от 0 до $N-1$, где N — *размер массива*. Имена массивов, так же как и переменных, начинаются с символа `$`. Для того чтобы обратиться к отдельному элементу массива, необходимо поместить после его имени квадратные скобки, в которых указать индекс элемента массива. Так, обращение к `$arr[0]` запрашивает первый элемент массива `$str`, `$arr[1]` — второй и т. д. В качестве элементов массива могут выступать другие массивы, в этом случае говорят о многомерных массивах, а для обращения к конечным элементам используют несколько пар квадратных скобок: две — если массив двумерный `$arr[0][0]`, три — если массив трехмерный `$arr[0][1][0]`, и т. д.

6.1. Создание массива

Существует несколько способов создания массивов. Первый из них заключается в использовании конструкции `array()`, в круглых скобках которой через запятую перечисляются элементы массива. Конструкция `array()` в качестве результата возвращает массив. В листинге 6.1 создается массив `$arr`, состоящий из трех элементов, пронумерованных от 0 до 2.

Листинг 6.1. Создание массива при помощи конструкции `array()`

```
<?php
  $arr = array("Hello, ", "world", "!");
  echo $arr[0]; // Hello,
  echo $arr[1]; // world
  echo $arr[2]; // !
?>
```

В листинге 6.1 каждый элемент массива выводится при помощи отдельной конструкции `echo`. При попытке вывода всего массива `$arr` в окно браузера вместо его содержимого будет выведена строка "Array". Для просмотра структуры и содержимого массива предусмотрена специальная функция `print_r()`. В листинге 6.2 с ее помощью

выводится структура массива $arr. Для удобства восприятия вызов функции print_r() обрамляется HTML-тегами <pre> и </pre>, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Листинг 6.2. Вывод структуры массива при помощи функции print_r()

```
<?php
  $arr = array("Hello, ", "world", "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.2 будет следующий дамп массива $arr:

```
Array
(
    [0] => Hello,
    [1] => world
    [2] => !
)
```

Как видно из листингов 6.1 и 6.2, элементам массива автоматически назначены индексы, начиная с нулевого, как это принято в C-подобных языках программирования. Однако индекс начального элемента, а также порядок следования индексов можно изменять. Для этого в конструкции array() перед элементом указывается его индекс при помощи оператора =>. В листинге 6.3 первому элементу массива $arr назначается индекс 10, последующие элементы автоматически получают номера 11 и 12.

Листинг 6.3. Назначение индекса первому элементу массива

```
<?php
  $arr = array(10 => "Hello, ", "world", "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.3 будет следующий дамп массива $arr:

```
Array
(
    [10] => Hello,
    [11] => world
    [12] => !
)
```

Замечание

Следует отметить, что элементы с индексами 0, 1, ..., 9 и 13, ... просто не существуют, обращение к ним воспринимается как обращение к неинициализированной переменной, т. е. переменной, имеющей значение `NULL`.

Назначать индексы можно не только первому элементу, но и всем последующим элементам массива (листинг 6.4).

Листинг 6.4. Назначение индексов всем элементам массива

```
<?php
  $arr = array(10 => "Hello, ", 9 => "world", 8 => "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.4 будет следующий дамп массива `$arr`:

```
Array
(
    [10] => Hello,
    [9] => world
    [8] => !
)
```

Следует отметить, что все непронумерованные элементы будут автоматически получать значение, равное максимальному и увеличенному на единицу (листинг 6.5).

Листинг 6.5. Ненумерованные элементы получают максимальное значение

```
<?php
  $arr = array(10 => "Hello, ", 9 => "world", "!");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.5 будет следующий дамп массива `$arr`:

```
Array
(
    [10] => Hello,
    [9] => world
    [11] => !
)
```

Еще одним способом создания массива является присвоение неинициализированным элементам массива новых значений (листинг 6.6).

Листинг 6.6. Создание массива при помощи квадратных скобок

```
<?php
  $arr[10] = "Hello, ";
  $arr[11] = "world";
  $arr[12] = "!";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Скрипт из листинга 6.6 по результату выполнения полностью эквивалентен листингу 6.3. Допускается и автоматическое назначение индексов элементам, для этого значение индекса просто не указывается в квадратных скобках (листинг 6.7).

Листинг 6.7. Автоматическое назначение индекса

```
<?php
  $arr[] = "Hello, ";
  $arr[] = "world";
  $arr[] = "!";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Еще один способ создания массива заключается в приведении скалярной переменной (т. е. переменной типа `integer`, `float`, `string` или `boolean`) к типу `array` — результатом этого становится массив, содержащий один элемент с индексом 0 и значением, равным значению переменной.

Листинг 6.8. Приведение переменной к массиву

```
<?php
  $var = "Hello world";
  $arr = (array) $var;
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.8 будет следующий дамп:

```
Array
(
    [0] => Hello world
)
```

Помимо конструкции `array()`, существует еще несколько функций, которые позволяют создавать массивы. Их список приведен в табл. 6.1.

> ***Замечание***
>
> Следует отметить, что список функций, позволяющий создавать массив, в табл. 6.1 далеко не полный и не включает специализированных функций `explode()`, `preg_split()` и т. п., которые рассматриваются в главах, посвященных строковым функциям и регулярным выражениям.

Таблица 6.1. *Функции создания массивов*

Функция	Описание
`array([...])`	Создает массив из значений, переданных конструкции в качестве параметров
`array_fill($start_index, $num, $value)`	Возвращает массив, содержащий `$num` элементов, имеющих значение `$value`. Нумерация индексов при этом начинается со значения `$start_index`
`range($low, $high [, $step])`	Создает массив со значениями из интервала от `$low` до `$high` и шагом `$step`

Если элементы массива должны содержать одинаковые значения, для его создания удобно воспользоваться функцией `array_fill()`, которая имеет следующий синтаксис:

```
array_fill($start_index, $num, $value)
```

Функция возвращает массив, содержащий `$num` элементов, имеющих значение `$value`. Нумерация индексов при этом начинается со значения `$start_index` (листинг 6.9).

Листинг 6.9. Создание массива при помощи функции `array_fill()`

```
<?php
  // Создаем массив
  $arr = array_fill(5, 6, "Hello, world!");

  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.9 будут следующие строки:

```
Array
(
    [5] => Hello, world!
    [6] => Hello, world!
    [7] => Hello, world!
    [8] => Hello, world!
    [9] => Hello, world!
    [10] => Hello, world!
)
```

В качестве последнего способа создания массива рассмотрим функцию `range()`, которая позволяет создавать массивы для интервалов значений и имеет следующий синтаксис:

```
range($low, $high [, $step])
```

Параметр `$low` определяет начало интервала, а `$high` — его окончание. Необязательный параметр `$step` позволяет задать шаг. В листинге 6.10 демонстрируется создание массива из 6 элементов, которые принимают значения в интервале от 0 до 5.

Листинг 6.10. Использование функции `range()`

```
<?php
  $arr = range(0, 5);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.10 будут следующие строки:

```
Array
(
    [0] => 0
    [1] => 1
    [2] => 2
    [3] => 3
    [4] => 4
    [5] => 5
)
```

По умолчанию, массив заполняется значениями из интервала с шагом, равным единице, однако этот шаг можно изменить при помощи третьего параметра (листинг 6.11).

Замечание

В функции `range()` допускается использование отрицательного шага.

Листинг 6.11. Использование шага в функции `range()`

```
<?php
  $arr = range(0, 1, 0.2);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.11 будут следующие строки:

```
Array
(
    [0] => 0
    [1] => 0.2
    [2] => 0.4
    [3] => 0.6
    [4] => 0.8
    [5] => 1
)
```

6.2. Ассоциативные и индексные массивы

В качестве индексов массивов могут выступать не только числа, но и строки. В последнем массив называют *ассоциативным*, а индексы называют ключами. Если в качестве индексов массива выступают числа, он называется *индексным*. В листинге 6.12 приводится пример создания ассоциативного массива с двумя элементами, имеющими в качестве ключей строки "one" и "two".

> ***ЗАМЕЧАНИЕ***
>
> Один массив может иметь как числовые индексы, так и строковые ключи. В этом случае говорят о смешанном массиве.

Листинг 6.12. Создание ассоциативного массива

```
<?php
  $number = array("one" => "1", "two" => "2");
  echo "<pre>";
  print_r($number);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.12 будет следующий дамп массива `$arr`:

```
Array
(
  [one] => 1
  [two] => 2
)
```

При обращении к элементам ассоциативного массива вместо индексов указываются соответствующие ключи (листинг 6.13).

> ***Замечание***
>
> Ключи ассоциативного массива являются обычными строками, поэтому для обращения к элементам можно использовать как двойные, так и одиночные кавычки.

Листинг 6.13. Обращение к элементам ассоциативного массива

```
<?php
  $number = array("one" => "1", "two" => "2");
  echo $number["one"];  // 1
  echo "<br>";          // Перевод строки
  echo $number["two"];  // 2
?>
```

Помимо конструкции `array()` допускается создание элементов ассоциативного массива посредством прямого обращения к ним (листинг 6.14).

Листинг 6.14. Создание ассоциативного массива прямым обращением к элементам

```
<?php
  $number['one'] = "1";
  $number['two'] = "2";
  echo "<pre>";
  print_r($number);
  echo "</pre>";
?>
```

Если осуществляется попытка создания двух элементов массива с одинаковыми ключами, создается одни элемент с последним значением (листинг 6.15).

> ***Замечание***
>
> Ключи ассоциативных массивов, как и обычные строки, зависят от регистра. "two" и "Two" считаются разными ключами.

Листинг 6.15. Попытка создания двух элементов с одинаковыми ключами

```
<?php
  $number = array("one" => "1", "two" => "2", "two" => "3");
  echo $number["one"];   // 1
  echo "<br>";           // Перевод строки
  echo $number["two"];   // 3
?>
```

Ряд функций PHP предназначен для создания именно ассоциативных массивов (табл. 6.2).

***Таблица 6.2.** Создание ассоциативных массивов*

Функция	Описание
`array_combine($keys, $values)`	Создает ассоциативный массив с ключами из массива `$keys` и значениями из массива `$values`. Возвращает массив в случае успеха и `FALSE` в случае неудачи (например, если количество элементов в массивах `$keys` и `$values` не совпадают)
`compact($varname [, ...])`	Создает массив из существующих переменных. В качестве параметров, функция принимает названия переменных
`extract($var_array [, $extract_type [, $prefix]])`	Позволяет создать из ассоциативного массива `$var_array` PHP-переменные. Необязательный параметр `$extract_type` определяет поведение функции, если переменные уже существуют. Параметр `$prefix` позволяет задать префикс для новых переменных

Начиная с версии 5.0, в PHP введена функция `array_combine()`, которая позволяет создавать ассоциативный массив из двух других. Функция имеет следующий синтаксис:

```
array_combine($keys, $values)
```

Функция принимает в качестве параметров массив `$keys` с ключами нового результирующего массива и массив `$values` со значениями и возвращает ассоциативный массив в случае успеха или `FALSE`, если количество элементов в массивах `$keys` и `$values` не совпадает. В листинге 6.16 демонстрируется использование функции `array_combine()`.

Листинг 6.16. Использование функции `array_combine()`

```
<?php
  // Массив ключей
  $keys = array('green', 'red', 'yellow');
  // Массив значений
  $values = array('avocado', 'apple', 'banana');
```

```
  // Результирующий массив
  $result = array_combine($keys, $values);

  echo "<pre>";
  print_r($result);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.16 будет следующий дамп массива `$arr`:

```
Array
(
    [green] => avocado
    [red] => apple
    [yellow] => banana
)
```

Еще одной функцией, позволяющей создавать ассоциативные массивы, является функция `compact()`, которая имеет следующий синтаксис:

```
compact($varname [, ...] )
```

В качестве параметров функция принимает строки с названиями переменных. Результирующий массив состоит из элементов, ключи и значения которых соответствуют названиям и значениям переменных. В результирующий массив попадают лишь те элементы, для которых существует переменная (листинг 6.17).

> ***ЗАМЕЧАНИЕ***
>
> В качестве параметров функции `compact()` могут передаваться массивы, в этом случае функция рекурсивно разбирает их и обрабатывает каждый элемент массива как отдельный параметр.

Листинг 6.17. Использование функции `compact()`

```
<?php
  $city  = "San Francisco";
  $state = "CA";
  $event = "SIGGRAPH";

  $location_vars = array("city", "state");

  $arr = compact("event", "nothing_here", $location_vars);

  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.16 будет следующий дамп массива `$arr`:

```
Array
(
    [event] => SIGGRAPH
    [city] => San Francisco
    [state] => CA
)
```

Для функции `compact()` существует обратная функция `extract()`, которая преобразует ассоциативный массив в переменные PHP и имеет следующий синтаксис:

```
extract($var_array [, $extract_type [, $prefix]])
```

Функция создает из ассоциативного массива `$var_array` PHP-переменные. Необязательный параметр `$extract_type` определяет поведение функции, если переменные уже существуют, и может принимать значения из табл. 6.3. Параметр `$prefix` позволяет задать префикс для новых переменных.

Таблица 6.3. *Константы, определяющие поведение функции* `extract()`

Константа	Описание
`EXTR_OVERWRITE`	Если переменная с таким именем существует, она будет переопределена
`EXTR_SKIP`	Если переменная с таким именем существует, ее значение не переопределяется
`EXTR_PREFIX_SAME`	Если переменная с таким именем существует, к ее имени будет добавлен префикс `$prefix`
`EXTR_PREFIX_ALL`	Префикс `$prefix` будет добавлен ко всем именам переменных
`EXTR_PREFIX_INVALID`	Префикс `$prefix` добавляется только к переменным с недопустимыми именами (числовыми)
`EXTR_IF_EXISTS`	Переопределяются только уже существующие переменные, в противном случае никаких действий не предпринимается
`EXTR_PREFIX_IF_EXISTS`	Создать переменные к именам, которым будет добавлен префикс `$prefix`, только для уже существующих переменных
`EXTR_REFS`	Создать переменные как ссылки

В листинге 6.18 приводится пример использования функции `extract()`.

Листинг 6.18. Использование функции `extract()`

```
<?php
  $arr = array("event" => "SIGGRAPH",
               "city" => "San Francisco",
```

```
                "state" => "CA");

  extract($arr);

  echo $city."<br>";  // San Francisco
  echo $state."<br>"; // CA
  echo $event."<br>"; // SIGGRAPH
?>
```

6.3. Многомерные массивы

В качестве элементов массива могут выступать другие массивы, в этом случае говорят о *многомерных массивах*. Принцип создания многомерных массивов аналогичен созданию одномерных. Массивы можно создавать, обращаясь к элементам или используя вложенные конструкции `array()`. В листинге 6.19 демонстрируется создание многомерного ассоциативного массива `$ship`.

Листинг 6.19. Создание многомерного массива

```
<?php
  $ship = array(
       "Пассажирские корабли" => array("Лайнер", "Яхта", "Паром"),
       "Военные корабли" => array("Авианосец", "Линкор", "Эсминец"),
       "Грузовые корабли" => array("Сормовский", "Волго-Дон", "Окский")
        );
  echo "<pre>";
  print_r($ship);
  echo "</pre>";
?>
```

В результате такой инициализации будет создан массив следующей структуры:

```
Array
(
  [Пассажирские корабли] => Array
   (
      [0] => Лайнер
      [1] => Яхта
      [2] => Паром
    )
  [Военные корабли] => Array
   (
      [0] => Авианосец
      [1] => Линкор
```

```
      [2] => Эсминец
    )
  [Грузовые корабли] => Array
   (
      [0] => Сормовский
      [1] => Волго-Дон
      [2] => Окский
    )
)
```

В этом примере создан двумерный массив с размерами 3×3, т. е. получилось три массива, каждый из которых содержит в себе по три элемента. Следует отметить, что полученный массив является смешанным, т. е. в нем присутствуют как индексы, так и ключи ассоциативного массива — обращение к элементу `$ship['Пассажирские корабли'][0]` возвратит значение "`Лайнер`".

6.4. Интерполяция элементов массива в строки

Интересно подробнее остановиться на интерполяции элементов массива в строки. Вместо переменных в строках с двойными кавычками подставляется их значение *(см. разд. 3.6)*. Точно так же ведут себя элементы массива (листинг 6.20).

Листинг 6.20. Интерполяция элемента массива в строку

```
<?php
  $arr[0] = 14;
  echo "Событие произошло $arr[0] дней назад";
  // Событие произошло 14 дней назад
?>
```

Следует отметить, если речь идет об ассоциативных массивах, то кавычки, которыми обрамляется ключ, указывать не следует (листинг 6.21), в противном случае скрипт вернет ошибку разбора.

Листинг 6.21. Интерполяция элемента ассоциативного массива

```
<?php
  $arr['one'] = 14;
  echo "Событие произошло $arr[one] дней назад";
  // Событие произошло 14 дней назад
?>
```

Однако в случае многомерных массивов интерполировать элемент так, как это продемонстрировано в листинге 6.21, уже не получится. Для интерполяции потребуется либо

заключить элемент в фигурные кавычки, либо использовать оператор "точка" (листинг 6.22).

Листинг 6.22. Интерполяция элемента многомерного массива

```
<?php
  $arr[0][0] = 14;
  echo "Событие произошло ".$arr[0][0]." дней назад<br>";
  // Событие произошло 14 дней назад
  echo "Событие произошло {$arr[0][0]} дней назад<br>";
  // Событие произошло 14 дней назад
?>
```

Следует отметить, что при использовании фигурных скобок для ключей элементов ассоциативных массивов можно указывать кавычки (листинг 6.23).

Листинг 6.23. Использование фигурных скобок

```
<?php
  $arr['one'] = 14;
  echo "Событие произошло {$arr['one']} дней назад";
  // Событие произошло 14 дней назад
?>
```

6.5. Конструкция *list()*

Если конструкция `array()` позволяет создавать массивы, то конструкция `list()` решает обратную задачу — преобразует элементы массива в обычные переменные (листинг 6.24).

Листинг 6.24. Использование конструкции `list()`

```
<?php
  $arr = array(1, 2, 3);
  list($one, $two, $three) = $arr;
  echo $one;    // 1
  echo $two;    // 2
  echo $three;  // 3
?>
```

Следует отметить, что конструкция `list()` работает только с числовыми массивами, нумерация индексов которых начинается с нуля. В листинге 6.25 осуществляется попытка преобразования элементов ассоциативного массива в переменные `$one`, `$two` и `$three`, которые, однако, остаются неинициализированными.

Листинг 6.25. Конструкция `list()` не работает с ассоциативными массивами

```
<?php
  $arr = array('one' => 1, 'two' => 2, 'three' => 3);
  list($one, $two, $three) = $arr;
  echo $one;   // Notice: Undefined offset
  echo $two;   // Notice: Undefined offset
  echo $three; // Notice: Undefined offset
?>
```

Еще одной интересной особенностью конструкции `list()` является тот факт, что количество элементов в разбираемом массиве и в круглых скобках конструкции `list()` может не совпадать. Если элементов в массиве больше чем нужно, лишние просто будут отброшены, если меньше — часть переменных останется неинициализированной. Более того, часть первых элементов массива может быть пропущена, для этого достаточно указать соответствующее количество запятых (листинг 6.26).

Листинг 6.26. Количество элементов в массиве и конструкции `list()` может не совпадать

```
<?php
  $arr = array(1, 2, 3);
  list(, $two) = $arr;
  echo $two; // 2
?>
```

`list()` — достаточно любопытная конструкция, которая может применяться для реализации различных приемов, например, для реализации обмена значений переменных без переменной-посредника (листинг 6.27).

Листинг 6.27. Обмен значений двух переменных

```
<?php
  $x = 4;
  $y = 5;

  echo "до:<br>";
  echo "x = $x<br>"; // 4
  echo "y = $y<br>"; // 5

  list($y, $x) = array($x, $y);

  echo "после:<br>";
```

```
  echo "x = $x<br>"; // 5
  echo "y = $y<br>"; // 4
?>
```

В листинге 6.28 приводится аналогичный скрипт (применимый только для числовых переменных), решающий данную задачу без участия конструкций `list()` и `array()`.

Листинг 6.28. Альтернативное решение

```
<?php
  $x = 4;
  $y = 5;

  echo "до:<br>";
  echo "x = $x<br>";
  echo "y = $y<br>";

  $x = $x + $y;
  $y = $x - $y;
  $x = $x - $y;

  echo "после:<br>";
  echo "x = $x<br>";
  echo "y = $y<br>";
?>
```

6.6. Обход массива

При работе с массивами часто возникающей задачей является обход их элементов в цикле. В случае индексных массивов для их обхода можно воспользоваться операторами `for` или `while` *(см. разд. 5.10—5.12)*, для ассоциативных массивов предназначен специализированный оператор `foreach` *(см. разд. 6.7)*. В листинге 6.29 демонстрируется обход индексного массива при помощи цикла `for`.

Листинг 6.29. Обход массива в цикле `for`

```
<?php
  $number = array("1", "2", "3");
  for ($i = 0; $i < count($number); $i++)
  {
    echo $number[$i];
  }
?>
```

Результатом работы скрипта из листинга 6.29 будет строка: `123`.

Для подсчета количества элементов в массиве используется функция `count()`, которая принимает в качестве единственного параметра массив и возвращает количество элементов в нем. Так в листинге 6.29 для массива `$number` будет возвращено значение 3.

Для обхода массивов предусмотрено несколько специализированных функций, список которых представлен в табл. 6.4.

Таблица 6.4. *Функции для обхода массивов*

Функция	Описание
`current($arr)`	Возвращает текущее значение массива `$arr`, на которое указывает курсор
`pos($arr)`	Синоним для функции `current()`
`each($arr)`	Возвращает текущее значение массива `$arr`, на которое указывает курсор, и переводит его на следующую позицию
`end($arr)`	Переносит курсор в конец массива `$arr`
`key($arr)`	Возвращает ключ текущего элемента массива `$arr`
`next($arr)`	Переводит курсор на следующую позицию массива `$arr`
`prev($arr)`	Переводит курсор на предыдущую позицию массива `$arr`
`reset($arr)`	Устанавливает курсор на начало массива `$arr`
`list($varname, ...)`	Преобразует массив в переменные. Функция подробнее обсуждается в *разд. 6.5*

Если возникает необходимость обхода ассоциативного массива, используется цикл `while` совместно с конструкциями `list()` и `each()`. Функция `each()` возвращает пару "индекс-значение" текущего элемента массива и сдвигает курсор (указатель) массива на следующий элемент. Таким образом, повторный вызов функции возвращает следующий элемент массива, еще один вызов — следующий и так до тех пор, пока не будут извлечены все элементы массива.

Функция `each()` имеет следующий синтаксис:

```
each($arr)
```

Функция принимает в качестве единственного аргумента массив `$arr` и возвращает в качестве результата массив из четырех элементов:

```
[1] => "значение"
[value] => "значение"
[0] => индекс
[key] => индекс
```

В листинге 6.30 приводится пример использования функции `each()`.

Листинг 6.30. Применение функция `each()`

```
<?php
  $name = array("sacha ", "masha");

  echo "<pre>";
  $each_name = each($name);
  print_r($each_name);
  $each_name = each($name);
  print_r($each_name);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 6.30 представлен далее:

```
Array
(
    [1] => sacha
    [value] => sacha
    [0] => 0
    [key] => 0
)
Array
(
    [1] => masha
    [value] => masha
    [0] => 1
    [key] => 1
)
```

При каждом применении этой функции происходит сдвиг курсора массива на следующий элемент. Если курсор достиг конца массива, функция возвращает `FALSE`. Все это вместе делает ее идеальной для использования в цикле `while` (листинг 6.31).

> ***Замечание***
>
> Способ обход массива с помощью конструкции `list()` и функции `each()` считается устаревшим. Этот способ применялся при программировании в третьей версии PHP, т. к. в этой версии не было цикла `foreach`.

Листинг 6.31. Обход массива в цикле `while`

```
<?php
  $number = array("1", "2", "3");
  while(list($key, $val) = each($number))
```

```
  {
    echo $val;
  }
?>
```

Для перебора массивов в циклах можно использовать специализированные функции, работающие с курсором. Далее приводится их краткое описание.

Функция `current()` предназначена для определения значения текущего элемента массива и имеет следующий синтаксис:

```
current($arr)
```

Функция возвращает значение элемента, на который указывает в данный момент курсор. В том случае, если курсор оказался за пределами массива, или массив `$arr` состоит из пустых элементов, функция возвращает `FALSE`.

Функция `next()` производит перевод курсора вперед на одну позицию и имеет следующий синтаксис:

```
next($arr)
```

Функция перемещает курсор массива на следующий элемент, при этом возвращая значение элемента, на котором находился курсор до перемещения. Если элементов в массиве больше не осталось, функция возвращает `FALSE`. `FALSE` также возвращается в том случае, если курсору повстречается элемент с пустым значением. Поэтому для работы с массивами, содержащими пустые элементы, лучше использовать функцию `each()`.

Функция `prev()` производит перевод курсора на одну позицию назад. В остальном функция работает аналогично функции `next()`. Функция имеет следующий синтаксис:

```
prev($arr)
```

Функция `reset()` устанавливает курсор на первый элемент массива и возвращает его значение. Функция имеет следующий синтаксис:

```
reset($arr)
```

В листинге 6.32 приводится пример обхода массива при помощи специализированных функций и цикла `for`.

Листинг 6.32. Обход массива при помощи специализированных функций

```
<?php
  $arr = array(1, 2, 3, 4, 5);
  for (reset($arr); $value = current($arr); next($arr))
  {
     echo "$value<br>";
  }
?>
```

В результате работы скрипта из листинга 6.32 будет выведена последовательность чисел от 1 до 5.

Функция `end()` выполняет действие, обратное функции `reset()` — устанавливает курсор в конец массива и имеет следующий синтаксис:

```
end($arr)
```

В листинге 6.33 массив `$arr` выводится в обратном порядке.

Листинг 6.33. Вывод содержимого массива в обратном порядке

```
<?php
  $arr = array(1, 2, 3, 4, 5);
  for (end($arr); $value = current($arr); prev($arr))
  {
     echo "$value<br>";
  }
?>
```

6.7. Цикл *foreach*

Обход массива в цикле можно организовать при помощи цикла `foreach`, который специально создан для ассоциативных массивов и имеет следующий синтаксис:

```
foreach($array as [$key =>] $value)
{
    операторы;
}
```

Цикл последовательно обходит элементы массива `$array` с первого до последнего, помещая на каждой итерации цикла ключ в переменную `$key`, а значение в переменную `$value`. Имена этих переменных могут быть любыми (листинг 6.34).

> ***Замечание***
>
> В качестве первого аргумента конструкции `foreach` обязательно должен выступать массив, иначе конструкция выводит предупреждение.

Листинг 6.34. Обход массива в цикле `foreach`

```
<?php
  $number = array("first" => "1",
                  "second" => "2",
                  "third" => "3");
  foreach($number as $index => $val) echo "$index = $val <br>";
?>
```

Результатом работы скрипта из листинга 6.34 будут следующие строки:

```
first = 1
second = 2
third = 3
```

Переменная `$index` для ключа массива необязательна и может быть опущена (листинг 6.35).

Листинг 6.35. Вариант обхода массива в цикле `foreach`

```
<?php
  $number = array("first" => "1",
                  "second" => "2",
                  "third" => "3");
  foreach($number as $val)
  {
    echo $val; // выведет 123
  }
?>
```

Обход многомерных массивов проводится с помощью вложенных циклов `foreach`, при этом число вложенных циклов соответствует размерности массива (листинг 6.36).

Листинг 6.36. Обход многомерных массивов в цикле

```
<?php
  $ship = array(
       "Пассажирские корабли" => array("Лайнер","Яхта","Паром"),
       "Военные корабли" => array("Авианосец","Линкор","Эсминец"),
       "Грузовые корабли" => array("Сормовский","Волго-Дон","Окский")
        );
  foreach($ship as $key => $type)
  {
    // Вывод значений основных массивов
    echo "<b>$key</b><br>";
    foreach($type as $ship)
    {
      // Вывод значений для каждого из массивов
      echo "<li>$ship</li><br>";
    }
  }
?>
```

Результат выполнения скрипта из листинга 6.36 представлен на рис. 6.1.

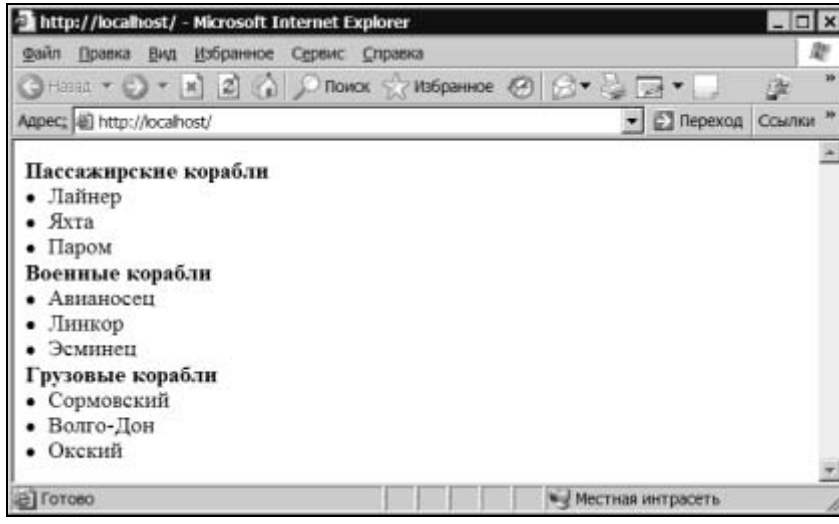


Рис. 6.1. Вывод многомерного массива

6.8. Сечения массива

Сечением называют часть массива, для их получения используются специализированные функции `array_slice()` и `array_splice()` (табл. 6.5).

***Таблица 6.5.** Функции для работы с сечениями массивов*

Функция	Описание
`array_slice($arr, $offset [, $length [, $preserve_keys]] )`	Возвращает часть массива `$arr`, начиная с элемента со смещением (номером) `$offset` от начала и длиной `$length` (или до конца массива, если аргумент `$length` не задан). Если `$offset` > 0, возвращаются элементы от начала массива, а если `$offset` < 0, то отсчет производится от конца массива. Когда значение аргумента `$length` положительно, то это число возвращаемых в массиве элементов, если же `$length` отрицательное, то это позиция последнего возвращаемого элемента в массиве `$array` от его конца. Если необязательный параметр `$preserve_keys` принимает значение `TRUE`, ключи сохраняются, в противном случае элементы индексируются по новой, начиная с 0
`array_splice($arr, $offset [, $length [, $replacement]] )`	Удаляет часть массива или заменяет ее частью другого массива

Функция `array_slice()` имеет следующий синтаксис:

```
array_slice($arr, $offset [, $length [, $preserve_keys]])
```

Функция возвращает часть массива $arr, начиная с элемента со смещением (номером) $offset от начала и длиной $length (или до конца массива, если аргумент $length не задан). Если $offset > 0, возвращаются элементы от начала массива, а если $offset < 0, то отсчет производится от конца массива. Когда значение аргумента $length положительно, то это число возвращаемых в массиве элементов, если же $length отрицательное, то это позиция последнего возвращаемого элемента в массиве $arr от его конца (листинг 6.37). Если необязательный параметр $preserve_keys принимает значение TRUE, ключи сохраняются, в противном случае элементы индексируются по новой, начиная с 0.

Листинг 6.37. Использование функции array_slice()

```
<?php
  $arr = array("a", "b", "c", "d", "e");
  $out = array_slice($arr, 2);     // возвращает "c", "d", "e"
  $out = array_slice($arr, 2, -1); // возвращает "c", "d"
  $out = array_slice($arr, -2, 1); // возвращает "d"
  $out = array_slice($arr, 0, 3);  // возвращает "a", "b", "c"
?>
```

Функция array_splice() удаляет часть массива или заменяет ее частью другого массива и имеет следующий синтаксис:

```
array_splice($arr, $offset [, $length [, $replacement]])
```

Удаляемые из массива $arr элементы могут быть заменены теми, что находятся в массиве $replacement (если аргумент $replacement не указан, то элементы просто удаляются). Значения аргументов $offset и $length точно такие же, как и для функции array_slice(). В листинге 6.38 демонстрируется пример с использованием функции array_splice().

Листинг 6.38. Использование функции array_splice()

```
<?php
  $arr = array("красный", "зеленый", "синий", "желтый");
  array_splice($arr, 2); // $arr == array("красный", "зеленый")

  $arr = array("красный", "зеленый", "синий", "желтый");
  array_splice($arr, 1, -1); // $arr == array("красный", "желтый")

  $arr = array("красный", "зеленый", "синий", "желтый");
  array_splice($arr, 1, count($arr), "оранжевый");
  // $arr == array("красный", "оранжевый")
?>
```

6.9. Слияние массивов

Оператор "плюс" (+), позволяющий складывать значения двух переменных, применим и для массивов (листинг 6.39).

> ***Замечание***
>
> Наряду с оператором "плюс" (+), допускается использование оператора +=.

Листинг 6.39. Слияние массивов при помощи оператора +

```
<?php
  $fst = array(1 => "one", 2 => "two");
  $snd = array(3 => "three", 4 => "four");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.39 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [1] => one
    [2] => two
    [3] => three
    [4] => four
)
```

Если в обоих складываемых массивах присутствуют элементы с одинаковыми индексами, в результирующий массив попадают элементы из левого массива (листинг 6.40).

Листинг 6.40. В результирующий массив попадает только один из элементов с одинаковыми индексами

```
<?php
  $fst = array("one", "two");
  $snd = array("three", "four", "five");
  $sum = $fst + $snd;
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.40 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [0] => one
    [1] => two
    [2] => five
)
```

Для того чтобы в результирующий массив попали элементы обоих массивов, вместо оператора + используют специальные функции (табл. 6.6).

***Таблица 6.6.** Функции для слияния массивов*

Функция	Описание
`array_merge($array1 [, $array2 [, ...]])`	Сливает массивы `$array1`, `$array2`, ... в один и возвращает его в качестве значения
`array_merge_recursive($array1 [,$array2, [, ...]])`	Рекурсивно сливает массивы `$array1`, `$array2`, ... в один и возвращает его в качестве значения

В листинге 6.41 приводится пример использования функции `array_merge()`.

> ***Замечание***
>
> Функция `array_merge()` может принимать в качестве параметров более чем два массива.

Листинг 6.41. Использование функции `array_merge()`

```
<?php
  $fst = array("one", "two");
  $snd = array("three", "four", "five");
  $sum = array_merge($fst, $snd);
  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.41 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [0] => one
```

```
    [1] => two
    [2] => three
    [3] => four
    [4] => five
)
```

В арсенале PHP имеется еще одна функция `array_merge_recursive()`, которая ведет себя аналогично `array_merge()` в отношении индексных массивов и как оператор `+` в отношении ассоциативных массивов (листинг 6.42).

Листинг 6.42. Использование функции `array_merge_recursive()`

```
<?php
  $ar1 = array("color" => array("favorite" => "red"), 5);
  $ar2 = array(10, "color" => array("favorite" => "green", "blue"));
  $sum = array_merge_recursive($ar1, $ar2);

  echo "<pre>";
  print_r($sum);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.42 в окно браузера будет выведен следующий дамп нового массива `$sum`:

```
Array
(
    [color] => Array
       (
            [favorite] => Array
               (
                    [0] => red
                    [1] => green
                )

            [0] => blue
        )

    [0] => 5
    [1] => 10
)
```

6.10. Сравнение массивов

Как и любые две переменные, массивы можно сравнивать при помощи операторов равенства == и неравенства != (листинг 6.43), которые более подробно рассматриваются в *разд. 5.5*. Два массива считаются равными, если количество их элементов, значения элементов и значения ключей совпадают.

> ***ЗАМЕЧАНИЕ***
>
> Оператор неравенства != имеет синоним <>.

Листинг 6.43. Сравнение массивов при помощи оператора равенства ==

```
<?php
  $ar1 = array(1, 2, 3, 4, 5);
  $ar2 = array(1, 2, 3, 4, 5);
  $ar3 = array(1, 2, 3, 4);
  $ar4 = array(1, 2, 6, 4, 5);

  if ($ar1 == $ar2) echo "Массивы равны<br>";
  else echo "Массивы не равны<br>";

  if ($ar1 == $ar3) echo "Массивы равны<br>";
  else echo "Массивы не равны<br>";

  if ($ar1 == $ar4) echo "Массивы равны<br>";
  else echo "Массивы не равны<br>";
?>
```

В качестве результата скрипт из листинга 6.43 выводит следующие строки:

```
Массивы равны
Массивы не равны
Массивы не равны
```

Операторы эквивалентности === и неэквивалентности !== соответственно схожи с операторами равенства == и неравенства !=. Два массива считаются эквивалентными, если совпадают не только количества их элементов, ключи и значения, но имеется и совпадение типов последних (листинг 6.44).

Листинг 6.44. Сравнение массивов при помощи оператора эквивалентности ===

```
<?php
  $fst = array(1 => 1, 2 => 2);
  $snd = array(1 => 1, 2 => "2");
```

```
  if ($fst == $snd) echo "Массивы равны<br>";
  else echo "Массивы не равны<br>";

  if ($fst === $snd) echo "Массивы эквивалентны<br>";
  else echo "Массивы не эквивалентны<br>";
?>
```

В качестве результата скрипт из листинга 6.44 выводит следующие строки:

```
Массивы равны
Массивы не эквивалентны
```

Интересно отметить, что массивы, в которых используются ключи разного типа, не считаются неэквивалентными (листинг 6.45).

Листинг 6.45. Оператор эквивалентности не учитывает тип ключей

```
<?php
  $fst = array(1 => 1, 2 => 2);
  $snd = array(1 => 1, "2" => 2);

  if ($fst == $snd) echo "Массивы равны<br>";
  else echo "Массивы не равны<br>";

  if ($fst === $snd) echo "Массивы эквивалентны<br>";
  else echo "Массивы не эквивалентны<br>";
?>
```

В качестве результата скрипт из листинга 6.45 выводит следующие строки:

```
Массивы равны
Массивы эквивалентны
```

PHP предоставляет широкий набор функций для получения разницы между двумя или большим количеством массивов. Функции класса `array_diff()` возвращают массив, содержащий элементы, которые не входят хотя бы в один из сравниваемых массивов, функции класса `array_intersect()` возвращают массивы, содержащие элементы, входящие во все сравниваемые массивы. Полный список функции приводится в табл. 6.7.

Таблица 6.7. *Функции сравнения массивов*

Функция	Описание
`array_diff($array1, $array2 [, ...])`	Возвращает массив со значениями, которые отсутствуют хотя бы в одном из массивов *$array1*, *$array2*, ...

***Таблица 6.7** (продолжение)*

Функция	Описание
`array_diff_assoc($array1, $array2 [, ...])`	Возвращает массив с парами "ключ/значение", которые отсутствуют хотя бы в одном из массивов `$array1`, `$array2`, ... В отличие от функции `array_diff()` при сравнении учитывается и ключ элемента
`array_diff_key($array1, $array2 [, ...])`	Возвращает массив с ключами, которые отсутствуют хотя бы в одном из массивов `$array1`, `$array2`, ...
`array_diff_uassoc($array1, $array2 [, ..., $key_compare_func])`	Функция `array_diff_uassoc()` эквивалентна `array_diff_assoc()`, однако для сравнения ключей используется функция `$key_compare_func`, определенная пользователем
`array_diff_ukey($array1, $array2 [, ..., $key_compare_func])`	Функция `array_diff_ukey()` эквивалентна `array_diff_key()`, однако для сравнения ключей применяется пользовательская функция `$key_compare_func`
`array_udiff($array1, $array2 [, ..., $data_compare_func])`	Функция `array_udiff()` эквивалентна `array_diff()`, однако для сравнения значений применяется пользовательская функция `$data_compare_func`
`array_udiff_assoc($array1, $array2 [, ..., $data_compare_func])`	Функция `array_udiff_assoc()` эквивалентна функции `array_diff_assoc()`, однако для сравнения значений применяется пользовательская функция `$data_compare_func`
`array_udiff_uassoc($array1, $array2 [, ..., $data_compare_func, $key_compare_func])`	Функция `array_udiff_uassoc()` эквивалентна функции `array_diff_assoc()`, однако для сравнения значений применяется пользовательская функция `$data_compare_func`, а для ключей — `$key_compare_func`
`array_intersect ($array1, $array2 [, ...])`	Возвращает массив со значениями, которые присутствуют во всех массивах `$array1`, `$array2`, ...
`array_intersect_assoc($array1, $array2 [, ...])`	Возвращает массив с парами "ключ/значение", которые присутствуют во всех массивах `$array1`, `$array2`, ... В отличие от функции `array_intersect()` при сравнении учитывается и ключ элемента
`array_intersect_key($array1, $array2 [, ...])`	Возвращает массив с ключами, которые присутствуют во всех массивах `$array1`, `$array2`, ...
`array_intersect_uassoc($array1, $array2 [, ..., $key_compare_func])`	Функция `array_intersect_uassoc()` эквивалентна `array_intersect_assoc()`, однако для сравнения ключей используется функция `$key_compare_func`, определенная пользователем

Таблица 6.7 *(окончание)*

Функция	Описание
`array_intersect_ukey($array1, $array2 [, ..., $key_compare_func])`	Функция `array_intersect_ukey()` эквивалентна `array_intersect_key()`, однако для сравнения ключей используется функция `$key_compare_func`, определенная пользователем
`array_uintersect ($array1, $array2 [, ..., $data_compare_func])`	Функция `array_uintersect()` эквивалентна `array_intersect()`, однако для сравнения значений применяется пользовательская функция `$data_compare_func`
`array_uintersect_assoc($array1, $array2 [, ..., $data_compare_func])`	Функция `array_uintersect_assoc()` эквивалентна функции `array_intersect_assoc()`, однако для сравнения значений применяется пользовательская функция `$data_compare_func()`
`array_uintersect_uassoc($array1, $array2 [, ..., $data_compare_func, $key_compare_func])`	Функция `array_uintersect_uassoc()` эквивалентна функции `array_intersect_assoc()`, однако для сравнения элементов применяется пользовательская функция `$data_compare_func()`, а ключей — пользовательская функция `$key_compare_func`

Функция `array_diff()` определяет *исключительное* пересечение массивов и имеет следующий синтаксис:

```
array_diff($array1, $array2 [, ...] )
```

Функция возвращает массив, который содержит значения, имеющиеся только в массиве `$array1` и отсутствующие в других (`$array2` и т. д.). Листинг 6.46 содержит пример с использованием функции `array_diff()`.

Листинг 6.46. Определение исключительного пересечения массивов

```
<?php
  $arr1 = array("a" => "зеленый",
                "синий",
                "красный",
                "желтый");
  $arr2 = array("b" => "синий",
                "желтый",
                "красный");

  $result = array_diff($arr1, $arr2);

  echo "<pre>";
```

```
  print_r($result);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.46 будет следующий дамп:

```
Array
(
    [a] => зеленый
)
```

Функция `array_intersect()` определяет *включительное* пересечение массивов, т. е. она возвращает массив, который содержит значения массива `$array1`, имеющиеся во всех остальных массивах. Функция имеет следующий синтаксис:

```
array_intersect($array1, $array2 [, ...] )
```

В листинге 6.47 приводится пример с использованием функции `array_intersect()`.

Листинг 6.47. Определение включительного пересечения массивов

```
<?php
  $arr1 = array("a" => "зеленый", "синий", "красный", "желтый");
  $arr2 = array("b" => "синий", "желтый", "красный");
  $result = array_intersect($arr1, $arr2);
  echo "<pre>";
  print_r($result);
  echo "</pre>";
?>
```

В качестве результат работы скрипта из листинга 6.47 будет следующий дамп:

```
Array
(
  [0] => синий
  [1] => красный
  [2] => желтый
)
```

6.11. Проверка существования элементов массива

Проверить, существует тот или иной элемент массива, можно при помощи конструкции `isset()` (листинг 6.48).

Листинг 6.48. Проверка существования элементов массива при помощи `isset()`

```
<?php
  $arr = array(5 => 1, 2, 3);
```

```
  for ($i = 0; $i < 10; $i++)
  {
    if (isset($arr[$i])) echo "Элемент \$arr[$i] существует<br>";
    else echo "Элемент \$arr[$i] не существует<br>";
  }
?>
```

В качестве результата скрипт из листинга 6.48 выводит следующие строки:

```
Элемент $arr[0] не существует
Элемент $arr[1] не существует
Элемент $arr[2] не существует
Элемент $arr[3] не существует
Элемент $arr[4] не существует
Элемент $arr[5] существует
Элемент $arr[6] существует
Элемент $arr[7] существует
Элемент $arr[8] не существует
Элемент $arr[9] не существует
```

Разумеется, существование массива также может быть проверено при помощи конструкции `isset()`, если необходимо убедиться, является ли текущая переменная массивом, используют функцию `is_array()` (листинг 6.49).

Листинг 6.49. Использование функции `is_array()`

```
<?php
  $arr = array(1, 2, 3);

  if (is_array($arr)) echo "Это массив<br>";
  else echo "Это не массив<br>";

  if (is_array($arr[0])) echo "Это массив<br>";
  else echo "Это не массив<br>";
?>
```

В качестве результата скрипт из листинга 6.49 выводит следующие строки:

```
Это массив
Это не массив
```

Конструкция `isset()` осуществляет проверку существования элемента с заданным ключом. Однако зачастую требуется выяснить, входит ли в состав того или иного массива элемент с заданным ключом или значением. Для этого удобнее использовать специализированные функции (табл. 6.8).

***Таблица 6.8.** Проверка существования элементов*

Функция	Описание
in_array($*value*, $*arr* [, $*strict*])	Проверяет, существует ли значение $*value* в массиве $*arr*, и возвращает TRUE, если значение найдено, и FALSE в противном случае. Если необязательный параметр $strict принимает значение TRUE, то при сравнении значения элемента массива с $*search_value* будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==
array_key_exists($*key*, $*arr*)	Возвращает TRUE, если в массиве $arr присутствует ключ $key, в противном случае возвращается FALSE
array_search($*value*, $*arr* [,$*strict*])	Ищет значение $*value* в массиве $*arr* и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается FALSE. Если необязательный параметр $*strict* принимает значение TRUE, то при сравнении значения элемента массива с $*needle* будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==

Функция in_array() осуществляет поиск значения в массиве и имеет следующий синтаксис:

```
in_array($value, $arr [, $strict])
```

Функция ищет в массиве $*arr* значение $*value* и возвращает TRUE, если значение найдено, и FALSE в противном случае (листинг 6.50).

Листинг 6.50. Поиск элемента в массиве

```
<?php
  $number = array(0.57, '21.5', 40.52);
  if (in_array(21.5, $number)) echo "Значение 21.5 найдено";
  else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

```
Значение 21.5 найдено
```

Заметим, что найденный элемент массива взят в одинарные кавычки, т. е. относится к строковому типу, в отличие от других элементов этого же массива. В приведенном варианте использования функции это различие не фиксируется. Для того чтобы функция использовала для сравнения оператор эквивалентности === вместо оператора равенства ==, необходимо третий необязательный параметр $strict установить в значение TRUE (листинг 6.51).

Листинг 6.51. Поиск элемента в массиве с различием по типу

```
<?php
  $number = array(0.57, '21.5', 40.52);
  if (in_array(21.5, $number, true)) echo "Значение 21.5 найдено";
  else echo "Ничего не найдено";
?>
```

В результате будет выведена фраза:

```
Ничего не найдено
```

В этом случае ничего выведено не будет, т. к. тип элемента, передаваемого функции `in_array()` (`float`), отличается от типа элемента в массиве (`string`). При таком вызове функции, она найдет элемент 21.5 только, если он тоже будет взят в кавычки (т. е. тоже будет строкового типа):

```
if (in_array('21.5', $number, true))
```

Аналогично функции `in_array()` для поиска заданного ключа в массиве можно воспользоваться функцией `array_key_exists()`, которая имеет следующий синтаксис:

```
array_key_exists($key, $arr)
```

Функция возвращает `TRUE`, если ключ `$key` найден в массиве `$arr` (листинг 6.52).

Листинг 6.52. Поиск ключей массива

```
<?php
  $arr = array("first_numb" => 1, "second_numb" => 2);
  if (array_key_exists("first_numb", $arr) echo "OK";
?>
```

Найти ключ массива по значению позволяет функция `array_search()`, которая имеет следующий синтаксис:

```
array_search($value, $arr [, $strict])
```

Функция ищет значение `$value` в массиве `$arr` и, если значение найдено, возвращает соответствующий ключ, в противном случае возвращается `FALSE`. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$needle` будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==. Пример использования функции приводится в листинге 6.53.

Листинг 6.53. Использование функции `array_search()`

```
<?php
  $array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');

  $key = array_search('green', $array); // $key = 2;
  $key = array_search('red', $array);   // $key = 1;
?>
```

6.12. Строки как массивы

В С-подобных языках программирования, к которым относится и PHP, строки рассматриваются как массивы, элементами которых выступают символы. Точно так же как в числовых массивах, первый элемент помечается индексом 0, второй 1 и т. д. В листинге 6.54 приводится пример, в котором строка "PHP" выводится вертикально.

ЗАМЕЧАНИЕ

До версии PHP 6.0 вместо квадратных скобок `[]` для доступа к отдельным символам строки допускалось использование фигурных скобок `{}`, однако, начиная с PHP 6.0, такая возможность была аннулирована.

Листинг 6.54. Обращение к строке, как к символьному массиву

```
<?php
  $str = "PHP";
  echo $str[0]."<br>";
  echo $str[1]."<br>";
  echo $str[2]."<br>";
?>
```

Результат работы скрипта из листинга 6.54 представлен на рис. 6.2.

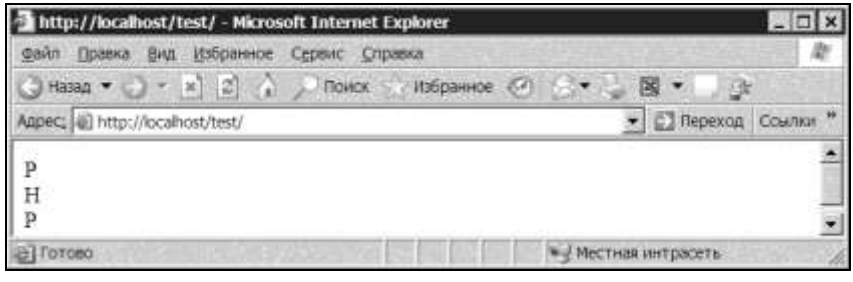


Рис. 6.2. Обращение к отдельным символам строки

По аналогии с сечениями, рассмотренными в *разд. 6.8*, если указать в квадратных скобках два значения, разделенных запятой, из строки можно извлекать подстроки. В листинге 6.55 из строки "Hello, world!" извлекается подстрока "Hello".

ЗАМЕЧАНИЕ

Возможность извлечения подстрок при помощи квадратных скобок появилась в PHP, только начиная с версии 6.0.

Листинг 6.55. Извлечение подстрок из строки

```
<?php
  $str = "Hello, world!";
  echo $str[0, 4]; // Hello
?>
```

6.13. Количество элементов в массиве

Функции данной группы осуществляют подсчет количества элементов в массиве (табл. 6.9).

***Таблица 6.9.** Количество элементов в массиве*

Функция	Описание
`count($arr [, $mode])`	Возвращает количество элементов массива `$arr`. Если необязательный параметр `$mode` принимает значение константы `COUNT_RECURSIVE`, функция рекурсивно обходит многомерный массив, в противном случае подсчитывается количество элементов только на текущем уровне
`sizeof()`	Синоним для функции `count()`
`array_count_values ($arr)`	Подсчитывает количество уникальных значений среди элементов массива. В результате возвращает ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значения — количество их вхождения в массив `$arr`

Для подсчета количества элементов в массиве предназначена функция `count()`, которая имеет следующий синтаксис:

```
count($arr [, $mode])
```

Возвращает количество элементов массива `$arr`. В листинге 6.56 демонстрируется пример с использованием этой функции.

Листинг 6.56. Использование функции `count()`

```
<?php
  $car = array("жигули",
               "волга",
               "запорожец",
               "ока");
  echo count($car); // 4
?>
```

Если в качестве аргумента функции передается многомерный массив, то по умолчанию она возвращает размерность текущего измерения (листинг 6.57).

Листинг 6.57. Использование `count()` совместно с многомерными массивами

```
<?php
  $arr = array(array(1, 2, 3, 4),
               array(5, 6, 7, 8));
  echo count($arr);    // 2
  echo count($arr[0]); // 4
?>
```

В первом случае возвращается число 2, т. к. в первом измерении только 2 элемента — это массивы `array(1, 2, 3, 4)` и `array(5, 6, 7, 8)`. Во втором случае в качестве аргумента передается уже массив `array(1, 2, 3, 4)`, в котором четыре элемента, о чем и сообщает функция `count()`.

Для того чтобы функция рекурсивно обходила многомерный массив, подсчитывая количество всех элементов, в том числе и во вложенных массивах, необязательному параметру `$mode` необходимо передать в качестве значения константу `COUNT_RECURSIVE` (листинг 6.58).

Листинг 6.58. Подсчет элементов многомерного массива

```
<?php
  $arr = array(array(1, 2, 3, 4),
               array(5, 6, 7, 8));
  echo count($arr, COUNT_RECURSIVE); // 10
?>
```

Еще одной интересной функцией является `array_count_values()`, которая подсчитывает количество уникальных значений среди элементов массива и имеет следующий синтаксис:

```
array_count_values($arr)
```

Она возвращает ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значения — количество их вхождения в массив `$arr` (листинг 6.59).

Замечание

Массив `$array` может иметь в качестве элементов лишь числа или строки, в противном случае генерируется предупреждение.

Листинг 6.59. Использование функции `array_count_values()`

```
<?php
  $array = array(1, "hello", 1, "world", "hello");
  $new = array_count_values($array);

  echo "<pre>";
  print_r($new);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.59 будет выведен следующий дамп массива `$new`:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

6.14. Сумма элементов массива

Функция `array_sum()` возвращает сумму всех числовых элементов массива и имеет следующий синтаксис:

```
array_sum($arr)
```

Тип возвращаемого числа определяется типом значений элементов массива. Пример с использованием функции `array_sum()` демонстрируется в листинге 6.60.

Листинг 6.60. Использование функции `array_sum()`

```
<?php
  $arr = array(1,2,3,4,5);
  $sum = array_sum($arr);
  echo $sum; // выводит 15
?>
```

6.15. Случайные элементы массива

Для получения случайного значения индексного массива можно использовать функцию `rand()`, которая возвращает случайное число из заданного диапазона. Функция `rand()` имеет следующий синтаксис:

```
rand([$min, $max]);
```

Необязательные параметры `$min` и `$max` задают соответственно начало и конец интервала, из которого выбирается случайное значение. При помощи функции `rand()` можно извлечь случайный индекс массива, указав в качестве начала интервала 0, а в качестве конца — количество элементов в массиве за вычетом единицы (листинг 6.61).

> ***Замечание***
>
> Функция `rand()` является математической функцией и более подробно рассматривается в *главе 12*.

Листинг 6.61. Получение случайного элемента массива

```
<?php
  // Объявляем массив
  $arr = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);

  // Получаем случайный индекс массива
  $index = rand(0, count($arr) - 1);

  // Выводим случайный элемент массива
  echo $arr[$index];
?>
```

В табл. 6.10 приведены дополнительные функции обработки случайного элемента массива.

Таблица 6.10. *Случайные элементы массива*

Функция	Описание
`array_rand($arr [, $num_req])`	Выбирает одно или несколько случайных значений из массива. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$arr`; если в параметре `$num_req` задается число большее 1, возвращается массив со случайным набором ключей массива `$arr`
`shuffle($arr)`	Перемешивает элементы массива `$arr` в случайном порядке

Подход, описанный в листинге 6.61, применим только для индексных массивов, индексы которых начинаются с 0 и не имеют перерывов в нумерации. Для всех остальных массивов, для получения случайного значения используется функция `array_rand()`, которая возвращает массив выбранных случайным образом индексов элементов массива `$arr`. Функция имеет следующий синтаксис:

```
array_rand($arr [, $num_req])
```

Функция выбирает случайное значение из массива `$arr`. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$arr`, если в параметре `$num_req` задается число большее 1, возвращается массив со случайным набором ключей массива `$arr` (листинг 6.62).

Листинг 6.62. Использование функции `array_rand()`

```
<?php
  // Инициируем массив
  $arr = array("синий", "желтый", "зеленый", "красный", "оранжевый");
  $rand_keys = array_rand($arr, 2);

  echo "<pre>";
  print_r($rand_keys);
  echo "</pre>";
?>
```

Еще одной полезной функцией для получения случайных значений является функция `shuffle()`, которая перемешивает элементы массива случайным образом и имеет следующий синтаксис:

```
shuffle($arr)
```

Листинг 6.63 демонстрирует работу функции.

Листинг 6.63. Функция `shuffle()`

```
<?php
  $arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  shuffle($arr);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат работы функции `shuffle()` может выглядеть следующим образом:

```
Array
(
    [0] => 7
    [1] => 9
    [2] => 8
    [3] => 1
    [4] => 3
    [5] => 10
    [6] => 2
    [7] => 4
    [8] => 5
    [9] => 6
)
```

6.16. Сортировка массивов

PHP предоставляет разработчику широкие возможности для сортировки массивов (табл. 6.11).

Таблица 6.11. *Функции для сортировки массивов*

Функция	Описание
`sort($arr [, $sort_flags])`	Сортирует массив `$arr` в прямом порядке. Параметр `$sort_flags` задает параметры сортировки: • `SORT_REGULAR` — обычная сортировка; • `SORT_NUMERIC` — сравнивать элементы как числа; • `SORT_STRING` — сравнивать элементы как строки; • `SORT_LOCALE_STRING` — сравнивать элементы как строки, основываясь на текущей локали
`rsort($arr [, $sort_flags])`	Сортирует массив `$arr` в обратном порядке
`asort($arr [, $sort_flags])`	Сортирует массив `$arr` в прямом порядке с сохранением отношения "ключ/значение"
`arsort($arr [, $sort_flags] )`	Сортирует массив `$arr` в обратном порядке с сохранением отношения "ключ/значение"
`natsort($arr)`	Осуществляет "естественную" сортировку массива `$arr`, учитывая регистр
`natcasesort($arr)`	Осуществляет "естественную" сортировку массива `$arr`, без учета регистра
`ksort($arr [, $sort_flags])`	Сортирует массив `$arr` в прямом порядке по ключам с сохранением отношения "ключ/значение"
`krsort($arr [, $sort_flags])`	Сортирует массив `$arr` в обратном порядке по ключам с сохранением отношения "ключ/значение"
`usort($arr, $cmp_function)`	Сортирует массив `$arr` с использованием пользовательской функции сравнения элементов `$cmp_function`
`uasort($arr, $cmp_function)`	Сортирует массив `$arr` с использованием пользовательской функции сравнения элементов `$cmp_function` и сохранением отношения "ключ/значение"
`uksort($arr, $cmp_function)`	Сортирует массив `$arr` по ключам с использованием пользовательской функции сравнения элементов `$cmp_function` и сохранением отношения "ключ/значение"
`array_multisort($ar1 [, $arg [, ... [, ...]]])`	Сортирует один многомерный массив или подвергает сортировке несколько массивов сразу

Функция `sort()` позволяет сортировать массив `$arr` по возрастанию и имеет следующий синтаксис:

```
sort($arr [, $sort_flags])
```

Необязательный аргумент `$sort_flags` указывает, как именно должны сортироваться элементы (задает флаги сортировки). Допустимы следующие значения этого аргумента:

- `SORT_REGULAR` — задает нормальное сравнение элементов, т. е. сравнивает элементы "как есть";
- `SORT_NUMERIC` — сравнивает элементы как числа;
- `SORT_STRING` — сравнивает элементы как строки;
- `SORT_LOCALE_STRING` — сравнивает элементы как строки, основываясь на текущей локали.

В листинге 6.64 приводится пример использования функции `sort()`.

Листинг 6.64. Сортировка массива по возрастанию

```
<?php
  $number = array("2", "1", "4", "3","5"); // исходный массив
  echo "до сортировки: <br>";
  for ($i = 0; $i < count($number); $i++)
  {
    echo "$number[$i]  ";
  }
  sort($number); // сортируем массив по возрастанию
  echo "<br> после сортировки: <br>";
  for ($i = 0; $i < count($number); $i++)
  {
    echo "$number[$i]  ";
  }
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
1 2 3 4 5
```

Сортировка строк происходит по старшинству первой буквы в алфавите. Такую сортировку часто называют сортировкой в алфавитном порядке. К примеру, если имеется массив

```
array("one",
      "two",
```

```
      "abs",
      "three",
      "uic",
      "for",
      "five")
```

то применение к нему функции `sort()` приведет к следующему результату:

```
[0] => abs
[1] => five
[2] => for
[3] => one
[4] => three
[5] => two
[6] => uic
```

Функция `rsort()` сортировки массива по убыванию имеет следующий синтаксис:

```
rsort($arr [, $sort_flags])
```

Во всем остальном ведет себя аналогично функции `sort()`. В листинге 6.65 приводится пример использования функции `rsort()`.

Листинг 6.65. Сортировка массива по убыванию

```
<?php
  $number = array("2", "1", "4", "3","5"); // исходный массив
  echo("до сортировки: <br>");
  for ($i = 0; $i < count($number); $i++)
  {
    echo "$number[$i]  ";
  }
  rsort($number); // сортируем массив по убыванию
  echo "<br> после сортировки: <br>";
  for ($i=0; $i < count($number); $i++)
  {
    echo "$number[$i]  ";
  }
?>
```

Результат:

```
до сортировки:
2 1 4 3 5
после сортировки:
5 4 3 2 1
```

Функция `asort()` осуществляет сортировку ассоциативного массива по возрастанию и имеет следующий синтаксис:

```
asort($arr [, $sort_flags])
```

Функция `asort()` сортирует массив `$arr` так, чтобы его значения шли в алфавитном (если это строки) или возрастающем (для чисел) порядке. Значения необязательного аргумента `$sort_flags` такие же, как и для функции `sort()`. Важное отличие этой функции от функции `sort()` состоит в том, что при применении функции `asort()` сохраняются связи между ключами и соответствующими им значениями (листинг 6.66), чего нет в функции `sort()` (там эти связи попросту разрываются).

Листинг 6.66. Сортировка функцией `assort()`

```
<?php
  $arr = array("a" => "one",
               "b" => "two",
               "c" => "three",
               "d" => "four");
  asort($arr);
  foreach($arr as $key => $val)
  {
    echo " $key => $val ";
  }
?>
```

Результат:

```
d => four a => one c => three b => two
```

Как видно, связи "ключ/значение" сохранились при сортировке. При сортировке массива при помощи функции `sort()` результат бы выглядел следующим образом:

```
0 => four 1 => one 2 => three 3 => two
```

Функция `arsort()` аналогична функции `asort()`, только она упорядочивает массив не по возрастанию, а по убыванию.

```
arsort($arr [, $sort_flags])
```

Функции `ksort()` осуществляет сортировку не по значениям, а по ключам массива в порядке их возрастания, при этом связи между ключами и значениями сохраняются. Функция имеет следующий синтаксис:

```
ksort($arr [, $sort_flags])
```

Значения аргумента `$sort_flags` такие же, как и функции `sort()`. В листинге 6.67 приводится пример использования функции `ksort()`.

Листинг 6.67. Сортировка по индексам массива

```
<?php
  $arr = array("a" => "one",
               "d" => "four",
               "c" => "three",
               "b" => "two");
  ksort($arr);
  foreach($arr as $key => $val)
  {
    echo(" $key => $val ");
  }
?>
```

Результат:

```
a => one b => two c => three d => four
```

Функция `krsort()` осуществляет сортировку элементов массива по ключам в обратном порядке (по убыванию). Во всем остальном аналогична функции `ksort()`. Функция имеет следующий синтаксис:

```
krsort($arr [, $sort_flags])
```

Функция `natsort()` выполняет "естественную" сортировку массива и имеет следующий синтаксис:

```
natsort($arr)
```

Под естественной сортировкой понимается сортировка элементов таким образом, как их отсортировал бы человек ("естественным образом"). Приведем пример. Пусть у нас есть несколько файлов с именами

```
file1.txt
file10.txt
file2.txt
file12.txt
file20.txt
```

При обычной сортировке файлы будут расположены в следующем ("машинном") порядке:

```
file1.txt
file10.txt
file12.txt
file2.txt
file20.txt
```

Естественная же сортировка приведет к следующему результату:

```
file1.txt
file2.txt
```

```
file10.txt
file12.txt
file20.txt
```

Производится естественная сортировка с помощью функции `natsort()`, принимающей в качестве параметра сортируемый массив (листинг 6.68).

Листинг 6.68. Естественная сортировка

```
<?php
  $arr_first = $arr_second = array("file12.txt",
                                   "file10.txt",
                                   "file2.txt",
                                   "file1.txt");
  sort($arr_first);
  echo "Обычная сортировка<br>";
  echo "<pre>";
  print_r($arr_first);
  echo "</pre>";

  natsort($arr_second);
  echo "<br>Естественная сортировка<br>";
  echo "<pre>";
  print_r($arr_second);
  echo "</pre>";
?>
```

Результат:

```
Обычная сортировка
Array
(
    [0] => file1.txt
    [1] => file10.txt
    [2] => file12.txt
    [3] => file2.txt
)

Естественная сортировка
Array
(
    [3] => file1.txt
    [2] => file2.txt
    [1] => file10.txt
    [0] => file12.txt
)
```

Функция `array_multisort()` может быть использована для сортировки сразу нескольких массивов или одного многомерного массива. Функция имеет следующий синтаксис:

```
array_multisort($ar1 [,$arg [, ... [, ...]]] )
```

Массивы, которые передаются функции `array_multisort()`, должны содержать одинаковое количество аргументов и все вместе воспринимаются как своеобразная таблица. Сортировке подвергается лишь первый массив, а значения последующих массивов выстраиваются в соответствии с ним (рис. 6.3).

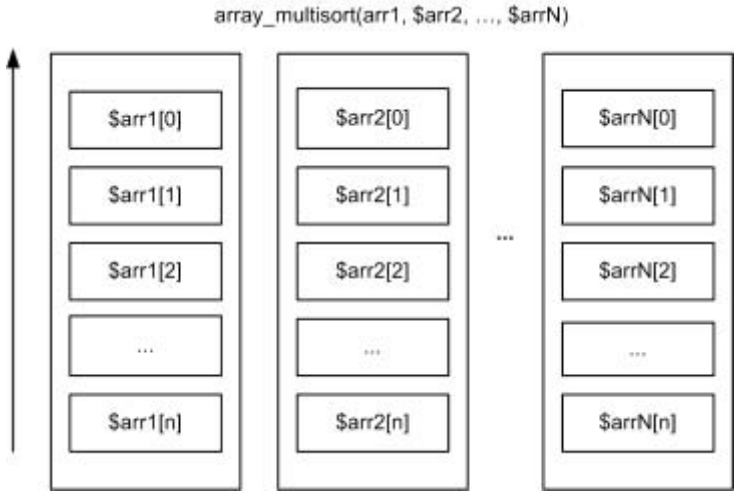


Рис. 6.3. Аргументы функции `array_multisort()` образуют таблицу

В листинге 6.69 приводится пример сортировки двух массивов `$arr1` и `$arr2`.

Листинг 6.69. Использование функции `array_multisort()`

```
<?php
  $arr1 = array(3, 4, 2, 7, 1, 5);
  $arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
  array_multisort($arr1, $arr2);
  echo "<pre>";
  print_r($arr1);
  print_r($arr2);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.69 будут следующие дампы массивов:

```
Array
(
    [0] => 1
```

```
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 7
)
Array
(
    [0] => apple
    [1] => yes
    [2] => world
    [3] => Hello
    [4] => wet
    [5] => no
)
```

Как видно из результатов, первый массив был отсортирован по возрастанию, а элементы второго массива выстроены в соответствии с первым так, как если бы между элементами двух массивов существовала связь.

Помимо массивов функция `array_multisort()` в качестве аргументов может принимать константы, задающие порядок сортировки (табл. 6.12).

> ***Замечание***
>
> Группы констант `SORT_ASC` и `SORT_DESC`, а также `SORT_REGULAR`, `SORT_NUMERIC` и `SORT_STRING` являются взаимоисключающими.

Таблица 6.12. *Константы, определяющие поведение функции* `array_multisort()`

Константа	Описание
`SORT_ASC`	Сортировать в возрастающем порядке
`SORT_DESC`	Сортировать в убывающем порядке
`SORT_REGULAR`	Сравнивать элементы обычным образом
`SORT_NUMERIC`	Сравнивать элементы в предположении, что это числа
`SORT_STRING`	Сравнивать элементы в предположении, что это строки

В листинге 6.70 приводится модифицированный вариант скрипта, сортирующий таблицу в обратном порядке.

Листинг 6.70. Использование констант

```
<?php
  $arr1 = array(3, 4, 2, 7, 1, 5);
  $arr2 = array("world", "Hello", "yes", "no", "apple", "wet");
```

```
  array_multisort($arr1, SORT_DESC, SORT_NUMERIC, $arr2);
  echo "<pre>";
  print_r($arr1);
  print_r($arr2);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 6.70 будут следующие дампы массивов:

```
Array
(
    [0] => 7
    [1] => 5
    [2] => 4
    [3] => 3
    [4] => 2
    [5] => 1
)
Array
(
    [0] => no
    [1] => wet
    [2] => Hello
    [3] => world
    [4] => yes
    [5] => apple
)
```

6.17. Добавление/удаление элементов массива

Удаление элемента массива, как и любой другой переменной, осуществляется при помощи конструкции `unset()` (листинг 6.71).

Листинг 6.71. Удаление элемента массива

```
<?php
  $arr = array(1, 2, 3, 4, 5);

  // Удаляем третий элемент массива
  unset($arr[2]);

  // Выводим дамп массива
  echo "<pre>";
```

```
  print_r($arr);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 6.71 в окно браузера будет выведен следующий дамп массива `$arr`:

```
Array
(
    [0] => 1
    [1] => 2
    [3] => 4
    [4] => 5
)
```

При помощи конструкции `unset()` может быть уничтожен и весь массив целиком (листинг 6.72).

Листинг 6.72. Удаление массива

```
<?php
  $arr = array(1, 2, 3, 4, 5);

  // Удаляем третий элемент массива
  unset($arr);

  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Если в конфигурационном файле php.ini включено отображение замечаний (Notice), обращение к несуществующему массиву `$arr` вызовет генерацию замечания "Notice: Undefined variable: arr".

Удалить все элементы массива можно также при помощи пустой конструкции `array()`. В листинге 6.73 демонстрируется обнуление массива. Следует отметить, что несмотря на то, что массив `$arr` не содержит ни одного элемента, функция `is_array()` возвращает для него `TRUE`.

Листинг 6.73. Обнуление массива

```
<?php
  $arr = array(1, 2, 3, 4, 5);

  // Обнуляем массив
```

```
  $arr = array();

  if (is_array($arr)) echo "Это массив<br>";
  else echo "Это не массив<br>";

  if (empty($arr)) echo "Переменная пустая<br>";
  else echo "Переменная не пустая<br>";
?>
```

Результатом работы скрипта из листинга 6.73 будут следующие строки

```
Это массив
Переменная пустая
```

Помимо стандартных функций удаления переменных, имеется большое количество специализированных функций для добавления и удаления элементов массива (табл. 6.13).

***Таблица 6.13.** Функции добавления и удаления элементов массива*

Функция	Описание
`array_pad($arr, $pad_size, $pad_value)`	Возвращает копию массива `$arr`, размер которого увеличивается до величины `$pad_size`. Новые элементы массива заполняются значением из параметра `$pad_value`. Если величина `$pad_size` положительная, новые элементы добавляются в конец массива, если отрицательная — в начало. Если размер массива больше величины `$pad_size`, функция `array_pad()` не производит никаких действий
`array_pop($arr)`	Извлекает элемент с вершины стека (последний элемент массива), удаляет его и возвращает массив `$arr` без удаленного элемента
`array_push($arr, $var [, ...])`	Использует массив `$arr` как стек, помещая все переданные в параметре значения в конец массива
`array_reduce($arr, $function [, $initial])`	Применяет функцию `$function` ко всем элементам массива `$arr`, от первого до последнего. Если массив пустой или достигнут конец, возвращается значение `$initial`. Функция обычно применяется для перемножения или суммирования элементов массива `$arr`
`array_shift($arr)`	Удаляет первый элемент массива `$arr` и возвращает его, т. е. в отличие от функции `array_pop()`, функция `array_shift()` извлекает начальный, а не конечный элемент массива. При этом оставшиеся элементы массива сдвигаются на одну позицию к началу массива. Функция подробнее обсуждается в *разд. 6.17*
`array_unshift($arr, mixed $var [, ...])`	Добавляет в начало массива `$arr` несколько новых элементов, значения которых перечисляются вслед за параметром `$arr`. Функция подробнее обсуждается в *разд. 6.17*

Функция `array_pad()` предназначена для увеличения размера до заданной величины и имеет следующий синтаксис:

```
array_pad($arr, $pad_size, $pad_value)
```

Функция возвращает копию массива `$arr`, размер которого был увеличен до значения параметра `$pad_size` элементами со значением `$pad_value`. Если `$pad_size` является положительным числом, то массив увеличивается с конца, если отрицательным, то сначала. В случае, когда `$pad_size` меньше или равно числу элементов исходного массива, функция не производит никаких операций. В листинге 6.74 демонстрируется пример с функцией `array_pad()`.

Листинг 6.74. Увеличение размера массива

```
<?php
  $arr = array(12, 10, 9); // Исходный массив
  $result = array_pad($arr, 6, 0);
  // Увеличиваем размер массива до шести элементов нулями справа
  //   результат: array(12, 10, 9, 0, 0, 0)
  $result = array_pad($arr, -6, -1);
  // Увеличиваем размер массива до шести элементов значениями -1 слева
  // результат: array(-1, -1, -1, 12, 10, 9)
?>
```

Функция `array_push()` добавляет к массиву новые элементы и имеет следующий синтаксис:

```
array_push($arr, $var1 [, $var2, ...])
```

Функция добавляет к массиву `$arr` элементы `$var1`, `$var2` и т. д. При добавлении элементов происходит присвоение им числовых индексов. Функция `array_push()` работает с массивом, как со стеком, и добавляет элементы всегда в конец массива (на вершину стека) (листинг 6.75).

Листинг 6.75. Добавление элемента в стек

```
<?php
  $arr = array("красный", "синий");
  array_push($arr, "зеленый");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Исходный массив `$arr` имел вид:

```
Array
(
```

```
  [0] => красный
  [1] => синий
)
```

Результат после добавления элемента "зеленый":

```
Array
(
  [0] => красный
  [1] => синий
  [2] => зеленый
)
```

Функция `array_pop()` извлекает элемент с вершины стека (последний элемент массива), удаляет его и возвращает массив без удаленного элемента. Функция имеет следующий синтаксис:

```
array_pop($arr);
```

Если на момент удаления массив `$arr` был пуст, функция возвращает пустую строку (листинг 6.76).

Листинг 6.76. Удаление элемента из стека

```
<?php
  $arr = array("красный", "синий", "зеленый");
  array_pop($arr);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат:

```
Array
(
  [0] => красный
  [1] => синий
)
```

Функция `array_unshift()` очень похожа на функцию `array_push()`, но добавляет перечисленные элементы не в конец, а в начало массива. Функция имеет следующий синтаксис:

```
array_unshift($arr, $var1 [, var2, ...])
```

При этом порядок следования элементов других остается тем же. Новые элементы массива индексируются, начиная с 0, при этом, конечно, индексы тех элементов, что уже были в массиве, изменяются. Функция возвращает новый размер массива (листинг 6.77).

Листинг 6.77. Функция `array_unshift()`

```
<?php
  $arr = array("красный", "синий");

  echo("Исходный массив:<br>");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";

  array_unshift($arr, "зеленый", "оранжевый");

  echo "<br>После добавления элементов:<br>";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат:

```
Исходный массив:
Array
(
  [0] => красный
  [1] => синий
)
После добавления элементов:
Array
(
  [0] => зеленый
  [1] => оранжевый
  [2] => красный
  [3] => синий
)
```

Функция `array_shift()` удаляет первый элемент массива `$arr` и возвращает его, т. е. в отличие от функции `array_pop()` функция `array_shift()` извлекает начальный, а не конечный элемент массива. При этом оставшиеся элементы массива сдвигаются на одну позицию к началу массива. Функция имеет следующий синтаксис:

```
array_shift($arr)
```

Пример с использованием функции `array_shift()` приводится в листинге 6.78.

Листинг 6.78. Функция `array_shift()`

```
<?php
  $arr = array("красный",
               "синий",
               "зеленый",
               "оранжевый");
  echo "Исходный массив:<br>";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";

  array_shift($arr);

  echo "<br>После удаления элементов:<br>";
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат:

```
Исходный массив:
Array
(
  [0] => красный
  [1] => синий
  [2] => зеленый
  [3] => оранжевый
)
После удаления элементов:
Array
(
  [0] => синий
  [1] => зеленый
  [2] => оранжевый
)
```

6.18. Работа с ключами массива

Данная группа функций предназначена для работы с ключами массива (табл. 6.14).

Таблица 6.14. *Функции для работы с ключами массива*

Функция	Описание
`array_reverse($arr [, $preserve_keys])`	Возвращает копию массива `$arr`, меняя порядок следования элементов в нем на обратный. Если необязательный параметр `$preserve_keys` принимает значение `TRUE`, связь значений с ключами сохраняется
`array_change_key_case ($arr [, $case])`	Возвращает копию массива `$arr`, в котором изменен регистр ключей. Если необязательный параметр `$case` принимает в качестве значение константу `CASE_UPPER`, ключи массива `$arr` преобразуются в верхний регистр, если константу `CASE_LOWER` — в нижний. По умолчанию, если параметр `$case` не указывается, ключи преобразуются в нижний регистр
`array_values($arr)`	Возвращает копию массива `$arr`, в которой все ключи заменены на числовые индексы
`array_flip($arr)`	Возвращает инвертированную копию массива `$arr` — ключи становятся значениями, а значения — ключами. Если значение встречается несколько раз, будет возвращено только последнее, все остальные будут отброшены
`array_keys(arr [, $search_value [, $strict]])`	Возвращает массив ключей для массива `$arr`. Если указан необязательный параметр `$search_value`, возвращаются только те ключи, которые указывают на элементы со значением `$search_value`. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$search_value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==`

Первая функция `array_reverse()` возвращает массив, элементы которого следуют в обратном порядке относительно массива `$arr`. Функция имеет следующий синтаксис:

```
array_reverse($arr [, $preserve_keys])
```

При этом связи между ключами и значениями сохраняются. Если необязательному аргументу `$preserve_keys` присвоить значение `TRUE`, тогда в обратном порядке отсортируются также и ключи. Листинг 6.79 демонстрирует работу функции `array_reverse()`.

Листинг 6.79. Функция `array_reverse()`

```
<?php
  $arr  = array("php", 5.0, array("green", "red"));
  $result = array_reverse($arr);
```

```
  foreach($result as $key => $val)
  {
    echo "$key => $val <br>";
  }
  echo "<br>";
  $result_key = array_reverse($arr, true);
  foreach($result_key as $key => $val)
  {
    echo "$key => $val <br>";
  }
?>
```

Результат:

```
0 => Array
1 => 5
2 => php
```

То есть в первом случае, когда аргумент `$preserve_keys` равен `FALSE`, массив имеет следующую структуру:

```
Array
(
  [0] => Array
   (
       [0] => green
       [1] => red
    )
  [1] => 5
  [2] => php
)
```

Когда же `$preserve_keys` равно `TRUE`, структура массива такова:

```
Array
(
  [2] => Array
   (
      [0] => green
      [1] => red
    )
  [1] => 5
  [0] => php
)
```

Функция `array_change_key_case()` изменяет регистр ключей массива и имеет следующий синтаксис:

```
array_change_key_case($arr [, $case])
```

Функция возвращает копию массива `$arr`, в котором изменен регистр ключей. Если необязательный параметр `$case` принимает в качестве значения константу `CASE_UPPER`,

ключи массива `$arr` преобразуются в верхний регистр, если константу `CASE_LOWER` — в нижний. По умолчанию, если параметр `$case` не указывается, ключи преобразуются в нижний регистр. В листинге 6.80 приводится пример использования функции.

Листинг 6.80. Использование функции `array_change_key_case()`

```
<?php
  // Исходный ассоциативный массив
  $arr = array("FirSt" => 1, "SecOnd" => 4);

  // Преобразуем ключи
  $arr = array_change_key_case($arr, CASE_UPPER);

  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [FIRST] => 1
    [SECOND] => 4
)
```

Функция `array_values()` возвращает копию ассоциативного массива, в котором все ключи заменены числовыми индексами. Функция имеет следующий синтаксис:

```
array_values($arr)
```

В листинге 6.81 приводится пример использования функции.

Листинг 6.81. Использование функции `array_values()`

```
<?php
  // Исходный ассоциативный массив
  $arr = array("size" => "XL", "color" => "gold");

  // Получаем индексный массив
  $num = array_values($arr);

  // Выводим дамп массива
  echo "<pre>";
```

```
  print_r($num);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => XL
    [1] => gold
)
```

Функция `array_flip()` возвращает инвертированную копию массива `$arr` — ключи становятся значениями, а значения — ключами. Функция имеет следующий синтаксис:

```
array_flip($arr)
```

Если значение встречается несколько раз, будет возвращено только последнее, все остальные будут отброшены. В листинге 6.82 приводится пример использования функции `array_flip()`.

Листинг 6.82. Использование функции `array_flip()`

```
<?php
  // Исходный массив
  $arr = array("a" => 1, "b" => 1, "c" => 2);

  // Инвертируем массив
  $trn = array_flip($arr);

  // Выводим дамп массива
  echo "<pre>";
  print_r($trn);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.82 будет следующий дамп массива `$trn`:

```
Array
(
    [1] => b
    [2] => c
)
```

Функция `array_keys()` позволяет вернуть массив ключей и имеет следующий синтаксис:

```
array_keys($arr [, $search_value [, $strict]])
```

Возвращает массив ключей для массива *$arr*. Если указан необязательный параметр *$search_value*, возвращаются только те ключи, которые указывают на элементы со значением *$search_value*. Если необязательный параметр *$strict* принимает значение TRUE, то при сравнении значения элемента массива с *$search_value* будет использоваться оператор эквивалентности ===, в противном случае будет использован оператор равенства ==. В листинге 6.83 приводится пример использования функции.

Листинг 6.83. Использование функции array_keys()

```
<?php
  // Исходный массив
  $arr = array(0 => 100, "color" => "red");

  // Получаем массив ключей
  $key = array_keys($arr);

  // Выводим дамп массива
  echo "<pre>";
  print_r($key);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => color
)
```

В листинге 6.84 демонстрируется использование параметра $search_value. Исходный массив содержит три элемента со значением "blue", при помощи функции array_keys() извлекаются их ключи.

Листинг 6.84. Извлечение ключей элементов со значением "blue"

```
<?php
  // Исходный массив
  $arr = array("blue", "red", "green", "blue", "blue");

  // Получаем массив ключей
  $key = array_keys($arr, "blue");

  // Выводим дамп массива
  echo "<pre>";
```

```
  print_r($key);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

6.19. Преобразование массивов

В данном разделе будут рассмотрены функции, осуществляющие преобразование массива (табл. 6.15).

Таблица 6.15. *Функции преобразования массивов*

Функция	Описание
`array_unique($arr)`	Возвращает копию массива `$arr`, содержащую только уникальные элементы
`array_chunk($arr, $size [, $preserve_keys])`	Разбивает массив `$arr` на несколько массивов с количеством элементов, равным `$size`. Если необязательный параметр `$preserve_keys` принимает значение `TRUE`, сохраняются ключи исходного массива, в противном случае элементы результирующих массивов нумеруются числами, начиная с 0
`array_filter($arr [, $callback])`	Возвращает копию массива `$arr`, в котором содержатся лишь те элементы, для которых пользовательская функция `$callback` возвращает `TRUE`
`array_walk($arr, $funcname [, $userdata])`	Применяет пользовательскую функцию `$funcname` ко всем элементам массива `$arr`. Параметр `$userdata` используется в качестве дополнительного параметра пользовательской функции
`array_walk_recursive($arr, $funcname [, $userdata])`	Функция `array_walk_recursive()` эквивалентна `array_walk()`, однако позволяет обходить многомерные массивы, рекурсивно спускаясь по ним
`array_map($callback, $arr1 [, ...])`	Применяет пользовательскую функцию `$callback` ко всем элементам массивов, указанных в последующих параметрах

Функция `array_unique()` возвращает копию массива `$arr`, содержащую только уникальные значения. Функция имеет следующий синтаксис:

```
array_unique($arr)
```

В листинге 6.85 приводится пример использования функции.

Листинг 6.85. Получение массива с уникальными значениями

```
<?php
  $arr = array("a" => "green", "red", "b" => "green", "blue", "red");
  $result = array_unique($arr);

  echo "<pre>";
  print_r($result);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.85 будет следующий дамп массива `$result`:

```
Array
(
    [a] => green
    [0] => red
    [1] => blue
)
```

Функция `array_chunk()` разбивает массив на несколько массивов заданной длины и имеет следующий синтаксис:

```
array_chunk($arr, $size [, $preserve_keys])
```

Функция возвращает копию массива *`$arr`*, разбитого на несколько массивов с количеством элементов равным *`$size`*. Если необязательный параметр `$preserve_keys` принимает значение `TRUE`, сохраняются ключи исходного массива, в противном случае элементы результирующих массивов нумеруются числами, начиная с 0. В листинге 6.86 демонстрируется использование функции `array_chunk()`.

Листинг 6.86. Использование функции `array_chunk()`

```
<?php
  $arr = array('a', 'b', 'c', 'd', 'e');

  echo "<pre>";
  print_r(array_chunk($arr, 2));
  print_r(array_chunk($arr, 2, TRUE));
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.86 будут следующие дампы массивов:

```
Array
(
```

```
    [0] => Array
       (
            [0] => a
            [1] => b
        )

    [1] => Array
       (
            [0] => c
            [1] => d
        )

    [2] => Array
       (
            [0] => e
        )
)
Array
(
    [0] => Array
       (
            [0] => a
            [1] => b
        )

    [1] => Array
       (
            [2] => c
            [3] => d
        )

    [2] => Array
       (
            [4] => e
        )
)
```

Функция `array_filter()` возвращает копию массива *`$arr`*, в котором содержатся лишь те элементы, для которых пользовательская функция *`$callback`* возвращает `TRUE`. Функция имеет следующий синтаксис:

```
array_filter($arr [, $callback])
```

Пользовательская функция *`$callback`* принимает в качестве единственного параметра текущий элемент массива и может возвращать либо `TRUE`, либо `FALSE`. Допускается возвращать и другие величины, однако они будут приводиться к логическому типу.

В листинге 6.87 представлен пример использования функции `array_filter()`, совместно с функцией `odd()`, которая возвращает `TRUE` для нечетных значений — результирующий массив имеет только нечетные значения.

Листинг 6.87. Использование функции `array_filter()`

```
<?php
  // Возвращает TRUE для нечетных
  function odd($var)
  {
    return($var & 1);
  }

  // Исходный массив
  $arr = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);

  // Извлекаем только нечетные значения
  $odd = array_filter($arr, "odd");

  // Выводим дамп массива
  echo "<pre>";
  print_r($odd);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 6.87 будет следующий дамп массива `$odd`:

```
Array
(
    [a] => 1
    [c] => 3
    [e] => 5
)
```

Функция `array_walk()` позволяет применять пользовательскую функцию к каждому элементу массива. Она имеет следующий синтаксис:

```
array_walk($arr, $funcname [, $userdata])
```

Как видно из синтаксиса, функция `array_walk()` применяет пользовательскую функцию `$funcname` к каждому элементу массива `$arr`. В пользовательскую функцию передаются два или три аргумента: значение текущего элемента, его индекс и аргумент `$userdata`. Последний аргумент является необязательным. В случае если `$funcname` требует более трех аргументов, при каждом ее вызове будет выдаваться предупреждение, и, чтобы они не выдавались, нужно поставить знак `@` перед функцией `array_walk()`.

Функция `$funcname` получает значения и индексы массива `$array` таким образом, что не может их изменять. Если возникает необходимость в таких изменениях, нужно передавать аргумент `$arr` по ссылке. В этом случае все изменения отразятся в массиве.

Пусть необходимо вывести все элементы массива. Для этого можно сначала написать функцию, которая будет их выводить, а затем вызвать ее для каждого из элементов массива (листинг 6.88).

> ***ЗАМЕЧАНИЕ***
>
> Функция `array_walk()` имеет модификацию `array_walk_recursive()`, которая позволяет применять пользовательскую функцию к многомерным массивам.

Листинг 6.88. Использование функции `array_walk()`

```
<?php
  $name = array("m"=>"maksim", "i"=>"igor", "s"=>"sergey");
  function print_array($item, $key)
  {
    echo "$key=>$item<br>\n";
  }
  /* теперь применим функцию print_array
  к каждому элементу массива $name */
  array_walk($name, 'print_array');
?>
```

Результат:

```
m => maksim
i => igor
s => sergey
```

Функция `array_map()` также предназначена для применения пользовательской функции к элементам массивов. Функция имеет следующий синтаксис:

```
array_map($callback, $arr1 [, ...])
```

Функция применяет пользовательскую функцию `$callback` ко всем элементам массивов, указанных в последующих параметрах. Количество параметров, передаваемых функции обратного вызова, должно совпадать с количеством массивов, переданных функции `array_map()`. В листинге 6.89 демонстрируется работа функции — массив `$arr` преобразуется в массив `$cub`, элементы которого представляют кубическую степень от элементов массива `$arr`.

Листинг 6.89. Возведение элементов массива в куб

```
<?php
  // Пользовательская функция для возведения элементов массива в куб
  function cube($n)
```

```
  {
    return $n*$n*$n;
  }

  // Исходный массив
  $arr = array(1, 2, 3, 4, 5);

  // Возводим элементы массива в куб
  $cub = array_map("cube", $arr);

  // Выводим дамп массива
  echo "<pre>";
  print_r($cub);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

В листинге 6.90 при помощи функции `array_map()` элементы массивов `$fst` и `$snd` перемножаются друг на друга.

> ***Замечание***
>
> Если функции `array_map()` в качестве параметра передается более чем один массив и их длина не совпадает, самый маленький из них дополняется нулевыми элементами.

Листинг 6.90. Перемножение элементов массивов `$fst` и `$snd`

```
<?php
  // Пользовательская функция для перемножения элементов массива
  function multi($n, $m)
  {
    return $n*$m;
  }

  // Исходные массивы
  $fst = array(1, 2, 3, 4, 5);
```

```
  $snd = array(5, 4, 3, 2, 1);

  // Перемножаем элементы массивов
  $res = array_map("multi", $fst, $snd);

  // Выводим дамп массива
  echo "<pre>";
  print_r($res);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => 5
    [1] => 8
    [2] => 9
    [3] => 8
    [4] => 5
)
```

Если вместо пользовательской функции передается значение `NULL`, функция `array_map()` возвращает двумерный массив (листинг 6.91).

Листинг 6.91. Использование `NULL` вместо пользовательской функции

```
<?php
  // Исходные массивы
  $a = array(1, 2, 3, 4, 5);
  $b = array("one", "two", "three", "four", "five");
  $c = array("uno", "dos", "tres", "cuatro", "cinco");

  // Объединяем массивы
  $res = array_map(NULL, $a, $b, $c);

  // Выводим дамп массива
  echo "<pre>";
  print_r($res);
  echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => Array
```

```
        (
            [0] => 1
            [1] => one
            [2] => uno
        )
    [1] => Array
        (
            [0] => 2
            [1] => two
            [2] => dos
        )
    [2] => Array
        (
            [0] => 3
            [1] => three
            [2] => tres
        )
    [3] => Array
        (
            [0] => 4
            [1] => four
            [2] => cuatro
        )
    [4] => Array
        (
            [0] => 5
            [1] => five
            [2] => cinco
        )
)
```

6.20. Суперглобальные массивы. Массив *$_SERVER*

PHP содержит ряд предопределенных массивов, которые доступны из любой части программы. Такие массивы заполняются автоматически интерпретатором PHP либо данными, полученными от клиента, либо переменными окружения. По мере изложения материала будут рассмотрены все переменные окружения, тут остановимся на предопределенном массиве `$_SERVER` — в него PHP-интерпретатор помещает переменные окружения Web-сервера Apache. Без данных переменных сложно организовать полноценную поддержку Web-приложений. Далее приводится описание наиболее важных элементов суперглобального массива `$_SERVER`.

Замечание

Просмотреть полный список элементов массива `$_SERVER` можно либо при помощи функции `print_r()`, которая распечатывает дамп массива, либо при помощи функции `phpinfo()`, которая выводит информацию о PHP-интерпретаторе.

6.20.1. Элемент *$_SERVER['DOCUMENT_ROOT']*

Элемент `$_SERVER['DOCUMENT_ROOT']` содержит путь к корневому каталогу сервера. Если скрипт выполняется в виртуальном хосте, в данном элементе, как правило, указывается путь к корневому каталогу виртуального хоста. То есть в конфигурационном файле httpd.conf виртуальный хост имеет директиву `DocumentRoot`, которой присвоено значение `"D:/main"` (листинг 6.92), элемент `$_SERVER['DOCUMENT_ROOT']` будет содержать значение `"D:\main"`.

Листинг 6.92. Виртуальный хост

```
NameVirtualHost 127.0.0.1:80
<VirtualHost 127.0.0.1:80>
    ServerAdmin admin@localhost
    DocumentRoot "D:/main"
    ServerName localhost
    ErrorLog logs/ error_log
    CustomLog logs/ access_log common
</VirtualHost>
```

6.20.2. Элемент *$_SERVER['HTTP_ACCEPT']*

В элементе `$_SERVER['HTTP_ACCEPT']` описываются предпочтения клиента относительно типа документа. Содержимое этого элемента извлекается из HTTP-заголовка `Accept`, который присылает клиент серверу. Содержимое данного заголовка может выглядеть следующим образом

```
image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-
flash, application/vnd.ms-excel, application/msword, */*
```

Заголовок `Accept` позволяет уточнить медиатип, который предпочитает получить клиент в ответ на свой запрос. Этот заголовок позволяет сообщить серверу, что ответ ограничен небольшим множеством предпочитаемых типов.

Символ `*` используется для группирования типов в медиаряду. К примеру, символами `*/*` задается использование всех типов, а обозначение `type/*` определяет использование всех подтипов выбранного типа `type`.

Замечание

Медиатипы отделяются друг от друга запятыми.

Каждый медиаряд характеризуется также дополнительным набором параметров. Одним из них является так называемый относительный коэффициент предпочтения `q`,

который принимает значения от 0 до 1, соответственно, от менее предпочитаемых типов к более предпочитаемым. Использование нескольких параметров q, позволяет клиенту сообщить серверу относительную степень предпочтения для того или иного медиатипа.

> ***Замечание***
>
> По умолчанию параметр q принимает значение 1. Кроме того, от медиатипа он отделяется точкой с запятой.

Пример заголовка типа Accept:

```
Accept: audio/*; q=0.2, audio/basic
```

В данном заголовке первым идет тип audio/*, включающий в себя все музыкальные документы и характеризующийся коэффициентом предпочтения 0.2. Через запятую указан тип audio/basic, для которого коэффициент предпочтения не указан и принимает значение по умолчанию, равное единице. Цитируя RFC 2616, данный заголовок можно интерпретировать следующим образом: "Я предпочитаю тип audio/basic, но мне можно также слать документы любого другого аудиотипа, если они будут доступны, после снижения коэффициента предпочтения более чем на 80%".

> ***Замечание***
>
> Загрузить документ RFC 2616, в котором описывается протокол HTTP, можно по адресу **http://www.ysn.ru/docs/cie/RFC/2616/index.htm**.

Пример может быть более сложным:

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
```

> ***Замечание***
>
> Следует учитывать, что элемент $_SERVER['HTTP_ACCEPT'] содержит точно такую же информацию, но без начального заголовка Accept:.

Этот заголовок интерпретируется следующим образом: типы документов text/html и text/x-c являются предпочтительными, но если они недоступны, тогда клиент, отсылающий данный запрос, предпочтет text/x-dvi, а если и его нет, то он может принять тип text/plain.

6.20.3. Элемент *$_SERVER['HTTP_ACCEPT_LANGUAGE']*

В элементе $_SERVER['HTTP_ACCEPT_LANGUAGE'] описываются предпочтения клиента относительно языка. Данная информация извлекается из HTTP-заголовка Accept-Language, который присылает клиент серверу. Можно привести следующий пример:

```
Accept-Language: ru, en; q=0.7
```

Его можно интерпретировать следующим образом: клиент предпочитает русский язык, но в случае его отсутствия согласен принимать документы на английском. Элемент $_SERVER['HTTP_ACCEPT_LANGUAGE'] будет содержать точно такую же информацию, но без заголовка Accept-Language::

```
ru, en; q=0.7
```

Содержимое элемента `$_SERVER['HTTP_ACCEPT_LANGUAGE']` можно использовать для определения национальной принадлежности посетителей. Однако результаты будут приблизительными, т. к. многие пользователи используют английские варианты браузеров, которые будут извещать сервер о том, что посетитель предпочитает лишь один язык — английский.

6.20.4. Элемент *$_SERVER['HTTP_HOST']*

В элементе `$_SERVER['HTTP_HOST']` содержится имя сервера, которое, как правило, совпадает с доменным именем сайта, расположенного на сервере. Как правило, имя, указанное в данном параметре, совпадает с именем `$_SERVER['SERVER_NAME']`. В параметре приводится лишь доменное имя без названия протокола (http://), например

```
www.sofftime.ru
```

> ***Замечание***
>
> Некоторые хост-провайдеры помещают в данную переменную окружения домен третьего уровня **www.softtime.ru**, ряд — домен второго уровня — **softtime.ru**.

6.20.5. Элемент *$_SERVER['HTTP_REFERER']*

В элемент `$_SERVER['HTTP_REFERER']` помещается адрес страницы, с которой посетитель пришел на данную страницу. Переход должен осуществляться по ссылке. Создадим две страницы: index.php (листинг 6.93) и page.php (листинг 6.94).

Листинг 6.93. Страница index.php

```
<?php
  echo "<a href=page.php>Ссылка на страницу PHP</a><br>";
  echo "Содержимое \$_SERVER['HTTP_REFERER'] - ".
        $_SERVER['HTTP_REFERER']
?>
```

Страница page.php будет аналогичного содержания, но ссылка будет указывать на страницу index.php (листинг 6.93).

Листинг 6.94. Страница page.php

```
<?php
  echo "<a href=index.php>Ссылка на страницу PHP</a><br>";
  echo "Содержимое \$_SERVER['HTTP_REFERER'] - ".
        $_SERVER['HTTP_REFERER']
?>
```

При переходе с одной страницы на другую под ссылкой будет выводиться адрес страницы, с которой был осуществлен переход (рис. 6.4).

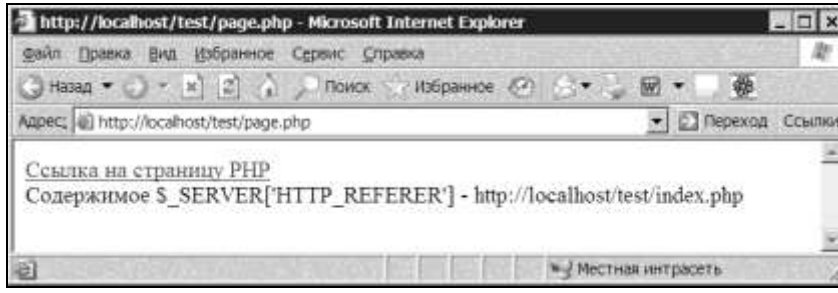


Рис. 6.4. Фиксация адреса предыдущей страницы

6.20.6. Элемент *$_SERVER['HTTP_USER_AGENT']*

Элемент `$_SERVER['HTTP_USER_AGENT']` содержит информацию о типе и версии браузера и операционной системы посетителя.

Вот типичное содержание этой строки: `"Mozilla/4.0(compatible; MSIE 6.0; Windows NT 5.1)"`. Наличие подстроки `"MSIE 6.0"` говорит о том, что посетитель просматривает страницу при помощи Internet Explorer версии 6.0. Строка `"Windows NT 5.1"` сообщает, что в качестве операционной системы используется Windows XP.

> ***Замечание***
>
> Для Windows 2000 элемент `$_SERVER['HTTP_USER_AGENT']` выглядит следующим образом: `"Mozilla/4.0(compatible; MSIE 5.01; Windows NT 5.0)')"`, в то время как для Windows XP — `"Mozilla/4.0(compatible; MSIE 6.0; Windows NT 5.1)"`.

Если посетитель воспользуется браузером Opera, то содержание `$_SERVER['HTTP_USER_AGENT']` может выглядеть следующим образом: `"Mozilla/4.0(compatible; MSIE 5.0; Windows 98) Opera 6.04 [ru]"`. Подстрока `"MSIE 6.0"` здесь так же присутствует, сообщая, что браузер Opera является совместимым с браузером Internet Explorer и использует те же динамические библиотеки Windows. Поэтому, при анализе строки, возвращаемой браузером, следует иметь в виду, что к Internet Explorer относится строка, содержащая подстроку `"MSIE 6.0"` и не содержащая подстроки `"Opera"`. Кроме того, из данной строки можно заключить, что пользователь использует операционную систему Windows 98.

При использовании браузера Netscape содержание элемента `$_SERVER['HTTP_USER_AGENT']` может выглядеть следующим образом: `"Mozilla/5.0(X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624 Netscape/7.1"`. Принадлежность к этому браузеру можно определить по наличию подстроки `"Netscape"`. Кроме того, можно узнать, что посетитель выходит в Интернет, используя операционную версию Linux, с ядром, оптимизированным под Pentium IV, находясь в графической оболочке X-Window. Этот механизм удобно использовать для сбора статистической информации, которая позволяет дизайнерам оптимизировать страницы под наиболее распространенные браузеры.

6.20.7. Элемент $_SERVER['REMOTE_ADDR']

В элемент `$_SERVER['REMOTE_ADDR']` помещается IP-адрес клиента. При тестировании на локальной машине этот адрес будет равен 127.0.0.1. Однако при тестировании в сети переменная вернет IP-адрес клиента или последнего прокси-сервера, через который клиент попал на сервер. Если клиент использует прокси-сервер, узнать его IP-адрес можно при помощи переменной окружения `$_SERBER['HTTP_X_FORWARDED_FOR']`.

> ***Замечание***
>
> Прокси-серверы являются специальными промежуточными серверами, предоставляющими специальный вид услуг: сжатие трафика, кодирование данных, адаптацию под мобильные устройства и т. п. Среди множества прокси-серверов различают так называемые анонимные прокси-серверы, которые позволяют скрывать истинный IP-адрес клиента, такие серверы не возвращают переменную окружения `HTTP_X_FORWARDED_FOR`.

6.20.8. Элемент $_SERVER['SCRIPT_FILENAME']

В элемент `$_SERVER['SCRIPT_FILENAME']` помещается абсолютный путь к файлу от корня диска. Так, если сервер работает под управлением операционной системы Windows, то такой путь может выглядеть следующим образом `"d:\main\test\index.php"`, т. е. путь указывается от диска, в UNIX-подобной операционной системе путь указывается от корневого каталога `/`, например `"/var/share/www/test/index.php"`.

6.20.9. Элемент $_SERVER['SERVER_NAME']

В элемент `$_SERVER['SERVER_NAME']` помещается имя сервера, как правило, совпадающее с доменным именем сайта, расположенного на нем. Например,

```
www.softtime.ru
```

Содержимое элемента `$_SERVER['SERVER_NAME']` часто совпадает с содержимым элемента `$_SERVER['HTTP_HOST']`. Помимо имени сервера суперглобальный массив `$_SERVER` позволяет выяснить еще ряд параметров сервера, например IP-адрес сервера, прослушиваемый порт, какой Web-сервер установлен и версию HTTP протокола. Эта информация помещается в элементы `$_SERVER['SERVER_ADDR']`, `$_SERVER['SERVER_PORT']`, `$_SERVER['SERVER_SOFTWARE']` и `$_SERVER['SERVER_PROTOCOL']`, соответственно. В листинге 6.95 приводится пример с использованием данных элементов.

Листинг 6.95. Использование элементов массива $_SERVER

```
<?php
  echo "Имя сервера - ".$_SERVER['SERVER_NAME']."<br>";
  echo "IP-адрес сервера - ".$_SERVER['SERVER_ADDR']."<br>";
  echo "Порт сервера - ".$_SERVER['SERVER_PORT']."<br>";
  echo "Web-сервер - ".$_SERVER['SERVER_SOFTWARE']."<br>";
  echo "Версия HTTP-протокола - ".$_SERVER['SERVER_PROTOCOL']."<br>";
?>
```

Результат работы скрипта из листинга 6.95 представлен на рис. 6.5.

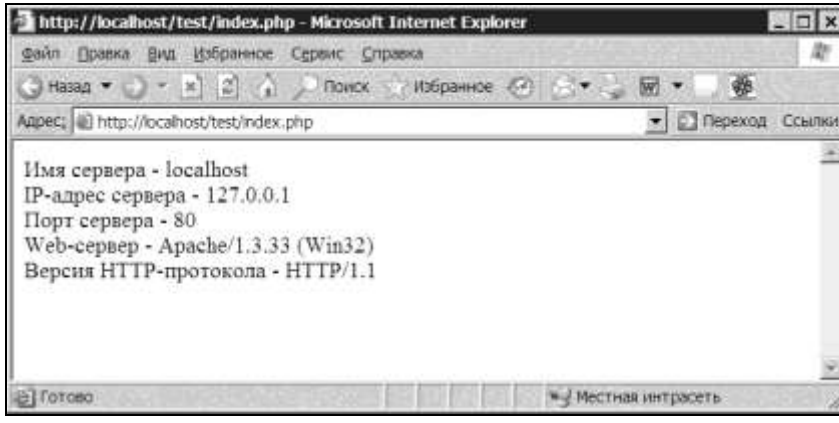


Рис. 6.5. Результат работы скрипта из листинга 6.95

6.20.10. Элемент *$_SERVER['REQUEST_METHOD']*

В элемент `$_SERVER['REQUEST_METHOD']` помещается метод запроса, который применяется для вызова скрипта: GET или POST (листинг 6.96).

Листинг 6.96. Определение метода запроса

```
<?php
  echo $_SERVER['REQUEST_METHOD']; // GET
?>
```

6.20.11. Элемент *$_SERVER['QUERY_STRING']*

В элемент `$_SERVER['QUERY_STRING']` заносятся параметры, переданные скрипту. Если строка запроса представляет собой адрес

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то в элемент `$_SERVER['QUERY_STRING']` попадет весь текст после знака `?`. Например, при обращении к скрипту, представленному в листинге 6.97, помещая в строке запроса произвольный текст после знака `?`, получим страницу с введенным текстом.

Листинг 6.97. Использование элемента `$_SERVER['QUERY_STRING']`

```
<?php
  echo $_SERVER['QUERY_STRING'];
?>
```

Результат работы скрипта из листинга 6.97 представлен на рис. 6.6.

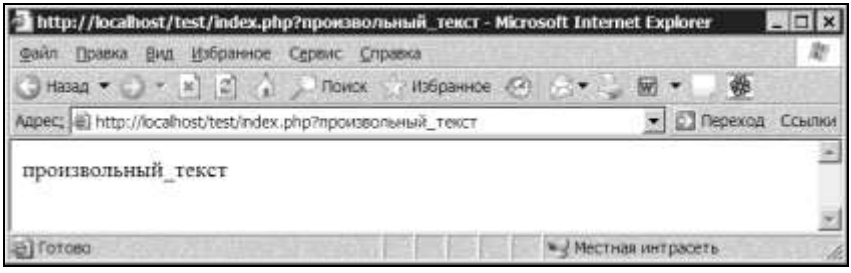


Рис. 6.6. Результат работы скрипта из листинга 6.97

6.20.12. Элемент *$_SERVER['PHP_SELF']*

В элемент `$_SERVER['PHP_SELF']` помещается имя скрипта, начиная от корневого каталога виртуального хоста, т. е. если строка запроса представляет собой адрес

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то элемент `$_SERVER['PHP_SELF']` будет содержать фрагмент `"/test/index.php"`. Как правило, этот же фрагмент помещается в элемент `$_SERVER['SCRIPT_NAME']`.

6.20.13. Элемент *$_SERVER['REQUEST_URI']*

В элемент `$_SERVER['REQUEST_URI']` содержит имя скрипта, начиная от корневого каталога виртуального хоста, и параметры, т. е. если строка запроса представляет собой адрес:

```
http://www.mysite.ru/test/index.php?id=1&test=wet&id_theme=512
```

то элемент `$_SERVER['REQUEST_URI']` будет содержать фрагмент `"/test/index.php?id=1&test=wet&id_theme=512"`. Для того чтобы восстановить в скрипте полный адрес, который помещен в строке запроса, достаточно использовать комбинацию элементов массива `$_SERVER`, представленную в листинге 6.98.

Листинг 6.98. Полный адрес к скрипту

```
<?php
  echo "http://".$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'];
?>
```



Глава 7

Функции

В предыдущих главах встроенные функции уже неоднократно описывались и применялись для самых разнообразных задач, от преобразования типа до настройки чувствительности PHP-интерпретатора к ошибкам.

Помимо встроенных функций каждый язык высокого уровня позволяет разработчикам создавать собственные пользовательские функции. Пользовательская функция представляет собой самостоятельный фрагмент программы, предназначенный для реализации определенных действий, выполнение которых достигается вызовом функции в требуемом месте программы. Это позволяет исключить дублирующие фрагменты кода. Обычно функция принимает несколько аргументов, выполняет с использованием этих аргументов некоторые операции и, если нужно, возвращает полученное значение.

7.1. Объявление и вызов функции

Функция объявляется при помощи ключевого слова `function`, после которого следует имя функции, в круглых скобках параметры функции и в фигурных скобках записываются различные операторы, составляющие тело функции:

```
function MyFunction()
{
  // операторы
}
```

Если функция возвращает какое-либо значение, в теле функции обязательно должен присутствовать оператор `return`:

```
function MyFunction()
{
  // Вычисления
  return $ret; // возвращается значение переменной $ret
}
```

Листинг 7.1. Пример простой функции

```
<?php
  function get_sum()
  {
```

```
    $sum = 10 + 5;
    return $sum;
  }
  echo get_sum(); // выводит 15
?>
```

В листинге 7.1 демонстрируется функция, вычисляющая сумму двух чисел. Эта функция не принимает ни одного аргумента, а просто вычисляет сумму и возвращает полученный результат. После этого она вызывается в теле оператора `echo` для вывода результата в браузер. Модифицируем эту функцию так, чтобы она не возвращала полученный результат, а выводила его в браузер. Для этого достаточно внести оператор `echo` в тело функции (листинг 7.2).

Листинг 7.2. Функция `get_sum()` сама выводит результат в браузер

```
<?php
  function get_sum()
  {
    $sum = 10 + 5;
    echo $sum;
  }
  get_sum();
?>
```

Во многих языка программирования функция не может вызываться до ее объявления. В PHP отсутствуют подобные ограничения, функция может вызываться до ее объявления (листинг 7.3).

Листинг 7.3. Вызов может осуществляться до объявления функции

```
<?php
  get_sum();
  function get_sum()
  {
    $sum = 10 + 5;
    echo $sum;
  }
?>
```

Это правило изменяется, если объявление функции осуществляется внутри фигурных скобок. Функции могут быть объявлены в блоке, обрамленном фигурными скобками. Такой способ объявления функций часто используется, если объявление должно быть условным (листинг 7.4).

Листинг 7.4. Условное объявление функции

```
<?php
  // Объявляем логическую переменную
  $flag = TRUE;

  // Если переменная $flag равна TRUE, объявляем функцию
  if ($flag)
  {
    function get_sum()
    {
      $sum = 10 + 5;
      echo $sum;
    }
  }

  // Вызываем функцию, если переменная $flag равна TRUE
  if ($flag) get_sum(); // 15
?>
```

Результатом выполнения скрипта из листинга 7.4 будет вывод числа 15 в окно браузера. Однако объявить функцию позднее ее вывода в этом случае уже не получится (листинг 7.5).

Листинг 7.5. При условном объявлении вызвать функцию раньше объявления нельзя

```
<?php
  // Объявляем логическую переменную
  $flag = TRUE;

  // Вызываем функцию, если переменная $flag равна TRUE
  if ($flag) get_sum(); // Ошибка

  // Если переменная $flag равна TRUE, объявляем функцию
  if ($flag)
  {
    function get_sum()
    {
      $sum = 10 + 5;
      echo $sum;
    }
  }
?>
```

Попытка вызова функции, объявленной условно, раньше объявления приводит к генерации ошибки (рис. 7.1).

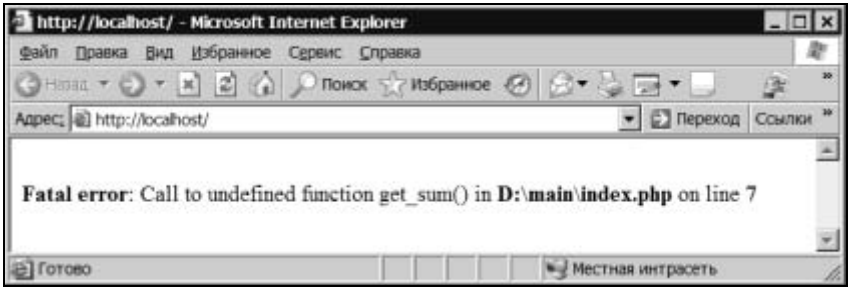


Рис. 7.1. При условном объявлении функции ее вызов раньше объявления не допускается

7.2. Параметры функции

Можно значительно увеличить гибкость функции из листинга 7.2, если складываемые числа будут передаваться в качестве параметров (листинг 7.6).

Листинг 7.6. Передача складываемых функций через аргументы

```
<?php
  function get_sum($left, $right)
  {
    $sum = $left + $right;
    echo $sum;
  }
  get_sum(10, 5); // 15
?>
```

Переменная, содержащая значение, переданное через аргумент, называется *параметром* функции, т. е. в примере, показанном в листинге 7.6, числа 10 и 5 являются аргументами, а переменные `$left` и `$right` — параметрами.

В качестве параметров могут выступать выражения, и даже другие функции. В PHP выражения в этом случае вычисляются слева направо (листинг 7.7).

Листинг 7.7. Вычисление параметров осуществляется слева направо

```
<?php
  function funct($left, $middle, $right)
  {
    echo $left . "<br>";
```

```
    echo $middle . "<br>";
    echo $right . "<br>";
  }
  $i = 10;
  funct(++$i, $i = $i * 2, --$i);
?>
```

Результатом выполнения скрипта из листинга 7.5 будет такая последовательность чисел:

```
11
22
21
```

7.3. Передача параметров по значению и ссылке

В рассмотренных примерах аргументы функции передаются *по значению*, т. е. значения параметров изменяются только внутри функции, и эти изменения не влияют на значения переменных за пределами функции (листинг 7.8).

Листинг 7.8. Передача аргумента по значению

```
<?php
  function get_sum($var) // аргумент передается по значению
  {
    $var = $var + 5;
    return $var;
  }
  $new_var = 20;
  echo get_sum($new_var); // 25
  echo "<br>$new_var";    // 20
?>
```

Для того чтобы переменные, переданные функции, сохраняли свое значение при выходе из нее, применяется передача параметров по ссылке. Для этого перед именем переменной необходимо поместить амперсанд (`&`):

```
function get_sum(&$var)
```

В этом случае переменная `$var` будет передана *по ссылке*. В случае, если аргумент передается по ссылке, при любом изменении значения параметра происходит изменение переменной-аргумента (листинг 7.9).

> ***Замечание***
>
> Объекты и массивы передаются в функцию по ссылке, поэтому применительно к таким параметрам можно не использовать символ амперсанда `&`.

Листинг 7.9. Передача аргумента по ссылке

```
<?php
  function get_sum(&$var) // аргумент передается по ссылке
  {
    $var = $var + 5;
    return $var;
  }
  $new_var = 20;
  echo get_sum($new_var); // выводит 25
  echo "<br>$new_var";    // выводит 25
?>
```

7.4. Необязательные параметры

Параметры можно объявлять как необязательные. Для этого при объявлении параметра необходимо присвоить ему значение по умолчанию (листинг 7.10).

Листинг 7.10. Объявление необязательных параметров

```
<?php
  function get_sum($left = 10, $right = 5)
  {
    $sum = $left + $right;
    echo $sum;
  }
  get_sum();     // выводит 15
  get_sum(5);    // выводит 10
  get_sum(5, 0); // выводит 5
?>
```

Как видно из листинга 7.10, даже если функции `get_sum()` не передаются параметры, она успешно производит вычисления с участием параметров по умолчанию.

Если функция содержит множество обязательных и необязательных параметров, то все обязательные параметры следует располагать до необязательных. В листинге 7.11 приводится некорректное объявление функции `get_sum()`, в котором необязательный параметр предшествует обязательному.

Листинг 7.11. Ошибочное объявление функции

```
<?php
  function get_sum($left = 10, $right)
  {
```

```
    $sum = $left + $right;
    echo $sum;
  }
  get_sum(5);
?>
```

Впрочем, попытка выполнения скрипта из листинга 7.11 не заканчивается прекращением — PHP-интерпретатор ограничивается выводом замечания. Единственный параметр 5 подставляется вместо `$left`, параметр `$right` получает неопределенное значение, которое при сложении приводится к нулю (рис. 7.2).

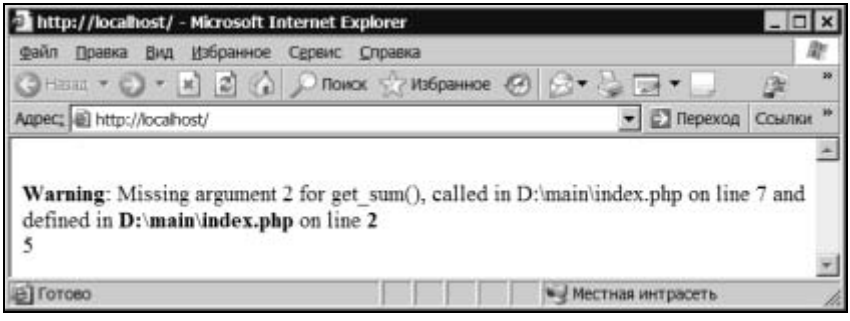


Рис. 7.2. Вместо двух параметров функции передан лишь один

7.5. Переменное количество параметров

С-подобные языки программирования обычно поддерживают синтаксис функций с переменным количеством параметров. В PHP для этого необходимо объявить функцию без параметров — такой функции можно передавать любое их количество, без каких-либо последствий (листинг 7.12).

Листинг 7.12. Функции без параметров могут принимать любое их количество

```
<?php
  function get_sum()
  {
    echo "Вызов функции";
  }
  get_sum(5, "Второй параметр", 124);
?>
```

Выполнив скрипт из листинга 7.12, можно убедиться, что интерпретатор PHP не генерирует ни замечаний, ни предупреждений (в противовес ситуации, когда функция имеет хотя бы один параметр). Для работы с переменным количеством параметров в PHP предусмотрены специальные функции, перечисленные в табл. 7.1.

Замечание

Функции из табл. 7.1 можно применять ко всем функциям без исключения, а не только к функциям без параметров.

Таблица 7.1. *Функции обработки переменного количества параметров*

Функция	Описание
`func_num_args()`	Возвращает количество параметров, переданных функции
`func_get_args()`	Возвращает массив с параметрами, переданными функции
`func_get_arg($arg_num)`	Возвращает значение параметра с номером `$arg_num`. Если параметр с таким номером не существует, функция возвращает `FALSE`

В листинге 7.13 приводится пример функции, которая выводит значения всех своих параметров.

Листинг 7.13. Вывод функцией собственных параметров

```
<?php
  // Объявляем функцию
  function get_parameters()
  {
     for ($i = 0; $i < func_num_args(); $i++)
     {
        echo "Параметр номер $i: ".func_get_arg($i)."<br>";
     }
  }

  // Вызываем функцию
  echo get_parameters("Hello,", "world", "!", 123456);
?>
```

Результатом выполнения скрипта из листинга 7.13 будут следующие строки:

```
Параметр номер 0: Hello,
Параметр номер 1: world
Параметр номер 2: !
Параметр номер 3: 123456
```

Решить задачу, которую решает скрипт из листинга 7.13, можно лишь при помощи одной функции `func_get_args()` (листинг 7.14).

Листинг 7.14. Использование функции `func_get_args()`

```
<?php
  // Объявляем функцию
  function get_parameters()
  {
    $arr = func_get_args();
    for ($i = 0; $i < count($arr); $i++)
    {
      echo "Параметр номер $i: {$arr[$i]}<br>";
    }
  }

  // Вызываем функцию
  echo get_parameters("Hello,", "world", "!", 123456);
?>
```

7.6. Глобальные переменные

Переменные в функциях имеют локальную область видимости. Это означает, что даже если локальная (внутри функции) и внешняя (вне функции) переменные имеют одинаковые имена, то изменение локальной переменной никак не повлияет на внешнюю переменную (листинг 7.15).

Листинг 7.15. Внешняя переменная не влияет на локальную и наоборот

```
<?php
  function get_sum()
  {
    $var = 5;       // локальная переменная
    echo $var;
  }
  $var = 10;        // внешняя переменная
  get_sum();        // выводит  5 (локальная переменная)
  echo "<br>$var"; // выводит 10 (внешняя переменная)
?>
```

Локальную переменную можно сделать внешней, если перед ее именем указать ключевое слово `global`. В этом случае изменения как внутри функции, так и вне ее будут влиять на переменную, а сама переменная будет называться *глобальной*. Если локальная переменная объявлена как `global`, то к ней возможен доступ из любой части программы (листинг 7.16).

Замечание

Использовать глобальные переменные следует только в случае острой необходимости. Злоупотребление ими ведет к созданию запутанного, сложно сопровождаемого кода.

Листинг 7.16. Использование глобальной переменной

```
<?php
  function get_sum()
  {
    global $var;
    $var = 5; // изменяем глобальную переменную
    echo $var;
  }
  $var = 10;
  echo "$var<br>"; // выводит 10
  get_sum();       // выводит 5(глобальная переменная изменена)
  echo "$var<br>"; // выводит 5
?>
```

Доступ к глобальным переменным можно получить также через суперглобальный массив `$GLOBALS` (листинг 7.17).

Листинг 7.17. Использование суперглобального массива `$GLOBALS`

```
<?php
  function get_sum()
  {
    $GLOBALS['var'] = 20; // изменяем глобальную переменную $var
    echo($GLOBALS['var']);
  }
  $var = 10;
  echo "$var<br>"; // выводит 10
  get_sum();       // выводит 20 (глобальная переменная изменена)
?>
```

Массив `$GLOBALS` доступен в области видимости любой функции и содержит все глобальные переменные, которые используются в программе.

7.7. Статические переменные

Временем жизни переменной называется интервал выполнения программы, в течение которого она существует. Поскольку локальные переменные имеют своей областью видимости функцию, то время жизни локальной переменной определяется временем выполнения функции, в которой она объявлена. Это означает, что в разных функциях

совершенно независимо друг от друга могут использоваться переменные с одинаковыми именами. Локальная переменная при каждом вызове функции инициализируется заново, поэтому функция-счетчик, пример которой показан в листинге 7.18, всегда будет возвращать значение 1.

Листинг 7.18. Пример неправильной организации функции-счетчика

```
<?php
  function counter()
  {
    $counter = 0;
    return ++$counter;
  }
?>
```

Для того чтобы локальная переменная сохраняла свое предыдущее значение при новых вызовах функции, ее можно объявить статической при помощи ключевого слова `static` (листинг 7.19).

Листинг 7.19. Организации функции-счетчика с помощью статической переменной

```
<?php
  function counter()
  {
    static $counter = 0;
    return ++$counter;
  }
?>
```

В скрипте из листинга 7.19 `$counter` устанавливается в ноль при первом вызове функции и при последующих вызовах функции помнит, каким было значение переменной при предыдущих вызовах.

Временем жизни статических и глобальных переменных является время выполнения сценария. То есть если пользователь перезагружает страницу, что приводит к новому выполнению сценария, переменная `$counter` инициализируется заново.

7.8. Возврат массива функцией

Функция может возвращать массив в качестве значения, для этого достаточно передать его в качестве параметра оператору `return`. Более того, такой массив может создаваться динамически при помощи конструкции `array()`. К такому приему прибегают всякий раз, когда функция должна вернуть несколько значений, а передача значений по ссылке не допускается.

В листинге 7.20 демонстрируется функция `format_size()`, которая принимает в качестве значения размер файла в байтах и возвращает массив, первый элемент которого

содержит размер в байтах, второй — в килобайтах, третий — в мегабайтах, а четвертый — в гигабайтах.

Листинг 7.20. Функция возвращает массив

```
<?php
  function format_size($byte)
  {
    $kbyte = $byte / 1024;
    $mbyte = $kbyte / 1024;
    $gbyte = $mbyte / 1024;

    return array($byte, $kbyte, $mbyte, $gbyte);
  }
?>
```

Оперировать массивом в скрипте не всегда удобно, особенно, если он имеет постоянное небольшое количество элементов. Поэтому часто при вызове функции массив сразу же разбивается на переменные при помощи конструкции `list()` (листинг 7.21).

Листинг 7.21. Преобразование массива в переменные

```
<?php
  ...
  list($byte, $kbyte, $mbyte, $gbyte) = format_size(18642678);
?>
```

7.9. Рекурсивные функции

Рекурсия — это вызов функцией самой себя, пример рекурсивной функции приведен в листинге 7.22.

Листинг 7.22. Рекурсивная функция `callself()`

```
<?php
  function callself($counter)
  {
    // Если параметр $counter больше, продолжаем рекурсивный спуск
    if ($counter>0)
    {
      // Уменьшаем значение параметра $counter и выводим его значение
      // в окно браузера
      echo($counter--)."<br>";
```

```
      // Осуществляем рекурсивный вызов функции callself()
      callself($counter);
    }
    // Если значение параметра меньше или равно 0, прекращаем
    // рекурсивный спуск
    else return;
  }
  // Вызываем функцию callself()
  callself(4);
?>
```

Результатом работы функции будет последовательность цифр:

```
4
3
2
1
```

Функция `callself()` вызывает саму себя до тех пор, пока ее параметр `$counter` положителен и не равен нулю. Рекурсивных функций по возможности стараются избегать, т. к. они относятся к трудным по восприятию конструкциям языка, и отладка их достаточно сложна, особенно, когда приходится иметь дело не с простейшей рекурсивной функцией, представленной в листинге 6.22, а со сложной функцией, осуществляющей рекурсивный вызов в нескольких местах функции.

> ***ЗАМЕЧАНИЕ***
>
> Опасность использования неотлаженных рекурсивных функций заключается в возможности перехода их в режим бесконечной рекурсии, когда условие, прекращающее спуск вниз по рекурсии из-за ошибки, не наступает, в результате чего, как и в случае бесконечных циклов, наступает зависание программы.

Практически в любом случае можно избежать рекурсивных функций. Исключение составляют задачи, так или иначе связанные с обходом деревьев. К таким задачам относится, например, удаление каталогов *(см. главу 13)*, когда число файлов и подкаталогов заранее не известно и необходимо вызывать функцию удаления до тех пор, пока не будут удаленны файлы на самом глубоком уровне вложенности. К таким задачам относится и подсчет вложенных сумм, когда число вложений заранее не известно: $\Sigma\Sigma\Sigma...\Sigma\Sigma\, x_i$, а так же любая другая задача, являющаяся аналогом бинарного дерева.

7.10. Вложенные функции

Язык программирования PHP позволяет объявлять функции внутри другой функции. В отличие от обычных функций, вложенная функция не может использоваться до тех пор, пока не будет осуществлен вызов основной функции, который произведет объявление вложенной (листинг 7.23).

Листинг 7.23. Объявление вложенной функции

```
<?php
  // Объявление внешней и вложенной функций
  function outter()
  {
    function inner()
    {
      echo "Hello, world!";
    }
  }

  // Вызываем функцию outter(), чтобы объявить функцию inner()
  outter();

  // Функция inner() не может быть вызвана до тех пор,
  // пока не будет вызвана функция outter();
  inner();
?>
```

7.11. Динамическое имя функции

По аналогии с переменными, имя функции может быть динамическим и храниться в строковой переменной — передача такой переменной оператором круглых скобок (с параметрами, если они требуются) приводит к вызову функции. В листинге 7.24 демонстрируется вызов функций `hello()` и `bye()` по случайному закону.

Листинг 7.24. Использование динамических имен функций

```
<?php
  // Объявление функций
  function hello()
  {
    echo "Hello!";
  }
  function bye()
  {
    echo "Bye!";
  }

  // Случайный выбор функции
  if (rand(0, 1)) $var = "hello";
```

```
  else $var = "bye";

  // Вызов функции
  $var();
?>
```

7.12. Анонимные функции

Динамическими могут быть не только имена функций *(см. разд. 7.9)*, но и сами функции. Таким функциям, как правило, присваивается уникальное имя в процессе их динамического создания — при помощи этого имени их можно вызвать либо так, как это описывалось в предыдущем разделе, либо при помощи специализированных функций, представленных в табл. 7.2.

***Таблица 7.2.** Функции для создания и обслуживания динамических функций*

Функция	Описание
`create_function($args, $code)`	Создает анонимную функцию с аргументами `$args` и телом `$code`
`call_user_func_array($function [, $param_arr])`	Вызывает функцию с именем `$function` с параметрами, в качестве которых выступают элементы массива `$param_arr`. Возвращает либо результат выполнения функции `$function`, либо `FALSE`, если функцию выполнить не удалось
`call_user_func($function [, $parameter [, ...]] )`	Вызывает функцию с именем `$function` с параметрами, перечисленными в последующих параметрах функции. Возвращает либо результат выполнения функции `$function`, либо `FALSE`, если функцию выполнить не удалось

В листинге 7.25 приводится пример создания анонимной функции, принимающей два числовых параметра и возвращающей в качестве результата их сумму.

> ***Замечание***
>
> Тело функции должно быть синтаксически верным, в противном случае генерируется предупреждение.

Листинг 7.25. Создание анонимной функции

```
<?php
  // Создание функции
  $name = create_function('$fst, $snd', 'return $fst + $snd;');
  // Выводим динамическое имя функции
  echo "Имя функции - $name <br>";
  // Вызываем функцию
  echo $name(6, 10); // 16
?>
```

В качестве результата вызова скрипта из листинга 7.25 в окно браузера будут выведены следующие строки:

```
Имя функции - lambda_1
16
```

Интересно отметить, что при повторном вызове скрипта счетчик после имени `lambda` будет увеличиваться: lambda_1, lambda_2, lambda_3, ...

Вызвать функцию можно не только при помощи имени, но и воспользовавшись специализированными функциями `call_user_func_array()` и `call_user_func()` из табл. 7.2 (листинг 7.26).

Листинг 7.26. Использование функций `call_user_func_array()` и `call_user_func()`

```
<?php
  // Создание функции
  $name = create_function('$fst, $snd', 'return $fst + $snd;');
  // Вызываем функцию
  echo call_user_func($name, 6, 10); // 16
  echo "<br>";
  echo call_user_func_array($name, array(6, 10)); // 16
?>
```

Следует отметить, что при помощи функций `call_user_func_array()` и `call_user_func()` можно вызывать не только анонимные функции, но и любые другие функции PHP, в том числе и встроенные (листинг 7.27).

> ***Замечание***
>
> Конструкции языка программирования PHP, такие как `echo`, `include` и т. п., не являются функциями, поэтому их имена не могут передаваться в качестве параметров функций `call_user_func_array()` и `call_user_func()`.

Листинг 7.27. Вызов функции `print_r()` при помощи `call_user_func()`

```
<?php
  // Объявляем массив
  $arr = array("Hello", "world", "!");
  // Выводим дамп массива
  call_user_func("print_r", $arr);
?>
```

Результатом выполнения скрипта из листинга 7.27 будет следующая строка кода:

```
Array( [0] => Hello [1] => world [2] => ! )
```

7.13. Проверка существования функции

Как видно из *разд. 7.1*, *7.10—7.12*, функция с заданным или динамическим именем может быть доступна для вызова или нет. В связи с этим одной из важных задач является проверка существования функции. Эту задачу выполняют функции, представленные в табл. 7.3.

> ***Замечание***
>
> Помимо пользовательских функций, существует большое количество функций, определенных в расширениях. Проверка существования таких функций позволяет корректно обрабатывать ситуацию, когда то или иное расширение не подключено к ядру PHP.

Таблица 7.3. *Проверка существования функции*

Функция	Описание
`function_exists($function_name)`	Возвращает `TRUE`, если функция с именем `$function_name` существует, в противном случае возвращается `FALSE`
`get_defined_functions()`	Возвращает массив всех доступных в настоящий момент функций
`get_extension_funcs($module_name)`	Возвращает массив с именами функций для расширения `$module_name`
`extension_loaded($name)`	Проверяет, подключено ли расширение с именем `$name`
`get_loaded_extensions ([$zend_extensions = FALSE])`	Возвращает массив подключенных расширений

Проверить существование одиночной функции проще всего при помощи функции `function_exists()` (листинг 7.28).

Листинг 7.28. Использование функции `function_exists()`

```
<?php
  if (TRUE)
  {
    function get_sum_if($fst, $snd)
    {
      return $fst + $snd;
    }
  }

  if (function_exists("get_sum")) // TRUE
    echo "Функция get_sum() существует<br>";
```

```
  else
    echo "Функция get_sum() не существует<br>";

  if (function_exists("get_sum_if")) // TRUE
    echo "Функция get_sum_if () существует<br>";
  else
    echo "Функция get_sum_if () не существует<br>";

  if (function_exists("get_sum_if_post")) // FALSE
    echo "Функция get_sum_if_post() существует<br>";
  else
    echo "Функция get_sum_if_post() не существует<br>";

  function get_sum($fst, $snd)
  {
    return $fst + $snd;
  }
  if (TRUE)
  {
    function get_sum_if_post($fst, $snd)
    {
      return $fst + $snd;
    }
  }
?>
```

Результатом выполнения скрипта из листинга 7.28 будут следующие строки:

```
Функция get_sum() существует
Функция get_sum_if () существует
Функция get_sum_if_post() не существует
```

При помощи функции `get_defined_functions()` можно получить массив всех функций, доступных на текущий момент (листинг 7.29).

Листинг 7.29. Использование функции `get_defined_functions()`

```
<?php
  function get_sum($fst, $snd)
  {
    return $fst + $snd;
  }

  $arr = get_defined_functions();
```

```
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Функция `get_defined_functions()` возвращает двумерный ассоциативный массив, первый элемент `internal` содержит предопределенные функции из ядра PHP и расширений, второй элемент `user` содержит функции, определенные пользователем:

```
Array
(
    [internal] => Array
       (
            [0] => zend_version
            [1] => func_num_args
            [2] => func_get_arg
            [3] => func_get_args
            [4] => strlen
            [5] => strcmp
            [6] => strncmp
            ...
            [1309] => socket_getopt
            [1310] => socket_setopt
        )

    [user] => Array
       (
            [0] => get_sum
        )
)
```

Функция `get_extension_funcs()` позволяет вернуть массив с именами функций, принадлежащих расширению, имя которого передается в качестве параметра. В листинге 7.30 выводится список функций из расширения mysql, позволяющего работать с базой данных MySQL.

> ***Замечание***
>
> Функции для работы с MySQL более подробно обсуждаются в *главе 28*.

Листинг 7.30. Использование функции `get_extension_funcs()`

```
<?php
  // Получаем массив с именами функций для работы с MySQL
  $arr = get_extension_funcs("mysql");
```

```
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 7.30 будет следующий дамп массива `$arr`:

```
Array
(
    [0] => mysql_connect
    [1] => mysql_pconnect
    [2] => mysql_close
    [3] => mysql_select_db
    ...
    [59] => mysql_tablename
    [60] => mysql_table_name
)
```

Для того чтобы функция `get_extension_funcs()` вернула результат, соответствующее расширение должно быть подключено в конфигурационном файле php.ini *(см. главу 20)*. Если расширение не подключено, функция вернет `FALSE`. Для проверки факта, подключено расширение или нет, предназначена специальная функция `extension_loaded()`, которая принимает в качестве параметра имя расширения и возвращает `TRUE`, если оно подключено, и `FALSE` в противном случае (листинг 7.31).

Листинг 7.31. Использование функции `extension_loaded()`

```
<?php
  if (extension_loaded("mysql"))
    echo "Расширение для работы с MySQL подключено";
  else
    echo "Расширение для работы с MySQL не подключено";
?>
```

При помощи функции `get_loaded_extensions()` можно получить список всех подключенных расширений (листинг 7.32).

Листинг 7.32. Использование функции `get_loaded_extensions()`

```
<?php
  // Получаем список всех подключенных расширений
  $arr = get_loaded_extensions();

  echo "<pre>";
```

```
  print_r($arr);
  echo "</pre>";
?>
```

При условии, что в конфигурационном файле php.ini будут подключены внешние расширения mbstring, mcrypt и mysql, результат выполнения скрипта из листинга 7.32 может выглядеть следующим образом:

```
Array
(
    [0] => bcmath
    [1] => calendar
    [2] => com_dotnet
    [3] => ctype
    [4] => session
    [5] => ereg
    [6] => filter
    [7] => ftp
    [8] => hash
    [9] => iconv
    [10] => json
    [11] => mysqlnd
    [12] => odbc
    [13] => pcre
    [14] => Reflection
    [15] => date
    [16] => libxml
    [17] => standard
    [18] => tokenizer
    [19] => unicode
    [20] => zlib
    [21] => SimpleXML
    [22] => dom
    [23] => SPL
    [24] => wddx
    [25] => xml
    [26] => xmlreader
    [27] => xmlwriter
    [28] => apache2handler
    [29] => mbstring
    [30] => mcrypt
    [31] => mysql
)
```

Как видно из результатов, помимо трех внешних расширений mbstring, mcrypt и mysql функция `get_loaded_extensions()` возвращает еще 29 расширений. Это связно с тем, что часть расширений компилируется непосредственно в составе дистрибутива.

> ***Замечание***
>
> Существует большое количество расширений, предоставляющих разработчику широкий набор функций самой разной направленности. Часть из этих расширений доступна в составе дистрибутива, часть необходимо загружать отдельно в составе PECL-архива.

7.14. Неявное выполнение функций. Оператор *declare()*

До этого момента мы сталкивались либо с прямым вызовом функции, либо при помощи специализированных функций вроде `call_user_func_array()` и `call_user_func()`. PHP предоставляет разработчикам возможность неявного, условного вызова функций посредством директивы `declare`, которая имеет следующий синтаксис:

```
declare(directive)
{
    ...
}
```

Директива `directive` позволяет задать условие выполнения блока кода внутри фигурных скобок (как и в случае остальных конструкций, если блок состоит из одного оператора, фигурные скобки не обязательны). В настоящий момент поддерживается только одна директива `ticks`, которая позволяет задать количество строк кода, через которые выполняется специально зарегистрированная функция. В табл. 7.4 приводятся функции, обслуживающие неявное выполнение функций.

Таблица 7.4. *Функции неявного выполнения функций*

Функция	Описание
`register_shutdown_function ($function [, $parameter [, ...]] )`	Регистрирует функцию `$function`, которая выполняется по завершению работы скрипта
`register_tick_function ($function [, $arg [, ...]] )`	Регистрирует функцию `$function` для выполнения ticks-условия в конструкции `declare`
`unregister_tick_function ($function)`	Удаляет функцию `$function` из списка выполнения ticks-условия в конструкции `declare`

В листинге 7.33 приводится пример использования конструкции `declare`. Пользовательская функция `line()` регистрируется при помощи функции `register_tick_function()`, в результате чего она выполняется каждые 10 строк в конструкции `declare` (строки с фигурными скобками не считаются).

ЗАМЕЧАНИЕ

Функцию `register_tick_function()` не следует использовать в многопоточных Web-серверах, т. к. она корректно работает только в том случае, если под обслуживание запроса выделен отдельный процесс. В большинстве случаев это означает, что скрипт из листинга 7.33 будет успешно работать под управлением UNIX-подобных операционных систем и приведет к краху сервера под Windows.

Листинг 7.33. Использование конструкции `declare` (вариант 1)

```
<?php
  // Объявляем функцию
  function line()
  {
    echo "Line<br>";
  }

  // Регистрируем функцию для выполнения в условии declare
  register_tick_function("line");

  // Вызываем функцию line() каждые 10 строчек кода
  declare(ticks=10)
  {
    for($i = 0; $i < 10; ++$i)
    {
      echo($i * 2)."<br>";
    }
  }

  // Удаляем функцию line() из списка зарегистрированных функций
  unregister_tick_function("line");
?>
```

ЗАМЕЧАНИЕ

В листинге 7.33 в качестве примера приводится функция `line()`, которая не принимает параметров. Если же пользовательская функция принимает параметры, они могут быть переданы функции `register_tick_function()` в качестве второго, третьего и последующих параметров.

Результатом выполнения скрипта из листинга 7.33 будут следующие строки:

```
0
2
4
6
8
```

```
Line
10
12
14
16
18
Line
```

Как видно из результатов, функция `line()` вызывается каждые 10 строк. Одна итерация содержит вызов оператора `for` и конструкции `echo` — получается две строки за цикл. В листинге 7.34 количество строк в цикле увеличено до трех.

Листинг 7.34. Использование конструкции `declare` (вариант 2)

```
<?php
  // Объявляем функцию
  function line()
  {
    echo "Line<br>";
  }

  // Регистрируем функцию для выполнения в условии declare
  register_tick_function("line");

  // Вызываем функцию line() каждые 10 строчек кода
  declare(ticks=10)
  {
    for($i = 0; $i < 10; ++$i)
    {
      echo "";
      echo($i * 3)."<br>";
    }
  }

  // Удаляем функцию line() из списка зарегистрированных функций
  unregister_tick_function("line");
?>
```

Результатом выполнения скрипта из листинга 7.34 будут следующие строки:

```
0
3
6
```

```
Line
9
12
15
18
Line
21
24
27
Line
```

При помощи функции `register_tick_function()` можно зарегистрировать несколько функций — они будут выполнены одна за другой при достижении ticks-условия. Более того, одна и та же функция может быть зарегистрирована несколько раз (листинг 7.35).

Листинг 7.35. Можно зарегистрировать несколько функций

```
<?php
  function line()
  {
    echo "Line<br>";
  }

  register_tick_function("line");
  register_tick_function("line");

  declare(ticks=10)
  {
    for($i = 0; $i < 10; ++$i)
    {
      echo "";
      echo($i*3)."<br>";
    }
  }
?>
```

Результатом выполнения скрипта из листинга 7.35 будут следующие строки:

```
0
3
6
Line
```

```
Line
9
12
15
18
Line
Line
21
24
27
Line
Line
```

Помимо конструкции `declare()` осуществить неявный вызов функции можно при помощи функции `register_shutdown_function()`, которая позволяет зарегистрировать функцию, выполняющуюся после завершения скрипта. Точно так же как и функция `register_tick_function()`, функция `register_shutdown_function()` может вызываться несколько раз, чтобы зарегистрировать несколько функций или одну функцию несколько раз (листинг 7.36).

Листинг 7.36. Использование функции `register_shutdown_function()`

```
<?php
  // Объявляем функцию
  function line()
  {
    echo "Line<br>";
  }

  register_shutdown_function("line");
  register_shutdown_function("line");

  echo "...<br>";
  echo "...<br>";
  echo "...<br>";
?>
```

Результатом выполнения скрипта из листинга 7.36 будут следующие строки:

```
...
...
...
Line
Line
```

7.15. Вспомогательные функции

Ранее мы уже использовали различные встроенные функции и конструкции языка PHP для управления ходом выполнения программы. В табл. 7.5 приведены наиболее часто используемые из них.

Таблица 7.5. *Вспомогательные функции PHP*

Функция	Описание
`exit([$status])`	Прекращает выполнение скрипта. Если указан необязательный параметр `$status`, он выводится в окно браузера
`die()`	Синоним для функции `exit()`
`sleep($seconds)`	Осуществляет задержку выполнения скрипта на `$seconds` секунд
`time_nanosleep($seconds, $nanoseconds)`	Осуществляет задержку выполнения скрипта на `$seconds` секунд и `$nanoseconds` наносекунд
`usleep($micro_seconds)`	Осуществляет задержку выполнения скрипта на `$micro_seconds` микросекунд
`eval($code)`	Выполняет PHP-код из строки `$code` *(см. разд. 3.16)*
`empty($var)`	Определяет, не пуста ли переменная `$var`, возвращая `TRUE`, если переменная пустая, и `FALSE` в противном случае *(см. разд. 3.12)*
`isset($var)`	Определяет, существует ли переменная `$var`, возвращая `TRUE`, если переменная существует, и `FALSE` в противном случае *(см. разд. 3.12)*
`unset($var [, $var1 [, ...]])`	Уничтожает одну или более переменных, массивов, объектов, переданных конструкции в качестве параметров *(см. разд. 3.11)*
`print_r($var [, $return])`	Выводит дамп переменной, массива или объекта `$var` *(см. разд. 6.1)*. Если необязательный параметр `$return` принимает значение `TRUE`, вместо вывода дампа в окно браузера, он возвращается в виде строки
`var_dump($var [, $var1 [, ...]])`	Выводит дамп одной и большего количества переменных, включая их типы данных
`phpinfo()`	Выводит таблицы с параметрами текущей версии PHP, расширений и переменных окружений
`phpversion([$extension])`	Возвращает текущую версию PHP или версию расширения, если необязательный параметр `$extension` содержит его имя
`php_uname([$mode])`	Возвращает сведения об операционной системе, в которой выполняется PHP. Необязательный параметр `$mode` может принимать значения параметров для одноименной утилиты UNIX-подобных операционных систем
`getenv($varname)`	Возвращает значение переменной окружения с именем `$varname`. Имена типичных переменных окружений можно обнаружить в *разд. 6.20*

В любой момент работа скрипта может быть прекращена при помощи функции `exit()` (листинг 7.37).

Листинг 7.37. Использование функции `exit()`

```
<?php
  echo "Фраза будет выведена";
  // Останавливаем работу скрипта
  exit();
  echo "Эта фраза выведена не будет";
?>
```

Многие особенности языка программирования PHP зависят от версии, поэтому в сложных программных проектах может быть полезно использовать ту или иную реализацию в зависимости от текущей версии PHP. Версию всегда можно получить при помощи функции `phpversion()` (листинг 7.38).

Листинг 7.38. Использование функции `phpversion()`

```
<?php
  echo phpversion(); // 6.0.0
?>
```

Помимо версии PHP, на функциональности скрипта может сказываться версия и особенности операционной системы. В *разд. 7.14* уже упоминалось о том, что скрипты могут по-разному вести себя в зависимости от текущей операционной системы, в следующих главах еще не раз будет демонстрироваться, как операционная система может влиять на функциональность Web-приложений. Для определения текущей версии операционной системы предназначена функция `php_uname()` (листинг 7.39).

Листинг 7.39. Использование функции `php_uname()`

```
<?php
  echo php_uname();
?>
```

Скрипт из листинга 7.39 в операционной системе Windows может вернуть строку вида (Windows XP):

```
Windows NT PHOTON 5.1 build 2600
```

В UNIX-подобной операционной системе функция возвращает результат, который она получает от утилиты `uname`, например:

```
Linux localhost.localdomain 2.6.21-1.3194.fc7 #1 SMP Wed May 23 22:35:01 EDT
2007 i686
```

Функция `php_uname()` может принимать необязательный параметр `$mode` с форматирующей строкой. Ряд символов в такой строке имеет специальное значение:

- `'a'` — полный вывод (используется по умолчанию), аналогично вызову функции `php_uname` со строкой `"s n r v m"`;
- `'s'` — операционная система ("Windows NT", "Linux");
- `'n'` — имя хоста ("PHOTON", " localhost.localdomain");
- `'r'` — номер релиза ("5.1", "2.6.21-1.3194.fc7");
- `'v'` — версия операционной системы ("build 2600", "#1 SMP Wed May 23 22:35:01 EDT 2007");
- `'m'` — оптимизация под процессор ("i586", "i686").



Глава 8

Взаимодействие PHP с HTML

PHP выполняется на сервере, а HTML интерпретируется браузером на стороне клиента *(см. главу 1)*. При работе с Web-приложением посетитель переходит по ссылкам, заполняет HTML-формы, что вызывает выполнение PHP-скриптов. Данная глава посвящена тому, как PHP взаимодействует с HTML. В данной главе будут рассмотрены два метода протокола HTTP: GET — передача параметров в строке запроса, и POST — передача параметров в теле HTTP-документа.

8.1. Передача параметров методом *GET*

GET-параметры передаются в строке запроса после символа вопроса (`?`).

http://www.softtime.ru/forum/read.php?id_forum=1

В приведенном выше URL последовательность `id_forum=1` задает GET-параметр с именем `id_forum` и значением 1. GET-параметры автоматически помещаются в суперглобальный массив `$_GET`. Имена параметров выступают в качестве ключей массива. В листинге 8.1 выводится значение GET-параметра `id_forum`.

> ***Замечание***
>
> Стандарт HTTP, в отличие от PHP, допускает использование в качестве имени GET-параметра последовательности, начинающееся с цифры или содержащей тире. Такие параметры не попадают в суперглобальный массив `$_GET`, если отказаться от их использования нет никакой возможности, их значения следует самостоятельно извлекать из переменной окружения `$_SERVER['QUERY_STRING']`, содержащей GET-параметры.

Листинг 8.1. Извлечение GET-параметра `id_forum`

```
<?php
  echo $_GET['id_forum'];
?>
```

Если имеется необходимость передать скрипту одновременно несколько GET-параметров, они разделяются символом амперсанда `&`:

http://www.softtime.ru/forum/read.php?id_forum=1&id_theme=2&id_post=10

В приведенном ранее URL передаются три GET-параметра: `id_forum` со значением 1, `id_theme` со значением 2 и `id_post` со значением 10 (листинг 8.2).

Листинг 8.2. Вывод дампа суперглобального массива `$_GET`

```
<?php
  echo "<pre>";
  print_r($_GET);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 8.2 может быть выведен следующий дамп массива `$_GET`:

```
Array
(
    [id_forum] => 1
    [id_theme] => 2
    [id_post] => 10
)
```

В качестве GET-параметров могут выступать элементы массива, в этом случае суперглобальный массив `$_GET` становится двумерным. Так, для строки запроса:

http://www.softtime.ru/forum/read.php?id[]=1&id[]=2&id[]=10

скрипт из листинга 8.2 выведет следующий дамп:

```
Array
(
    [id] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 10
        )
)
```

Здесь элементам были автоматически назначены числовые индексы, начиная с 0, однако индексы и ключи могут назначаться элементам непосредственно в строке запроса. Для запроса

http://www.softtime.ru/forum/read.php?id[a]=1&id[b]=2&id[c]=10

скрипт из листинга 8.2 выведет следующий дамп:

```
Array
(
    [id] => Array
        (
```

```
            [a] => 1
            [b] => 2
            [c] => 10
        )
)
```

GET-параметры и их значения могут содержать недопустимые с точки зрения URL символы (пробелы, русские символы), которые при формировании URL обязательно должны преобразовываться в безопасный формат. Для такого преобразования в PHP предусмотрены специальные функции (табл. 8.1).

Таблица 8.1. *Функции для работы с URL*

Функция	Описание
`urlencode($str)`	Возвращает строку `$str`, в которой все символы, отличные от чисел, букв английского алфавита и символа подчеркивания, кодируются двумя шестнадцатеричными числами, предваренными символом `%`. В отличие от всех остальных символов пробел кодируется плюсом `+`
`urldecode($str)`	Обратная функция `urlencode()`, осуществляющая декодирование символов, начинающихся с `%`
`rawurlencode($str)`	Возвращает строку `$str`, в которой все символы, отличные от чисел, букв английского алфавита и символа подчеркивания, кодируются двумя шестнадцатеричными числами, предваренными символом `%`. Функция аналогична `urlencode()`, однако пробел кодируется не символом плюса `+`, а последовательностью `%20`
`rawurldecode($str)`	Обратная `rawurlencode()` функция, осуществляющая декодирование символов, начинающихся с `%`
`parse_url($url [, $component])`	Разбирает строку запроса `$url` на отдельные компоненты, которые возвращаются как элементы ассоциативного массива. Если необязательный параметр `$component` принимает одну из следующих констант: `PHP_URL_SCHEME`, `PHP_URL_HOST`, `PHP_URL_PORT`, `PHP_URL_USER`, `PHP_URL_PASS`, `PHP_URL_PATH`, `PHP_URL_QUERY` или `PHP_URL_FRAGMENT`, то один из компонентов возвращается в виде строки
`http_build_query ($formdata [, $numeric_prefix [, $arg_separator]])`	Позволяет сформировать строку запроса из элементов массива `$formdata`. Если указан необязательный параметр `$numeric_prefix`, он добавляется в качестве префикса GET-параметрам. Необязательный параметр `$arg_separator` позволяет задать разделитель между GET-параметрами, отличный от амперсанда `&`

В листинге 8.3 формируется ссылка на файл test.php, через GET-параметр `phrase` которому передается фраза "Привет, мир!".

Листинг 8.3. Использование функции `urlencode()`

```
<?php
  echo "<a href='test.php?phrase=".
           urlencode("Привет, мир!")."'>ссылка</a>";
?>
```

Исходный код результирующей HTML-страницы будет содержать следующую строку

```
<a href='test.php?phrase=%CF%F0%E8%E2%E5%F2%2C+%EC%E8%F0%21'>ссылка</a>
```

Помимо GET-параметров в скрипте может понадобиться информация о текущей странице, номере порта, хосте и т. п. Для разбора строки запроса предназначена специальная функция `parse_url()`, которая принимает в качестве первого параметра строку запроса и возвращает отдельные его компоненты в виде ассоциативного массива со следующими ключами:

- `scheme` — префикс, например, http, https, ftp и т. п.;
- `host` — домен;
- `port` — номер порта;
- `user` — пользователь;
- `pass` — его пароль;
- `path` — путь от корневого каталога;
- `query` — все, что расположено после символа вопроса (`?`);
- `fragment` — все, что расположено после символа `#`.

В листинге 8.4 приводится пример разбора URL при помощи функции `parse_url()`.

Листинг 8.4. Использование функции `parse_url()`

```
<?php
  $url = 'http://user:pass@www.site.ru/path/index.php?par=value#anch';
  $arr = parse_url($url);

  echo "<pre>";
  print_r($arr);
  echo "<pre>";
?>
```

Результатом выполнения скрипта из листинга 8.4 будет следующий дамп массива `$arr`:

```
Array
(
    [scheme] => http
    [host] => www.site.ru
```

```
    [user] => user
    [pass] => pass
    [path] => /path/index.php
    [query] => par=value
    [fragment] => anch
)
```

Если функция `parse_url()` принимает второй параметр, вместо массива возвращается строка с одним из компонентов строки запроса. Параметр может принимать следующие константы:

- `PHP_URL_SCHEME` — префикс, например, http, https, ftp и т. п.;
- `PHP_URL_HOST` — домен;
- `PHP_URL_PORT` — номер порта;
- `PHP_URL_USER` — пользователь;
- `PHP_URL_PASS` — его пароль;
- `PHP_URL_PATH` — путь от корневого каталога;
- `PHP_URL_QUERY` — все, что расположено после символа вопроса (`?`);
- `PHP_URL_FRAGMENT` — все, что расположено после символа `#`;

В листинге 8.5 из строки запроса при помощи функции `parse_url()` и константы `PHP_URL_HOST` извлекается лишь доменное имя.

Листинг 8.5. Использование второго параметра в функции `parse_url()`

```
<?php
  $url = 'http://user:pass@www.site.ru/path/index.php?par=value#anch';
  echo parse_url($url, PHP_URL_HOST); // www.site.ru
?>
```

8.2. HTML-форма и ее обработчик

HTML-формы создаются при помощи парных тегов `<form>` и `</form>`, между которыми располагаются теги элементов управления. В листинге 8.6 представлен HTML-код формы, содержащей два элемента управления с однострочной текстовой областью `text` и кнопку подтверждения `submit`.

Листинг 8.6. HTML-форма

```
<form method='post'>
  <input type='text' name='first'><br>
  <input type='text' name='second'><br>
  <input type='submit' value="Отправить">
</form>
```

Результатом интерпретации HTML-кода из листинга 8.6 будет простейшая HTML-форма, представленная на рис. 8.1.

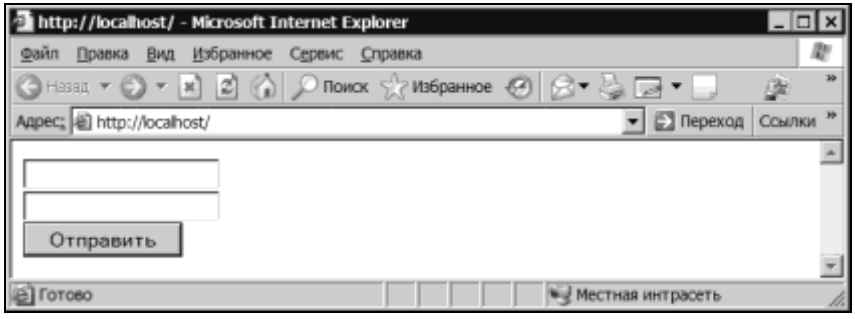


Рис. 8.1. HTML-форма в окне браузера

Элементы управления, как правило, подписываются слева, что может приводить к искажениям, нарушающим дизайн HTML-формы (рис. 8.2).



Рис. 8.2. Искажение дизайна HTML-формы

Для того чтобы предотвратить искажение, представленное на рис. 8.2, элементы управления помещают в ячейки HTML-таблицы (листинг 8.7).

Листинг 8.7. Использование таблицы для структурирования элементов формы

```
<form method='post'>
  <table>
    <tr>
      <td>Первый *:</td>
      <td><input type='text' name='first'></td>
    </tr>
    <tr>
      <td>Второй:</td>
      <td><input type='text' name='second'></td>
```

```
    </tr>
    <tr>
      <td> </td>
      <td><input type='submit' value="Отправить"></td>
    </tr>
  </table>
</form>
```

Результат интерпретации HTML-кода из листинга 8.7 представлен на рис. 8.3.

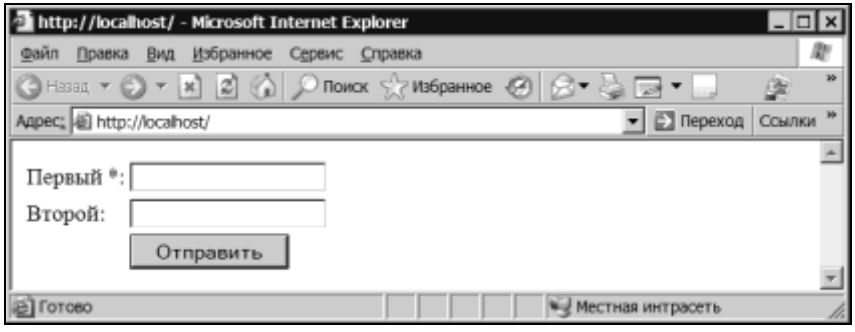


Рис. 8.3. Выравнивание элементов управления HTML-формы

В листингах 8.6 и 8.7 тег `<form>` содержит атрибут `method`, который устанавливает в качестве метода передачи метод POST. Помимо метода POST для передачи данных из HTML-формы в обработчик применяется также метод GET. В табл. 8.2 представлены основные атрибуты, позволяющие управлять поведением HTML-формы.

Таблица 8.2. *Атрибуты тега* `<form>`

Атрибут	Описание
`action`	Указывает адрес обработчика, которому передаются данные из HTML-формы. Если тег `<form>` не содержит атрибута `action`, то данные отправляются в файл, в котором описывается HTML-форма
`enctype`	Определяет формат отправляемых данных при использовании метода передачи данных POST. По умолчанию используется формат `application/x-www-form-urlendoded`. Если HTML-форма содержит элемент управления `file`, предназначенный для передачи файлов на сервер, то следует указать формат `multipart/form-data`
`method`	Определяет метод передачи данных (POST или GET) из HTML-формы обработчику. По умолчанию, если не указывается атрибут `method`, применяется метод GET
`name`	Определяет имя HTML-формы, которое может использоваться для доступа к элементам управления в скриптах, выполняющихся на стороне клиента (например, скриптах JavaScript)

Таблица 8.2 *(окончание)*

Атрибут	Описание
`target`	Указывает окно для вывода результата, полученного от обработчика HTML-формы. Атрибут может принимать следующие значения: • `_blank` — результат открывается в новом окне; • `_self` — результат открывается в текущем окне; данный режим используется по умолчанию, если атрибут `target` не указан явно; • `_parent` — результат открывается в родительском фрейме; при отсутствии фреймов режим аналогичен `_self`; • `_top` — отменяет все фреймы, если они имеются, и загружает страницу в полном окне браузера; при отсутствии фреймов работает как `_self`

Как видно из табл. 8.2, адрес обработчика указывается в атрибуте `action`. Различают два подхода к созданию обработчика HTML-формы:

- обработчик расположен в отдельном файле, после выполнения манипуляций над полученными данными, клиент направляется на главную страницу;
- обработчик располагается в том же самом файле, где находится HTML-форма.

Рассмотрим каждый из случаев более подробно. В листинге 8.8 приводится пример HTML-формы, расположенной в файле index.php, содержащей единственное текстовое поле `first` и кнопку.

Листинг 8.8. HTML-форма (файл index.php)

```
<form action='handler.php' method='post'>
<input type='text' name='first'>
<input type='submit' value="Отправить">
</form>
```

Данные отправляются обработчику, расположенному в файле handler.php, который осуществляет вывод введенной в поле `first` строки в окно браузера (листинг 8.9).

Листинг 8.9. Обработчик HTML-формы (файл handler.php)

```
<?php
  // Если поле first не заполнено, выводим сообщение об ошибке
  if (empty($_POST['first'])) exit("Текстовое поле не заполнено");
  else echo $_POST['first'];
?>
```

Обработчик handler.php проверяет при помощи функции `empty()`, не является ли значение поля `first` пустым. Если поле пустое, работа скрипта останавливается с выдачей сообщения об ошибке. Если поле заполнено корректно, его содержимое выводится в окно браузера.

Замечание

Значения элементов управления, переданных из HTML-формы, извлекаются из суперглобального массива `$_POST` или `$_GET` в зависимости от выбранного метода передачи данных. Для файлов, загружаемых на сервер, предназначен отдельный массив `$_FILES`.

Размещение HTML-формы и обработчика в разных файлах позволяет структурировать код, однако это не всегда удобно. Например, если пользователь забыл ввести данные, то узнает он об этом только на странице обработчика. В результате пользователь вынужден возвращаться обратно и заполнять HTML-форму заново, что может оказаться очень утомительным, если HTML-форма содержит множество обязательных полей.

Выходом из данной ситуации является расположение обработчика непосредственно в файле HTML-формы (листинг 8.10).

Листинг 8.10. Расположение HTML-формы и обработчика в одном файле

```
<?php
  // Обработчик HTML-формы
  $error = array();
  if (!empty($_POST))
  {
    // Если поле first не заполнено, выводим сообщение об ошибке
    if (empty($_POST['first'])) $error[] = "Текстовое поле не заполнено";

    // Если нет ошибок, начинаем обработку данных
    if (empty($error))
    {
      // Выводим содержимое текстового поля first
      echo $_POST['first'];
      // Останавливаем работу скрипта, чтобы после
      // перенаправления не грузилась HTML-форма
      exit();
    }
  }

  // Выводим сообщения об ошибках, если они имеются
  if (!empty($error))
  {
    foreach($error as $err)
    {
      echo "<span style=\"color:red\">$err</span><br>";
    }
  }
```

```
  // HTML-форма
?>
<form method='post'>
<input type='text' name='first'
       value="<?= htmlspecialchars($_POST['first'], ENT_QUOTES); ?>">
<input type='submit' value="Отправить">
</form>
```

Такой подход позволяет не только вывести сообщения об ошибках непосредственно перед HTML-формой, но и сохранить все введенные ранее данные. При проверке данных сообщения об ошибках сохраняются в массив `$error`. Если он оказывается пустым, начинается обработка; если массив содержит сообщения об ошибках, то происходит повторная загрузка HTML-формы с выводом списка обнаруженных ошибок в цикле `foreach`.

8.3. Текстовое поле

Текстовое поле (уже использованное нами в предыдущем разделе) предназначено для ввода пользователем строки текста, как правило, не очень большой. Для создания текстового поля необходимо поместить в HTML-форме между тегами `<form>` и `</form>` тег такого вида:

```
<input type='text'>
```

Следует отметить, для атрибута `type` тип элемента управления `text` является значением по умолчанию. Поэтому если атрибут `type` отсутствует, а также если ему присвоено неизвестное или ошибочное значение, браузер интерпретирует элемент управления как текстовое поле.

Помимо атрибута `type` тег `<input>` может содержать дополнительные атрибуты, представленные в табл. 8.3.

Таблица 8.3. *Атрибуты текстового поля*

Атрибут	Описание
`maxlength`	Определяет максимально допустимую длину текстовой строки. При отсутствии атрибута количество символов не ограничено
`name`	Имя элемента управления, предназначенное для идентификации в обработчике. Имя должно быть уникальным в пределах формы, т. е. отличаться от имен других элементов управления, т. к. используется в качестве ключа для доступа к значению поля в обработчике
`size`	Ширина элемента управления, определяющая физический размер элемента управления на странице сайта
`value`	Начальный текст, содержащийся в поле

8.4. Поле для приема пароля

Для ввода пароля обычно используют не текстовое поле, а специальное поле типа `password`, которое совпадает по атрибутам и внешнему виду с текстовым полем `text`, однако вводимый текст остается скрытым за символами звездочек или точек (листинг 8.11).

Листинг 8.11. HTML-форма с полем для приема пароля

```
<form method='post'>
  <table>
    <tr>
      <td>Имя *:</td>
      <td><input type='text' name='name'></td>
    </tr>
    <tr>
      <td>Пароль:</td>
      <td><input type='password' name='second'></td>
    </tr>
    <tr>
      <td> </td>
      <td><input type='submit' value="Отправить"></td>
    </tr>
  </table>
</form>
```

Внешний вид HTML-формы из листинга 8.11 представлен на рис. 8.4.

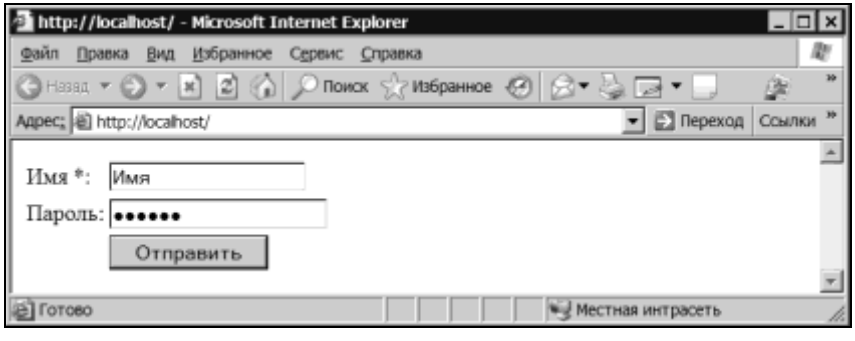


Рис. 8.4. Использование поля для приема пароля

8.5. Текстовая область

Текстовая область предназначена для ввода нескольких строк текста и создается при помощи парного тега <textarea>. В отличие от текстового поля <input> в текстовой области допускается создание переводов строк (абзацев). Тег <textarea> имеет следующий синтаксис:

```
<textarea>...</textarea>
```

Как можно заметить, текстовая область определяется не с помощью атрибута, а посредством собственного тега. Допустимые атрибуты тега <textarea> приводятся в табл. 8.4.

Таблица 8.4. *Атрибуты текстовой области*

Атрибут	Описание
cols	Ширина текстовой области, равная количеству символов моноширинного шрифта
disabled	Блокирует возможность редактирования и выделения текста в текстовой области, при этом сам элемент управления окрашивается в серый цвет
name	Имя элемента управления, предназначенное для идентификации в обработчике. Имя должно быть уникальным в пределах формы, т. е. отличаться от имен других элементов управления, поскольку используется в качестве ключа для доступа к значению поля в обработчике
readonly	Блокирует возможность редактирования текстовой области, однако, в отличие от атрибута disabled, цвет элемента управления остается неизменным
rows	Высота текстовой области, равная количеству отображаемых строк без прокрутки содержимого
wrap	Требует от браузера, чтобы перенос текста на следующую строку осуществлялся только при нажатии клавиши <Enter>, в противном случае должна появляться горизонтальная полоса прокрутки

В листинге 8.12 приводится пример использования текстовой области для приема большого объема текста.

Листинг 8.12. Поле с текстовой областью

```
<form method='post'>
<textarea name='name' cols='50' rows='10'></textarea><br>
<input type='submit' value="Отправить">
</form>
```

Внешний вид HTML-формы из листинга 8.12 представлен на рис. 8.5.

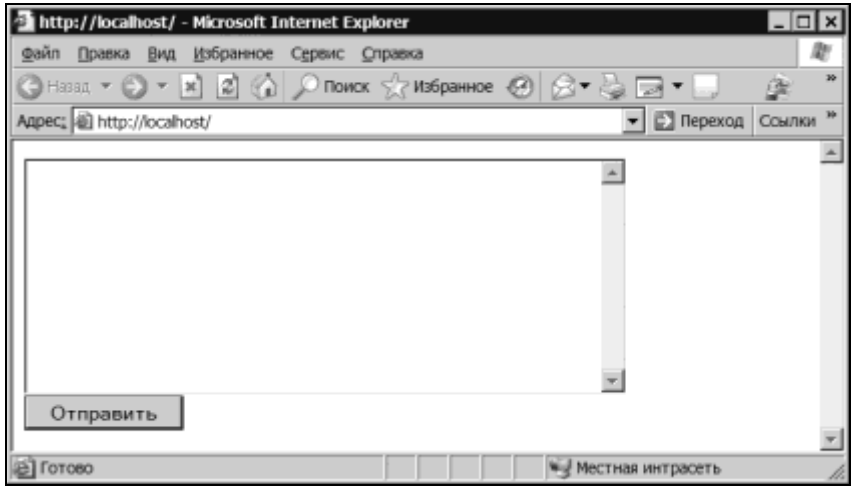


Рис. 8.5. Использование текстовой области

8.6. Скрытое поле

Скрытое поле служит для передачи незаметно от пользователя служебной информации. Скрытые поля не отображаются на странице.

> ***ЗАМЕЧАНИЕ***
>
> Дело в том, что протокол HTTP не сессионный, и проблема передачи информации от страницы к странице стоит в нем достаточно остро. Помимо скрытых полей передавать информацию можно при помощи механизма сессий (session), однако такой подход не всегда удобен, т. к. сессии носят глобальный характер и могут искажаться другой частью приложения.

Скрытое поле создается при помощи `input`-тега, атрибут `type` которого принимает значение `hidden`:

```
<input type="hidden">
```

Помимо атрибута `type`, скрытое поле поддерживает атрибут `name` для уникального имени элемента управления и атрибут `value` для его значения. Модифицируем HTML-форму из листинга 8.12 так, чтобы она включала скрытое поле `id`, заполняемое из GET-параметра (листинг 8.13).

Листинг 8.13. Использование скрытого поля (index.php)

```
<form method='post' action='handler.php'>
<textarea name='name' cols='50' rows='10'></textarea><br>
<input type='submit' value="Отправить">
<input name='id' type='hidden' value="<?= $_GET['id']; ?>">
</form>
```

Теперь если передать скрипту index.php GET-параметр `id`: **index.php?id=1**, он должен появиться в массиве `$_POST` в обработчике handler.php (листинг 8.14).

Листинг 8.14. Обработчик HTML-формы (handler.php)

```
<?php
  echo "<pre>";
  print_r($_POST);
  echo "</pre>";
?>
```

Результат выполнения обработчика handler.php из листинга 8.14 может выглядеть следующим образом:

```
Array
(
    [name] => 123
    [id] => 1
)
```

8.7. Флажок

Флажок — это элемент управления, позволяющий представлять логическое значение в HTML-форме, находясь в установленном или снятом состоянии. Синтаксис элемента управления выглядит следующим образом:

```
<input type="checkbox">
```

Помимо атрибута `type`, флажок поддерживает атрибут `name` для уникального имени элемента управления, атрибут `value` для его значения и атрибут `checked`, наличие которого означает, что флажок установлен.

Создадим HTML-форму, содержащую пять флажков, два из которых являются предустановленными (отмеченными) (листинг 8.15).

Листинг 8.15. Использование флажков

```
<form method='post' action='handler.php'>
<input type='checkbox' name='php' checked>Вы знакомы с PHP?<br>
<input type='checkbox' name='mysql' checked>Вы знакомы с MySQL?<br>
<input type='checkbox' name='perl'>Вы знакомы с Perl?<br>
<input type='checkbox' name='phyton'>Вы знакомы с Python?<br>
<input type='checkbox' name='ruby'>Вы знакомы с Ruby?<br>
<input type='submit' value="Отправить">
</form>
```

Внешний вид HTML-формы из листинга 8.15 представлен на рис. 8.6.

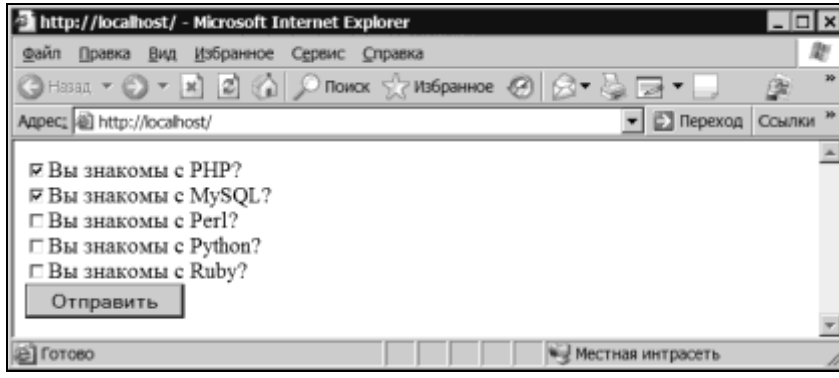


Рис. 8.6. Использование флажков

Интересно отметить, что в суперглобальный массив `$_POST` попадают только те флажки, которые были отмечены, для неотмеченных флажков соответствующий элемент не заводится. Если в качестве обработчика используется скрипт из листинга 8.14, то результатом отправки данных обработчику будет вывод следующего дампа массива:

```
Array
(
    [php] => on
    [mysql] => on
)
```

По умолчанию, в качестве значения для флажка выступает строка `"on"`, однако это значение можно изменить, если снабдить флажок атрибутом `value` (листинг 8.16).

Листинг 8.16. Использование атрибута `value` во флажках

```
<form method='post' action='handler.php'>
<input type='checkbox' name='php' value='1' checked>
Вы знакомы с PHP?<br>
<input type='checkbox' name='mysql' value='2' checked>
Вы знакомы с MySQL?<br>
<input type='checkbox' name='perl' value='3'>
Вы знакомы с Perl?<br>
<input type='checkbox' name='phyton' value='4'>
Вы знакомы с Python?<br>
<input type='checkbox' name='ruby' value='5'>
Вы знакомы с Ruby?<br>
<input type='submit' value="Отправить">
</form>
```

Результат отправки данных из HTML-формы из листинга 8.16 может выглядеть следующим образом:

```
Array
(
    [php] => 1
    [mysql] => 2
)
```

8.8. Список

Список позволяет выбрать одно или несколько значений из определенного набора и имеет следующий синтаксис:

```
<select>
  <option>Первый пункт</option>
  <option>Второй пункт</option>
  <option>Третий пункт</option>
</select>
```

Между тегами `<select>` и `</select>` располагаются пункты списка, которые оформляются в виде `option`-тегов. В приведенном выше примере список имеет три пункта.

Помимо традиционного атрибута `name`, тег `<select>` может иметь атрибуты `multiple` и `size`. Параметр `multiple` позволяет выбрать несколько пунктов списка, отмечая их правой кнопкой мыши при одновременном удержании клавиши `<Ctrl>`. Атрибут `size` определяет высоту списка в пунктах.

Тег `<select>` не имеет атрибута `value` — этот атрибут располагается в теге `<option>`. Помимо этого, тег `<option>` может иметь атрибут `selected` для обозначения выделения текущего пункта.

В листинге 8.17 приводится пример HTML-формы, содержащей два выпадающих списка: первый вариант — множественный список, который использует атрибут `multiple` и имеет высоту, равную 3 (по количеству элементов), второй список является выпадающим и позволяет выбрать только одно значение.

> ***ЗАМЕЧАНИЕ***
>
> Если используется множественный список, для корректной передачи всех выбранных значений, в качестве названия элемента `<select>` должен выступать массив.

Листинг 8.17. Использование выпадающих списков

```
<form method='post' action='handler.php'>
<table>
  <tr valign='top'>
    <td>Выбор<br>нескольких<br>значений</td>
```

```
    <td>
      <select name='fst[]' multiple size='3'>
        <option value='1' selected>Первый пункт</option>
        <option value='2'>Второй пункт</option>
        <option value='3' selected>Третий пункт</option>
      </select>
    </td>
  </tr>
  <tr valign='top'>
    <td>Одно значение</td>
    <td>
      <select name='snd'>
        <option value='1'>Первый пункт</option>
        <option value='2'>Второй пункт</option>
        <option value='3'>Третий пункт</option>
      </select>
    </td>
  </tr>
  <tr>
    <td></td>
    <td><input type='submit' value="Отправить"></td>
  </tr>
</table>
</form>
```

Внешний вид HTML-формы из листинга 8.17 представлен на рис. 8.7.

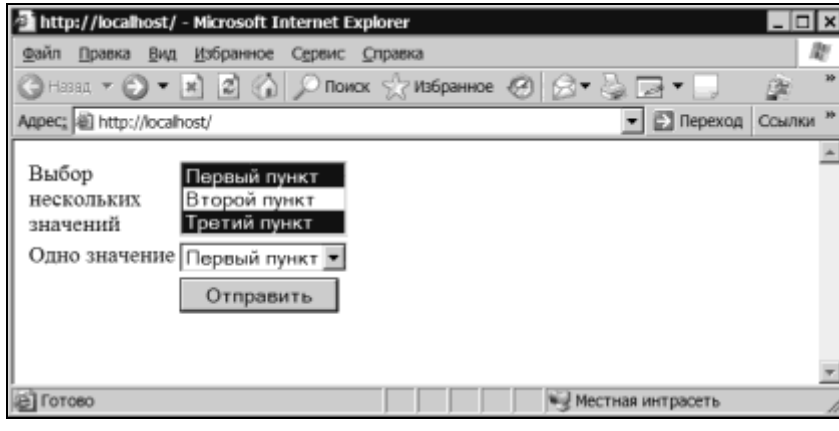


Рис. 8.7. Множественный и одиночный выпадающие списки

Если в качестве обработчика используется скрипт из листинга 8.14, то результатом отправки данных обработчику будет вывод следующего дампа массива:

```
Array
(
    [fst] => Array
        (
            [0] => 1
            [1] => 3
        )

    [snd] => 1
)
```

8.9. Переключатель

Переключатель, или, иначе, радиокнопка — элемент управления, позволяющий выбрать из набора утверждений только одно. Он имеет следующий синтаксис:

```
<input type="radio">
```

Для формирования набора утверждений используется несколько переключателей, которым присваивается одно и то же имя через атрибут `name`. Помимо традиционных атрибутов `type` и `name`, переключатели могут быть снабжены атрибутом `value` для передачи значения и атрибутом `checked` для того, чтобы отметить один из переключателей по умолчанию.

В листинге 8.18 приводится пример использования переключателей для оценки по пятибалльной системе.

Листинг 8.18. Использование переключателей

```
<form method='post' action='handler.php'>
Оцените сайт
<input type="radio" name="mark" value="1">1
<input type="radio" name="mark" value="2">2
<input type="radio" name="mark" value="3" checked>3
<input type="radio" name="mark" value="4">4
<input type="radio" name="mark" value="5">5
<input type='submit' value="Отправить">
</form>
```

Внешний вид HTML-формы из листинга 8.18 представлен на рис. 8.8.

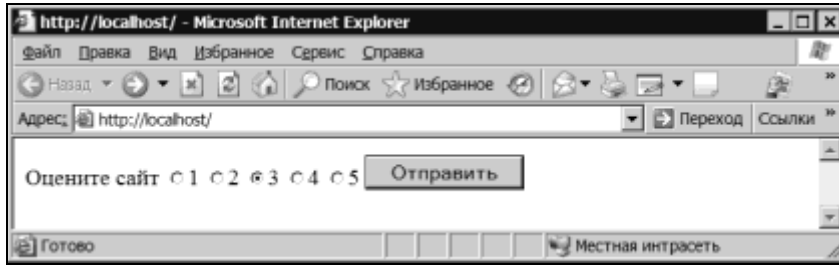


Рис. 8.8. Переключатели

Если в качестве обработчика используется скрипт из листинга 8.14, то результатом отправки данных обработчику будет вывод следующего дампа массива:

```
Array
(
    [mark] => 3
)
```

8.10. Загрузка файла на сервер

Для загрузки пользовательских файлов на сервер используется специальный элемент управления, позволяющий указать путь к загружаемому файлу (при помощи кнопки **Обзор**). Элемент управления имеет следующий синтаксис:

```
<input type='file'>
```

Помимо атрибута `type`, элемент управления допускает указания атрибутов `name` и `size`. Простая форма для отправки файла на сервер демонстрируется в листинге 8.19.

Листинг 8.19. HTML-форма для загрузки файлов на сервер — `index.html`

```
<html><head><title> Загрузка файлов на сервер </title></head><body>
  <h2><b> Форма для загрузки файлов </b></h2>
   <form action="upload.php" method="post" enctype="multipart/form-data">
   <input type="file" name="filename"><br>
   <input type="submit" value="Загрузить"><br>
   </form>
 </body>
</html>
```

Атрибут `entype` формы определяет вид кодировки, которую браузер применяет к параметрам формы. Для того чтобы отправка файлов на сервер действовала, атрибуту `entype` необходимо присвоить значение `multipart/form-data`. По умолчанию этот атрибут имеет значение `application/x-www-form-urlencoded`.

Если все сделано правильно, форма для отправки файлов на сервер должна выглядеть так, как это показано на рис. 8.9.

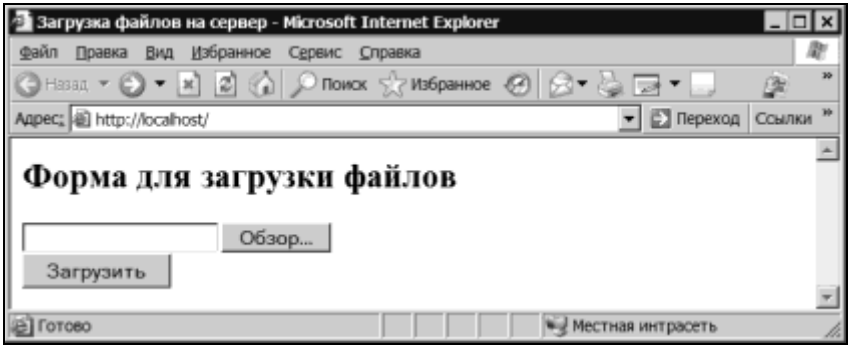


Рис. 8.9. Загрузка файла на сервер

После того как получен HTTP-запрос, содержимое загруженного файла записывается во временный файл, который создается в каталоге сервера, заданном по умолчанию для временных файлов, если другой каталог не задан в файле php.ini (директива `upload_tmp_dir`).

Характеристики загруженного файла доступны через двумерный суперглобальный массив `$_FILES`. При этом переменная со значениями этого массива может иметь следующий вид:

- `$_FILES['filename']['name']` (содержит исходное имя файла на клиентской машине);
- `$_FILES['filename']['size']` (содержит размер загруженного файла в байтах);
- `$_FILES['filename']['type']` (содержит MIME-тип файла);
- `$_FILES['filename']['tmp_name']` (содержит имя временного файла, в который сохраняется загруженный файл).

В листинге 8.20 приведен скрипт upload.php, который загружает файл на сервер и копирует его из временного каталога в каталог temp.

> ***ЗАМЕЧАНИЕ***
>
> Проверить успешность загрузки файла на сервер можно при помощи специальной функции `is_uploaded_file()`, которая принимает в качестве единственного параметра имя файла (`$_FILES['filename']['name']`) и возвращает `TRUE` в случае успешной загрузки и `FALSE` в случае неудачи. Функции, обслуживающие загруженный на сервер файл, более подробно обсуждаются в *разд. 13.2*.

Листинг 8.20. Загрузка файлов на сервер — upload.php

```
<html>
<head>
<title> Результат загрузки файла </title>
</head>
```

```
<body>
<?php
  if (copy($_FILES['filename']['tmp_name'],
     "temp/".$_FILES['filename']['name']))
  {
    echo "Файл успешно загружен";
  }
  else
  {
   echo "Ошибка загрузки файла";
  }
?>
</body>
</html>
```

После выполнения этого скрипта выбранный для загрузки файл будет помещен в подкаталог temp каталога, в котором расположен скрипт, а браузер выдаст фразу "Файл успешно загружен".

Скрипт из листинга 8.21 позволяет вывести характеристики загруженного файла.

Листинг 8.21. Характеристики файла

```
<html>
<head>
<title> Результат загрузки файла </title>
</head>
<body>
<?php
  if (copy($_FILES["filename"]["tmp_name"],
     "temp/".$_FILES["filename"]["name"]))
  {
    echo "Файл успешно загружен <br>";
    // далее выводится информация о файле
    echo "Характеристики файла: <br>";
    echo "Имя файла: ";
    echo $_FILES["filename"]["name"];
    echo "<br>Размер файла: ";
    echo $_FILES["filename"]["size"];
    echo "<br>Каталог для загрузки: ";
    echo $_FILES["filename"]["tmp_name"];
    echo "<br>Тип файла: ";
```

```
    echo $_FILES["filename"]["type"];
  }
  else
  {
    echo "Ошибка загрузки файла";
  }
?>
</body>
</html>
```

Результат выглядит так, как представлено на рис. 8.10.

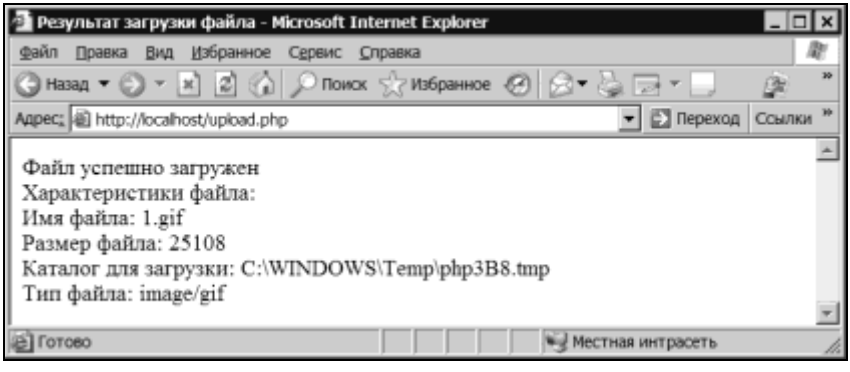


Рис. 8.10. Информация о загруженном файле

В некоторых случаях требуется ограничить размер файла, который может быть загружен на сервер. К примеру, чтобы разрешить загрузку на сервер файлов размером не более 3 Мбайт, нужно изменить скрипт так, как это представлено в листинге 8.22.

Листинг 8.22. Ограничение размера загружаемого файла

```
<?php
  if ($_FILES['filename']['size'] > 3*1024*1024)
  {
    exit "Размер файла превышает три мегабайта";
  }
  if (copy(_FILES['filename']['tmp_name'],
     "temp/".$_FILES['filename']['name']))
  {
    echo "Файл успешно загружен <br>";
  }
  else
```

```
  {
    echo "Ошибка загрузки файла";
  }
?>
```

Максимальный размер загружаемого файла можно также задать при помощи директивы `upload_max_filesize`, значение которой по умолчанию равно 2 Мбайт:

```
if ($_FILES['filename']['size'] > upload_max_filesize)
```

> ***Замечание***
>
> Значение директивы `upload_max_filesize` можно изменить в конфигурационном файле php.ini.



Глава 9

Строковые функции

Строки и функции их обработки являются одним из главных инструментов в скриптовых языках программирования, особенно если прикладной деятельностью является Web-разработка. Поэтому работе со строковыми переменными в PHP уделяется пристальное внимание. В данной главе будут рассмотрены функции, позволяющие обрабатывать строки.

> ***Замечание***
>
> Дополнительную информацию о строках можно получить в *разд. 3.5* и *6.12*.

9.1. Функции для работы с символами

В *разд. 6.12* уже упоминалось, что каждая строка может рассматриваться как массив символов. В табл. 9.1 приводится список функций, предназначенных для работы с символами строки.

Таблица 9.1. *Функции для работы с символами*

Функция	Описание
`strlen($str)`	Возвращает количество символов в строке `$str`
`chr($ascii)`	Принимает в качестве аргумента ASCII-код `$ascii` символа и возвращает соответствующий этому коду фактический символ
`ord($str)`	Возвращает ASCII-код символа `$str`, переданного ей в качестве аргумента
`count_chars($str [, $mode])`	Подсчитывает количество вхождений каждого из символов с ASCII-кодами в диапазоне 0...255 в строку `$str`. Необязательный параметр `$mode` позволяет задать формат результата
`str_shuffle($str)`	Возвращает копию строки `$str`, символы которой перемешаны в случайном порядке
`strrev($str)`	Возвращает копию строки `$str`, символы которой перевернуты, т. е. начальные символы помещены в конец, а конечные символы в начало

Функция `strlen()` принимает в качестве единственного аргумента строку и возвращает ее длину (листинг 9.1).

Листинг 9.1. Использование функции `strlen()`

```
<?php
  echo strlen("Hello, world!!!")."<br>"; // 15
  echo strlen("  Hello, PHP!  ");        // 15
?>
```

Функция `strlen()` часто используется совместно с другими функциями, а так же при работе с посимвольным представлением строки. В листинге 9.2 в цикле `for` в столбик выводится строка `$str`, содержащая слово `"Hello"`. Скрипт основан на том факте, что элемент `$str[0]` содержит символ `H`, `$str[1]` — символ `e`, `$str[2]` — символ `l` и т. д. до конца строки.

Листинг 9.2. Обход строки в цикле

```
<?php
  $str = "Hello";
  for($i = 0; $i < strlen($str); $i++)
  {
     echo $str[$i]."<br>";
  }
?>
```

Результатом работы скрипта из листинга 9.2 будет вертикальный вывод слова `"Hello"`:

```
H
e
l
l
o
```

Функция `chr()` принимает в качестве аргумента ASCII-код символа и возвращает соответствующий этому коду фактический символ (листинг 9.3).

Листинг 9.3. Использование функции `chr()`

```
<?php
  $str = chr(36);
  echo $str;
?>
```

Результат: `$`.

Функция `ord()` выполняет действие, обратное функции `chr()` — возвращает ASCII-код символа, переданного ей в качестве аргумента (листинг 9.4).

> ***ЗАМЕЧАНИЕ***
>
> Если параметр функции `ord()` содержит более одного символа, ASCII-код будет возвращен только для первого.

Листинг 9.4. Использование функции `ord()`

```
<?php
  $str = ord('$');
  echo $str;
?>
```

Результат: `36`.

Еще одной интересной функцией, позволяющей получить статистическое распределение символов в строке, является функция `count_chars()`, которая имеет следующий синтаксис:

```
count_chars($str [, $mode])
```

Функция подсчитывает количество вхождений каждого из символов с ASCII-кодами в диапазоне 0...255 в строку `$str` и возвращает эту информацию в различных форматах. Необязательный аргумент `$mode` по умолчанию равен 0. В зависимости от его значения возвращается:

- 0 — массив, индексами которого являются ASCII-коды, а значениями — число вхождений соответствующего символа;
- 1 — то же, что и для 0, но информация о символах с нулевым числом вхождений не включается в массив;
- 2 — то же, что и для 0, но в массив включается информация только о символах с нулевым числом вхождений;
- 3 — строка, состоящая из символов, которые входят в исходную строку хотя бы раз;
- 4 — строка, состоящая из символов, которые не входят в исходную строку.

В листинге 9.5 приводится пример использования функции `count_chars()`.

> ***ЗАМЕЧАНИЕ***
>
> Функция `count_chars()` работает только с символами, если необходимо подсчитать количество подстрок, входящих в ту или иную строку, следует воспользоваться функцией `substr_count()`, работа которой более подробно освещена в *разд. 9.2*.

Листинг 9.5. Пример с функцией `count_chars()`

```
<?php
  $data = "Две в и одна с";
  $result = count_chars($data, 0);
```

```
  for($i=0; $i < count($result); $i++)
  {
    if ($result[$i] != 0)
       echo "'".chr($i)."' встречается в строке $result[$i] раз(а).<br>";
  }
?>
```

Результатом работы скрипта из листинга 9.5 будут следующие строки:

```
' ' встречается в строке 4 раз(а).
'Д' встречается в строке 1 раз(а).
'а' встречается в строке 1 раз(а).
'в' встречается в строке 2 раз(а).
'д' встречается в строке 1 раз(а).
'е' встречается в строке 1 раз(а).
'и' встречается в строке 1 раз(а).
'н' встречается в строке 1 раз(а).
'о' встречается в строке 1 раз(а).
'с' встречается в строке 1 раз(а).
```

В листинге 9.6 демонстрируется использование функции `str_shuffle()`, перемешивающей символы строки в случайном порядке.

Листинг 9.6. Использование функции `str_shuffle()`

```
<?php
  $str = 'Hello, world!';
  echo str_shuffle($str); // dwr,!ll Holoe
?>
```

Функция `strrev()` производит реверс строки, т. е. переворачивает строку, помещая начальные символы в конец, а конечные символы в начало. Функция имеет следующий синтаксис:

```
strrev($str)
```

В листинге 9.7 приводится пример использования функции `strrev()`.

Листинг 9.7. Реверс строки

```
<?php
  echo strrev("компьютер"); // ретюьпмок
?>
```

Помимо представленных в табл. 9.1 существует дополнительный набор информационных функций, позволяющих ответить на вопрос о символьном составе строки (табл. 9.2).

Таблица 9.2. *Функции определения символьного состава строк*

Функция	Описание
`ctype_alnum($str)`	Возвращает `TRUE`, если строка `$str` содержит только буквы английского алфавита и цифры, в противном случае возвращается `FALSE`
`ctype_alpha($str)`	Возвращает `TRUE`, если строка `$str` содержит только буквы английского алфавита, в противном случае возвращается `FALSE`
`ctype_xdigit($str)`	Возвращает `TRUE`, если строка `$str` содержит только цифры и буквы A, B, C, D, E, F в верхнем или нижнем регистрах (т. е. символы, из которых состоят шестнадцатеричные числа). В противном случае возвращается `FALSE`
`ctype_cntrl($str)`	Возвращает `TRUE`, если строка `$str` содержит только управляющие символы (перевод строки, символ табуляции и т. п.), в противном случае возвращается `FALSE`
`ctype_digit($str)`	Возвращает `TRUE`, если строка `$str` содержит только цифры, в противном случае возвращается `FALSE`
`ctype_print($str)`	Возвращает `TRUE`, если строка `$str` содержит только видимые символы, в противном случае возвращается `FALSE`
`ctype_graph($str)`	Возвращает `TRUE`, если строка `$str` содержит только видимые графические символы, в противном случае возвращает `FALSE`. Пробел, символ табуляции традиционно относятся к видимым символам в таблице ASCII-кодов, однако не входят в группу графических символов — для них функция `ctype_graph()` вернет `FALSE`
`ctype_space($str)`	Возвращает `TRUE`, если строка `$str` содержит только пробельные символы (пробел, табуляция, перевод строки и т. п.), в противном случае возвращается `FALSE`
`ctype_punct($str)`	Возвращает `TRUE`, если строка `$str` содержит только символы пунктуации (видимые символы за исключением символов английского алфавита, цифр и пробельных символов), в противном случае возвращается `FALSE`
`ctype_lower($str)`	Возвращает `TRUE`, если строка `$str` содержит только буквы английского алфавита в нижнем регистре, в противном случае возвращается `FALSE`
`ctype_upper($str)`	Возвращает `TRUE`, если строка `$str` содержит только буквы английского алфавита в верхнем регистре, в противном случае возвращается `FALSE`

В листинге 9.8 приводится пример использования функции `ctype_digit()`, проверяющей, содержит ли строка что-то помимо чисел.

Листинг 9.8. Использование функции `ctype_digit()`

```
<?php
  if (ctype_digit("100.0")) // FALSE
    echo "Строка содержит только числа<br>";
  else
    echo "Строка содержит что-то помимо чисел<br>";

  if (ctype_digit("100")) // TRUE
    echo "Строка содержит только числа<br>";
  else
    echo "Строка содержит что-то помимо чисел<br>";
?>
```

Как видно из листинга 9.8, функции из таблицы 9.2 проверяют каждый символ строки, и, если он не удовлетворяет условию, вся строка признается не удовлетворяющей условию. В реальной практике при проверке, содержит ли строка число, используют функции `is_numeric()` (листинг 9.9).

Листинг 9.9. Использование функции `is_numeric()`

```
<?php
  if (is_numeric("100.0")) // TRUE
    echo "Число<br>";
  else
    echo "Не число<br>";

  if (is_numeric("100")) // TRUE
    echo "Число<br>";
  else
    echo "Не число<br>";

  if (is_numeric(".5")) // TRUE
    echo "Число<br>";
  else
    echo "Не число<br>";

  if (is_numeric("f.5")) // FALSE
    echo "Число<br>";
  else
    echo "Не число<br>";

  if (is_numeric("5f")) // FALSE
```

```
    echo "Число<br>";
  else
    echo "Не число<br>";
?>
```

9.2. Поиск в строке

Задача поиска в строках является очень распространенной при создании Web-приложений. Существуют два подхода к решению проблемы: при помощи строковых функций и регулярных выражений *(см. главу 10)*. Регулярные выражения позволяют решать более сложные задачи, и в то же время они менее производительны, т. к. поиск с использованием регулярных выражений требует значительных расчетов. В табл. 9.3 приводится список строковых функций, осуществляющих поиск в строке.

Таблица 9.3. *Строковые функции поиска*

Функция	**Описание**
`substr($str, $start [, $length])`	Возвращает для строки `$str` подстроку, которая начинается с символа `$start` (отсчет с нуля) и имеет `$length` символов. Если количество символов `$length` не указано, предполагается, что подстрока должна включать все символы до конца строки `$str`
`substr_count($str, $search [, $offset [, $length]])`	Возвращает количество вхождений подстроки `$search` в строку `$str`. Поиск осуществляется с позиции `$offset` в подстроке, длиной `$lenght` (если эти два параметра указаны)
`strpos($str, $search [, $offset])`	Возвращает позицию первого вхождения подстроки `$search` в строку `$str`. Поиск может начинаться с позиции `$offset`, если указывается третий параметр. Возвращает `FALSE`, если подстрока `$search` не найдена
`stripos($str, $search [, $offset])`	Функция `stripos()` аналогична функции `strops()`, за исключением того факта, что в процессе поиска не учитывается регистр
`strrpos($str, $search [, $offset])`	Возвращает позицию первого вхождения подстроки `$search` с конца строки `$str`. Необязательный параметр `$offset` позволяет задать позицию с конца строки, начиная с которой следует осуществлять поиск. Возвращает `FALSE`, если подстрока `$search` не найдена
`strripos($haystack, $needle [, $offset])`	Функция `strripos()` аналогична функции `strrpos()`, за исключением того факта, что в процессе поиска не учитывается регистр
`strstr($str, $search [, $before])`	Ищет первое вхождение подстроки `$search` в строку `$str`. По умолчанию функция возвращает подстроку, начиная с первого вхождения `$search` до конца строки `$str`, однако если необязательный параметр `$before` принимает параметр `TRUE`, возвращается подстрока, предшествующая первому вхождению подстроки `$search`. Возвращает `FALSE`, если подстрока `$search` не найдена

Таблица 9.3 *(окончание)*

Функция	Описание
`stristr($str, $search [, $before])`	Функция `stristr()` аналогична функции `strstr()`, за исключением того факта, что в процессе поиска не учитывается регистр
`strchr()`	Синоним функции `strstr()`
`strrchr($str, $search)`	Ищет последнее вхождение подстроки `$search` в строку `$str` и возвращает подстроку, начиная с `$search` и до конца строки `$str`. Возвращает `FALSE`, если подстрока `$search` не найдена
`strpbrk($str, $char_list)`	Ищет в строке `$str` символы из строки `$char_list` и возвращает подстроку, начиная с первого найденного символа и до конца строки. Если ни один символ не найден, возвращается `FALSE`

Функция `substr()` возвращает часть строки (подстроку). В качестве первого аргумента `$str` функции передается исходная строка, из которой вырезается текст; второй аргумент `$start` определяет начало подстроки (отсчет начинается с нуля); третий аргумент `$length` задает длину возвращаемой подстроки в символах. Если третий аргумент не указан, то возвращается вся оставшаяся часть строки.

Листинг 9.10. Извлечение даты, месяца и года из строки

```
<?php
  $str = "04.05.2009";
  echo "день - ".substr($str, 0, 2)."<br>";  // день - 04
  echo "месяц - ".substr($str, 3, 2)."<br>"; // месяц - 05
  echo "год - ".substr($str, 6)."<br>";      // год - 2009
?>
```

В листинге 9.10 приводится пример использования функции `substr()` — из строки `"04.05.2009"` извлекается дата, месяц и год.

Функция `substr_count()` позволяет ответить на вопрос, сколько раз подстрока встречается в строке. В листинге 9.11 при помощи этой функции анализируется, сколько раз подстрока "hello" входит в состав строки.

Листинг 9.11. Использование функции `substr_count()`

```
<?php
  $text = "hello world, hello";
  echo substr_count($text, "hello"); // 2
?>
```

По аналогии с функцией `substr()`, функция `substr_count()` может принимать два необязательных параметра: третий параметр `$offset` — задающий позицию, с которой начинается поиск, и `$lenght`, сообщающий длину подстроки, в которой поиск будет осуществляться. В листинге 9.12 функция обнаружит вхождение только одного символа `"l"`, т. к. поиск начинается лишь с пробела и продолжается только до запятой.

Листинг 9.12. Использование дополнительных параметров в функции `substr_count()`

```
<?php
  $text = "hello world, hello";
  echo substr_count($text, "l", 5, 7); // 1
?>
```

Функция `strpos()` возвращает позицию вхождения подстроки в строку и имеет следующий синтаксис:

```
strpos($str, $search[, $offset])
```

Функция возвращает позицию в строке *`$str`*, первого вхождения подстроки *`$search`*. Если строка не найдена, возвращается `FALSE`. В листинге 9.13 приводится простейший пример использования функции `strpos()`.

> ***Замечание***
>
> Функция `strpos()` при поиске учитывает регистр. Для того чтобы поиск осуществлялся без его учета, следует воспользоваться функцией `stripos()`.

Листинг 9.13. Использование функции `strpos()`

```
<?php
  $var = strpos("Hello, world!", "world");
  echo $var; // 7
?>
```

Необязательный параметр *`$offset`* позволяет указать в строке позицию, начиная с которой будет осуществляться поиск (листинг 9.14).

Листинг 9.14. Использование необязательного параметра *`$offset`*

```
<?php
  $var = strpos("Hello,world!Hello,world!Hello,world!", "world", 7);
  echo $var; // 18
?>
```

Функции `strops()` редко используется сама по себе, чаще в комбинации с другими строковыми функциями (листинг 9.15).

Листинг 9.15. Совместное использование функций `strpos()` и `substr()`

```
<?php
  $str = "PHP является интерпретируемым языком";
  echo substr($str, strpos($str, "является"));
  // Выводит "является интерпретируемым языком"
?>
```

Скрипт в листинге 9.15 ищет позицию, с которой начинается подстрока "является", и выводит в окно браузера строку, начиная с этой позиции до конца строки.

Функция `strrpos()` также ищет позицию вхождения подстроки в строку, но только начиная с конца строки, справа налево, т. е. функция является обратной функции `strpos()` и имеет следующий синтаксис:

```
strrpos($str, $search [, $offset])
```

Эта функция ищет в строке *`$str`* последнюю позицию, где встречается подстрока *`$search`*. В листинге 9.16 приводится простейший пример использования данной функции.

> ***Замечание***
>
> Функция `strrpos()` при поиске учитывает регистр. Для того чтобы поиск осуществлялся без его учета, следует воспользоваться функцией `strripos()`.

Листинг 9.16. Пример с функцией `strrpos()`

```
<?php
  $var = strrpos("Hello world!", "l");
  echo $var; // 9
  $var = strrpos("Hello world!", "l", -10);
  echo $var; // 2
  $var = strrpos("Hello world!", "l", 10);
  echo $var; // FALSE
?>
```

Необязательный третий параметр позволяет задать позицию, ограничивающую поиск, положительное значение с начала строки, отрицательное — с конца.

Функция `strstr()` предназначена для получения части строки и имеет следующий синтаксис:

```
strstr($str, $search [, $before])
```

Функция ищет первое вхождение подстроки *`$search`* в строку *`$str`*. По умолчанию функция возвращает подстроку, начиная с первого вхождения *`$search`* до конца строки *`$str`*, однако, если необязательный параметр *`$before`* принимает параметр `TRUE`, возвращается подстрока, предшествующая первому вхождению подстроки *`$search`*. Возвращает `FALSE`, если подстрока *`$search`* не найдена.

ЗАМЕЧАНИЕ

Третий параметр `$before` был введен в PHP, начиная с версии 6.0.0.

Листинг 9.17. Пример с функцией `strstr()`

```
<?php
  $url = "http://www.softtime.ru";
  echo strstr($url, "www");       // www.softtime.ru
  echo strstr($url, "www", TRUE); // http://
?>
```

В листинге 9.17 первый вызов функции `strstr()` возвращает часть адреса, начиная с подстроки `"www"` и до конца строки. Второй вызов возвращает часть строки, предшествующую подстроке `"www"`.

Следует заметить, что функция `strstr()` чувствительна к регистру, т. е. скрипт в листинге 9.18 не выведет ничего, т. к. подстрока `"WWW"` не будет найдена в строке `"http://www.softtime.ru"`.

Листинг 9.18. Функция `strstr()` зависит от регистра

```
<?php
  $url = "http://www.softtime.ru";
  echo strstr($url, "WWW");
?>
```

Если необходим поиск без учета регистра, следует воспользоваться функцией `stristr()` (листинг 9.19).

Листинг 9.19. Функция `stristr()` зависит от регистра

```
<?php
  $url = "http://www.softtime.ru";
  echo stristr($url, "WWW"); // www.softtime.ru
?>
```

Функция `strrchr()` отличается от функции `strstr()` тем, что осуществляет поиск последнего вхождения подстроки. Функция имеет следующий синтаксис:

```
strrchr($str, $search)
```

Функция возвращает часть строки *`$str`*, начиная с последнего вхождения подстроки *`$search`* в строку *`$str`*. В том случае, если подстрока *`$search`* не найдена в строке *`$str`*, возвращается `FALSE`. В листинге 9.20 при помощи функции `strrchr()` из пути извлекается имя файла.

Листинг 9.20. Использование функции `strrchr()`

```
<?php
  $path = "D:/main/scripts/index.php";
  echo strrchr($path, "/"); // /index.php
?>
```

Еще одной функцией, предназначенной для получения подстроки, является функция `strpbrk()`, имеющая следующий синтаксис:

```
strpbrk($str, $char_list)
```

Функция ищет в строке `$str` любой символ из строки `$char_list` и возвращает подстроку, начиная с первого найденного символа и до конца строки. Если ни один символ не найден, возвращается `FALSE`. В листинге 9.21 приводится пример использования функции `strpbrk()`.

Листинг 9.21. Использование функции `strpbrk()`

```
<?php
  echo strpbrk("Hello world", 'wl'); // llo world
?>
```

9.3. Замена в тексте

Функции замены осуществляют преобразование строк, связанные с заменой одних подстрок другими, а также удалением подстрок. Полный список функций этой группы представлен в табл. 9.4.

Таблица 9.4. *Функции замены в тексте*

Функция	Описание
`str_replace($search, $replace, $str [, &$count])`	Возвращает копию строки `$str`, в которой подстрока `$search` заменена на `$replace`. В необязательном параметре `$count` может быть возвращено количество осуществленных замен. Вместо отдельных строк, параметры `$search` и `$replace` могут содержать массивы строк
`str_ireplace($search, $replace, $str [, &$count])`	Функция `str_ireplace()` аналогична функции `str_replace()` за исключением того, что поиск осуществляется без учета регистра
`substr_replace($str, $replacement, $start [, $length])`	Возвращает копию строки `$str`, в которой подстрока, начинающаяся с символа `$start` и длиной `$length`, заменена подстрокой `$replacement`
`strtr($str, $from, $to)`	Возвращает копию строки `$str`, в которой каждый символ из строки `$from` заменен соответствующим символом из строки `$to`

***Таблица 9.4** (окончание)*

Функция	Описание
strtr($*str*, $*replace_pairs*)	Возвращает копию строки $*str*, в которой замены осуществлены в соответствии с массивом $*replace_pairs* (ключи заменяются значениями массива)
trim($*str* [, $*charlist*])	Удаляет пробельные символы из начала и конца строки $*str*
rtrim($*str* [,$*charlist*])	Удаляет пробельные символы из конца строки $*str*. Необязательный параметр $*charlist* позволяет уточнить список пробельных символов
chop()	Синоним для функции rtrim()
ltrim($*str* [, $*charlist*])	Удаляет пробельные символы из начала строки $*str*. Необязательный параметр $*charlist* позволяет уточнить список пробельных символов

Функция замены str_replace() позволяет заменить подстроку в тексте на другую подстроку и имеет следующий синтаксис:

```
str_replace($search, $replace, $str [, &$count])
```

Функция заменяет в строке $*str* все вхождения подстроки $*search* на $*replace* и возвращает результат замены. Одной из распространенных задач является замена тегов форматирования в стиле bbCode на их HTML-эквиваленты (листинг 9.22).

Листинг 9.22. Замена тегов bbCode на их HTML-эквиваленты

```
<?php
  // Исходная строка
  $postbody = "[b]Это[/b] очень жирный [b]текст[/b].";
  $postbody = str_replace("[b]","<b>",$postbody);
  $postbody = str_replace("[/b]","</b>",$postbody);
  echo $postbody;
?>
```

Результатом работы скрипта в листинге 9.22 будет замена всех символов [b] на , а [/b] на .

Специальным видом замены является удаление подстрок из строки, например следующий скрипт удаляет из текста все теги и (листинг 9.23).

**Листинг 9.23. Удаление тегов и **

```
<?php
  $str = "<b>Это</b> очень жирный <b>текст</b>.";
```

```
  $str = str_replace("<b>", "", $str);
  $str = str_replace("</b>", "", $str);
  echo $str;
?>
```

Помимо подстрок, функция `str_replace()` вместо параметров `$search` на `$replace` может принимать массивы с равным количеством элементов. В строке элемент массива `$search` заменяется соответствующим элементом массива `$replace`. Скрипт в листинге 9.24 аналогичен по результату скрипту из листинга 9.23.

Листинг 9.24. Альтернативное использование функции `str_replace()`

```
<?php
  $str = "<b>Это</b> очень жирный <b>текст</b>.";
  $search = array("<b>", "</b>");
  $replace = array("", "");
  $str = str_replace($search, $replace, $str);
  echo $str;
?>
```

Если необходимо выяснить, сколько замен было осуществлено в строке, функции `str_replace()` следует передать четвертый параметр. После завершения работы в него будет помещено количество осуществленных замен (листинг 9.25).

Листинг 9.25. Подсчет количества замен в строке

```
<?php
  $str = "<b>Это</b> очень жирный <b>текст</b>.";
  $search = array("<b>", "</b>");
  $replace = array("", "");
  $str = str_replace($search, $replace, $str, $number);
  echo $str;
  echo "<br>";
  echo "Осуществлено замен: $number";
?>
```

Результатом работы скрипта из листинга 9.25 будут следующие строки:

```
Это очень жирный текст.
Осуществлено замен: 4
```

Функция `substr_replace()` заменяет в исходной строке одни подстроки на другие и имеет следующий синтаксис:

```
substr_replace($str, $replacement, $start [, $length])
```

Функция возвращает строку `$str`, в которой часть от символа с позицией `$start` и длиной `$length` заменяется строкой `$replacement`. Если аргумент длины `$length` не указан, замена проводится до конца строки. Если значение аргумента `$start` положительно, то отсчет производится от начала строки `$str`, в противном случае — от конца строки. В случае неотрицательного значения `$length`, оно указывает длину заменяемого фрагмента. Если же `$length` отрицательно, то оно обозначает число символов от конца строки `$str` до последнего символа заменяемого фрагмента. В листинге 9.26 демонстрируется использование функции `substr_replace()`.

Листинг 9.26. Использование функции `substr_replace()`

```
<?php
  $var = 'ABCDEFGH:/MNRPQR/';
  echo "Оригинал: $var<br>";

  // Обе следующие строки заменяют всю строку $var на 'bob'
  echo substr_replace($var, 'bob', 0)."<br>";
  echo substr_replace($var, 'bob', 0, strlen($var))."<br>";

  // Вставляет 'bob' в начало $var
  echo substr_replace($var, 'bob', 0, 0)."<br>";

  // Обе следующие строки заменяют 'MNRPQR' in $var на 'bob'
  echo substr_replace($var, 'bob', 10, -1)."<br>";
  echo substr_replace($var, 'bob', -7, -1)."<br>";

  // Удаляет 'MNRPQR' из $var
  echo substr_replace($var, '', 10, -1)."<br>";
?>
```

Функция `strtr()` предназначена для комплексной замены в строке и имеет два варианта синтаксиса:

```
strtr($str, $from, $to)
```

и

```
strtr($str, $replace_pairs)
```

В первом случае функция возвращает копию строки `$str`, в которой каждый символ, присутствующий в строке `$from`, заменяется символом из строки `$to`. Если строки `$from` и `$to` различной длины, то лишние конечные символы у той строки, которая длиннее, игнорируются.

В листинге 9.27 приводится пример преобразования символов верхнего регистра в нижний регистр.

Листинг 9.27. Использование первого варианта функции `strtr()`

```
<?php
  $from = "АБВГДЕЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";
  $to = "абвгдеежзийклмнопрстуфхцчшщъыьэюя";
  $str = "ВСЕ БОЛЬШИЕ БУКВЫ";
  $str = strtr($str, $from, $to);
  echo $str; // все большие буквы
?>
```

Этот скрипт выведет строку, преобразованную в нижний регистр: "все большие буквы".

В случае использования второго варианта синтаксиса, функция `strtr()` возвращает строку `$str`, в которой подстроки, соответствующие ключам, заменяются значениями массива `$replace_pairs`. Сначала функция пытается заменить наибольшие фрагменты исходной строки, при этом не выполняя замену в уже модифицированных частях строки. Таким образом, можно выполнить несколько замен сразу (листинг 9.28).

Листинг 9.28. Использование второго варианта функции `strtr()`

```
<?php
  $str = array('<name1>' => 'М. Кузнецов',
               '<name2>' => 'И. Симдянов');
  $str_out = "Авторы данной книги <name1> и <name2> приветствуют вас!";
  echo strtr($str_out, $str);
?>
```

Результатом выполнения скрипта из листинга 9.28 будет следующая строка:

```
Авторы данной книги М. Кузнецов и И. Симдянов приветствуют вас!
```

Функция `trim()` удаляет символы из начала и конца строки и имеет следующий синтаксис:

```
trim($str [, $charlist])
```

Функция принимает в качестве аргумента `$str` строку и удаляет из нее ведущие и конечные пробельные символы, к которым относят:

- `" "` (ASCII 32 (0x20)), символ пробела;
- `"\t"` (ASCII 9 (0x09)), символ табуляции;
- `"\n"` (ASCII 10 (0x0A)), символ перевода строки;
- `"\r"` (ASCII 13 (0x0D)), символ возврата каретки;
- `"\0"` (ASCII 0 (0x00)), NUL-байт;
- `"\x0B"` (ASCII 11 (0x0B)), вертикальная табуляция.

Зачастую данных символов бывает недостаточно, в этом случае во втором необязательном параметре `$charlist` приводится список символов, которые необходимо считать пробельными.

Для демонстрации работы функции, создадим скрипт, в котором будет контролироваться длина строки до и после удаления из нее пробельных символов (листинг 9.29).

Листинг 9.29. Пример с функцией `trim()`

```
<?php
  $str = "   Hello, world!   "; // строка с пробелами
  $trim_str = trim($str);       // удаляем пробелы
  $str_len = strlen($str);      // определяем длину строки $str
  $trim_str_len = strlen($trim_str); // определяем длину строки $trim_str
  echo " размер исходной строки '$str' = $str_len, <br>
  размер строки '$trim_str' после удаления пробелов = $trim_str_len";
?>
```

Результат выполнения скрипта:

```
размер исходной строки ' Hello, world! ' = 19,
размер строки 'Hello, world!' после удаления пробелов = 13
```

Второй параметр функции `charlist()` бывает полезен при работе с национальными текстами. Например, в Windows для обозначения русского пробела используется символ с ASCII-кодом 160, который по умолчанию не воспринимается функцией `trim()` как пробельный. Для решения этой проблемы следует воспользоваться вызовом `trim()` представленным в листинге 9.30.

> ***Замечание***
>
> Список пробельных символов в параметре `charlist` задается в шестнадцатеричном формате. Последовательность `..` позволяет задать интервал. Так, в листинге 9.31 задается интервал ASCII-кодов от 0 (`\x00`) до 32 (`\x1F`), после чего отдельно задается символ с кодом 160 (`\xA0`).

Листинг 9.30. Настройка функции `trim()` для удаления русского пробела

```
<?php
  echo trim($text,"\x00..\x1F\xA0");
?>
```

9.4. Преобразование регистра

Функции данной группы позволяют преобразовывать регистр строк или первых букв слов. Список функций преобразования регистра представлен в табл. 9.5.

Таблица 9.5. *Список функций преобразования регистра*

Функция	Описание
`strtolower($str)`	Преобразует строку `$str` в нижний регистр
`strtoupper($str)`	Преобразует строку `$str` в верхний регистр
`ucfirst($str)`	Преобразует первый символ строки `$str` в верхний регистр
`ucwords($str)`	Преобразует каждое слово строки в верхний регистр

В листингах 9.31 и 9.32 приводятся примеры использования функций `strtolower()` и `strtoupper()`.

Листинг 9.31. Использование функции `strtolower()`

```
<?php
  echo strtolower("HELLO, WORLD!"); // hello, world!
?>
```

Листинг 9.32. Использование функции `strtoupper()`

```
<?php
  echo strtoupper("Hello, World!"); // HELLO, WORLD!
?>
```

Функция `ucfirst()` производит преобразование первого символа строки `$str` в верхний регистр (листинг 9.33). Если первый символ строки `str` уже находится в верхнем регистре, функция возвращает строку без изменений.

Листинг 9.33. Использование функции `ucfirst()`

```
<?php
  echo ucfirst('hello, world!'); // Hello, world!
?>
```

Функция `ucwords()` производит преобразование первого символа каждого слова строки в верхний регистр. Под словом понимается участок строки, которому предшествует любой пробельный символ: пробел, перевод строки, конец страницы, возврат каретки, горизонтальная и вертикальная табуляции (листинг 9.34).

Листинг 9.34. Использование функции `ucwords()`

```
<?php
  echo ucwords("hello, world!"); // Hello, World!
?>
```

9.5. Работа с HTML-кодом

Так как PHP проектировался как язык, специально предназначенный для Web, среди строковых функций имеется несколько, специально предназначенных для Web-разработки (табл. 9.6).

Таблица 9.6. *Функции для работы с HTML-кодом*

Функция	Описание
`nl2br($str)`	Добавляет перед переводами строки HTML-тег перевода строки ` `
`htmlspecialchars($str [, $quote_style [, $charset [, $double_encode]]])`	Преобразует HTML-теги в строке `$str` в представление, делающее их видимыми в браузере. Необязательный параметр `$quote_style` определяет режим замены кавычек и может принимать три значения: `ENT_NOQUOTES` — двойные и одиночные кавычки остаются без изменений, `ENT_COMPAT` — двойные кавычки преобразуются, одиночные остаются без изменений, `ENT_QUOTES` — одиночные и двойные кавычки преобразуются. Параметр `$charset` задает кодировку текста. Если четвертый необязательный параметр `$double_encode` принимает значение `FALSE`, один раз уже преобразованный HTML-тег не подвергает вторичному преобразованию функции. По умолчанию, этот параметр принимает значение `TRUE`
`htmlspecialchars_decode($str [, $quote_style])`	Выполняет обратную функции `htmlspecialchars()` задачу — преобразует представление HTML-тегов в исходные HTML-теги
`htmlentities($str [, $quote_style [, $charset [, $double_encode]]])`	Функция аналогична `htmlspecialchars()`, однако преобразует не минимальное количество символов для представления HTML-кода, а все возможные символы, для которых предусмотрено представление
`html_entity_decode($str [, $quote_style [,$charset]])`	Выполняет обратную функции `htmlentities()` задачу — преобразует представление HTML-тегов в исходные HTML-теги
`get_html_translation_table ( [$table [, $quote_style]])`	Возвращает таблицу преобразований, используемую функциями `htmlspecialchars()` и `htmlentities()`. Необязательный параметр `$table` может принимать либо константу `HTML_ENTITIES`, либо `HTML_SPECIALCHARS` для `htmlspecialchars()` и `htmlentities()`, соответственно. Параметр `$quote_style` определяет режим замены кавычек
`strip_tags($str [, $allowable_tags])`	Удаляет из строки `$str` HTML-теги, кроме тех, которые указываются в параметре `$allowable_tags`

Для решения задачи замены обычных переводов строк в HTML-переводы строк в PHP существует специальная функция `nl2br()`, которая имеет следующий синтаксис:

```
nl2br($str)
```

Основной особенностью функции `nl2br()` является тот факт, что переводы строк не удаляются, просто к ним прибавляется тег `<br>` (листинг 9.35).

Листинг 9.35. Использование функции `nl2br()`

```
<?php
  $str = '
<?php
  $str = "Строка";
  for($i = 0; $i < strlen($str); $i++)
  {
     echo $str[$i]."<br>";
  }
?>';
  echo $str; // Выводим как есть
  echo "<br><br><br>";
  echo nl2br($str);  // Обрабатываем переменную $str перед выводом
?>
```

Результат работы скрипта из листинга 9.35 представлен на рис. 9.1. Несмотря на то, что текст в переменной `$str` отформатирован, в первом случае она выводится без переводов строк, т. к. браузер переводит строку лишь в том случае, если встретит тег `<br>`.

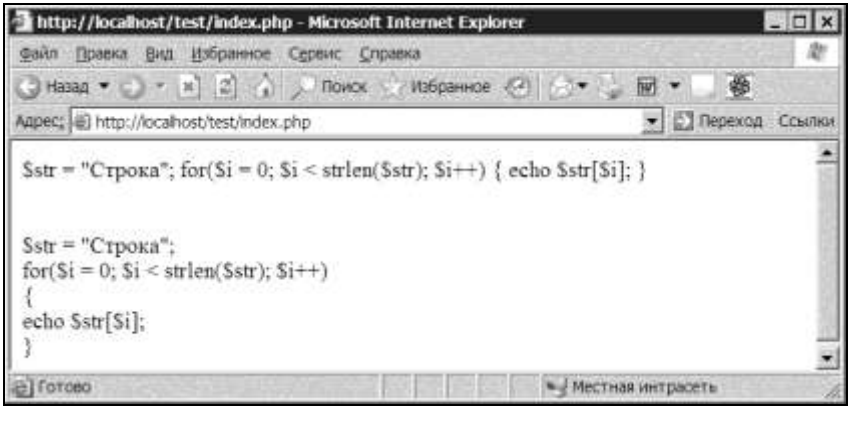


Рис. 9.1. Демонстрация работы функции `nl2br()`

Как видно из рис. 9.1, несмотря на то, что во втором случае переводы строк присутствуют, отступы тем не менее не сохраняются. Это связано с тем, что браузер объединяет

последовательность пробельных символов в один пробел. Для того чтобы сохранить отступы, необходимо либо заключить вывод форматированного текста в теги `<pre>` и `</pre>`, либо заменить все пробельные символы тегом неразрывного пробела ` `.

Особенностью функции `nl2br()` является тот факт, что тег `<br>` вставляется рядом с символом перевода строки, а не заменяет его собой. Такое поведение функции `nl2br()` может быть не удобным, если текст предназначен далее для работы с JavaScript-скриптом. В этом случае вместо функции `nl2br()` удобнее воспользоваться функцией `str_replace()` (листинг 9.36).

> ***ЗАМЕЧАНИЕ***
>
> В UNIX-подобных операционных системах для перевода строк используется символ `\n`, в Windows — последовательность `\r\n`, в Macintosh — `\n\r`. В большинстве случаев Windows-программы корректно обрабатывают и один символ `\n`, но некоторые программы (например, Блокнот) не воспринимают его как полноценный перевод строки и выводят в виде квадратика.

**Листинг 9.36. Замена перевода строк HTML-эквивалентом
**

```
<?php
  $str = "Текст, содержащий\nперевод строки";
  echo str_replace("\n", "<br>", $str);
?>
```

Скрипт, представленный в листинге 9.36, заменяет лишь символ перевода строки `\n`, оставляя незатронутым символ возврата каретки `\r`. Для того чтобы преобразовать как UNIX-, так и Windows-переводы строк, необходимо воспользоваться скриптом, представленным в листинге 9.37.

Листинг 9.37. Усовершенствованный скрипт

```
<?php
  $from = array("\r\n", "\n\r", "\n");
  $to = array_fill(0, 3, "<br>");
  $str = str_replace($from, $to, $str);
?>
```

Функция `htmlspecialchars()` позволяет преобразовать HTML-код в безопасное представление. Применение этой функции гарантирует, что любой введенный пользователем код (PHP, JavaScript и т. д.) будет отображен, но выполняться не будет. Таким образом, функцию следует применять, если необходимо вывести в браузере какой-то код или обезопасить ввод пользователя.

> ***ЗАМЕЧАНИЕ***
>
> Всегда применяйте эту функцию при обработке текстовых сообщений из формы, т. к. в этом случае вы будете надежно защищены от злоумышленников.

Функция `htmlspecialchars()` имеет следующий синтаксис:

```
htmlspecialchars($str [, $quote_style [, $charset [, $double_encode]]])
```

Первый аргумент `$str` — строка, в которой требуется выполнить преобразование. Второй необязательный аргумент `$quote_style` определяет режим обработки двойных и одинарных кавычек и может принимать одну из констант:

- `ENT_COMPAT` — режим по умолчанию, в этом режиме двойные кавычки заменяются символом `"`, при этом одиночные кавычки остаются без изменений;
- `ENT_QUOTES` — в этом режиме преобразуются и двойные, и одиночные кавычки, последние заменяются символом `'`;
- `ENT_NOQUOTES` — в данном режиме двойные и одиночные кавычки остаются без изменений.

Третий необязательный аргумент `$charset` принимает строку с названием кодировки, в которой поступает текст `$str` (по умолчанию ISO-8859-1). Список кодировок приводится в табл. 9.7.

> ***ЗАМЕЧАНИЕ***
>
> Те кодировки, которые отсутствуют в табл. 9.7, не поддерживаются; вместо них используется ISO-8859-1.

Таблица 9.7. *Поддерживаемые функцией* `htmlspecialchars()` *кодировки*

Кодировка	Псевдоним	Описание
ISO-8859-1	ISO8859-1	Западно-европейская Latin-1
ISO-8859-15	ISO8859-15	Западно-европейская Latin-9. Добавляет знак евро, французские и финские буквы к кодировке Latin-1(ISO-8859-1)
UTF-8		8-битная Unicode, совместимая с ASCII
cp866	ibm866, 866	Кириллическая кодировка, применяемая в DOS
cp1251	Windows-1251, win-1251, 1251	Кириллическая кодировка, применяемая в Windows
cp1252	Windows-1252, 1252	Западно-европейская кодировка, применяемая в Windows
KOI8-R	koi8-ru, koi8r	Русская кодировка Интернета
BIG5	950	Традиционный китайский, применяется в основном на Тайване
GB2312	936	Упрощенный китайский, стандартная национальная кодировка
BIG5-HKSCS		Расширенная Big5, применяемая в Гонг-Конге
Shift_JIS	SJIS, 932	Японская кодировка
EUC-JP	EUCJP	Японская кодировка

Простейший пример обработки текстового сообщения `$msg` функцией `htmlspecialchars()` может выглядеть следующим образом:

```
$msg = htmlspecialchars($msg);
```

В листинге 9.38 приведен код HTML-формы для ввода сообщений (index.html).

Листинг 9.38. Код HTML-формы для ввода сообщений (index.html)

```
<form action='handler.php' method='post'>
Сообщение:<br>
<textarea cols='50' rows='5' name='msg'></textarea><br>
<input type='submit' value='Добавить'>
</form>
```

HTML-форма, представленная в листинге 9.38, имеет текстовую область `msg`, кнопку **Добавить**, после нажатия на которую данные отправляются в обработчик handler.php (листинг 9.39).

Листинг 9.39. Код обработчика HTML-формы (handler.php)

```
<?php
  $msg = htmlspecialchars($_POST['msg']);
  echo $msg;
?>
```

Теперь введем в форму для ввода сообщения вместо безобидного текста JavaScript-код (рис. 9.2):

```
<Script Language="JavaScript">
  alert("Приветик!"); // функция вывода в JavaScript
</Script>
```

С JavaScript-кодом форма должна выглядеть так, как это представлено на рис. 9.2.



Рис. 9.2. HTML-форма с введенным JavaScript-кодом

Так как при обработке сообщения использовалась функция `htmlspecialchars()`, ничего страшного не произойдет, обработчик просто выведет текст скрипта, который был набран (рис. 9.3).

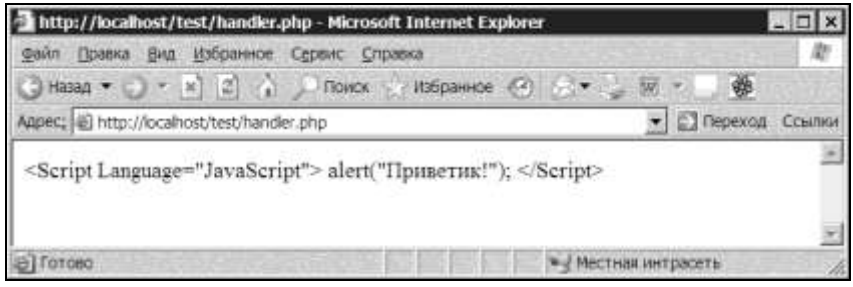


Рис. 9.3. Результат обработки введенного текста при помощи функции `htmlspecialchars()`

Если же введенный пользователем текст не обрабатывать функцией `htmlspecialchars()`, то вместо текста скрипта будет выведен результат его выполнения (рис. 9.4).



Рис. 9.4. Вот что получается без использования функции `htmlspecialchars()`

Злоумышленник может поместить редирект на страницу гостевой книги или форума, а в случае удачи похитить пароли. Иногда HTML-код может многократно пропускаться через функцию `htmlspecialchars()`, и представление для HTML-тегов само может подвергаться отображению:

```
&lt;Script Language=&quot;JavaScript&quot;&gt;
alert(&quot;Приветик!&quot;);   // функция вывода в JavaScript
&lt;/Script&gt;
```

Для того чтобы избежать такого поведения, четвертому параметру функции `htmlspecialchars()` следует передать `FALSE` (листинг 9.40).

Листинг 9.40. Блокирование повторного преобразования HTML-тегов

```
<?php
  $msg = htmlspecialchars($msg, ENT_QUOTES, "cp1251", FALSE);
?>
```

Если необходимо восстановить преобразованные HTML-теги обратно в исходные значения, удобно воспользоваться функцией `htmlspecialchars_decode()`, которая имеет следующий синтаксис:

```
htmlspecialchars_decode($str [, $quote_style])
```

Функция ищет в строке `$str` преобразованные HTML-теги и возвращает строку с их исходными значениями. Второй необязательный параметр `$quote_style` сообщает, должны ли последовательности `"` и `'` преобразовываться, соответственно, в `"` и `'`. Если параметр `$quote_style` принимает значение `ENT_QUOTES`, обе последовательности подвергаются преобразованию, если параметр принимает значение `ENT_COMPAT`, преобразованию подвергается только символ `"`, если параметр принимает значение `ENT_NOQUOTES`, последовательности `"` и `'` не подвергаются преобразованию. В листинге 9.41 приводится пример использования функции `htmlspecialchars_decode()`.

Листинг 9.41. Использование функции `htmlspecialchars_decode()`

```
<?php
  $str = htmlspecialchars("Слово <b>PHP</b> состоит
                           из 3-х <i>символов</i><br>");
  echo htmlspecialchars_decode($str);
  echo $str;
?>
```

Разница между функциями `htmlspecialchars()` и `htmlentities()` заключается в том, что `htmlspecialchars()` преобразует минимально возможное количество символов для отображения HTML-кода, а `htmlentities()` — максимально возможное. В листинге 9.42 одна и та же фраза подвергается преобразованию функций `htmlspecialchars()` и `htmlentities()`.

Листинг 9.42. Использование функции `htmlentities()`

```
<?php
  $str = "Язык программирования '<b>PHP</b>'";

  echo htmlentities($str, ENT_QUOTES, "cp1251");
  echo "<br>";
  echo htmlspecialchars($str, ENT_QUOTES, "cp1251");
?>
```

Если теперь посмотреть исходный код HTML-страницы, можно увидеть, что функция `htmlentities()` преобразовала все русские символы, в то время как `htmlspecialchars()` только те, которые реально используются для формирования HTML-тегов.

> ***Замечание***
>
> В повседневной практике следует отдавать предпочтение функции `htmlspecialchars()`, т. к. преобразованный русский текст не только увеличит размер страницы, но исключит возможность ее индексации роботами поисковых систем.

Выяснить детальное различие между функциями `htmlspecialchars()` и `htmlentities()` может помочь функция `get_html_translation_table()`, которая имеет следующий синтаксис:

```
get_html_translation_table([$table [, $quote_style]])
```

Функция возвращает массив преобразований, используемый функциями `htmlspecialchars()` и `htmlentities()`. Необязательный параметр `$table` может принимать либо константу `HTML_ENTITIES`, либо `HTML_SPECIALCHARS` для `htmlspecialchars()` и `htmlentities()`, соответственно. Параметр `$quote_style` определяет режим замены кавычек. В листинге 9.43 приводится пример использования функции `get_html_translation_table()`.

Листинг 9.43. Использование функции `get_html_translation_table()`

```
<?php
  $htmlspecialchars = get_html_translation_table(HTML_SPECIALCHARS);
  echo "<pre>";
  print_r($htmlspecialchars);
  echo "</pre>";
  $htmlentities = get_html_translation_table(HTML_ENTITIES);
  echo "<pre>";
  print_r($htmlentities);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 9.43 будут следующие дампы массивов преобразования:

```
Array
(
    ["] => &quot;
    [<] => &lt;
    [>] => &gt;
    [&] => &amp;
)
Array
(
```

```
    [ ] =>  
    [Ў] => &iexcl;
    ...
    [ю] => &thorn;
    [я] => &yuml;
    ["] => &quot;
    [<] => &lt;
    [>] => &gt;
    [&] => &amp;
)
```

Полученные массивы могут использоваться в функции `strtr()`, синтаксис которой подробно рассматривается в *разд. 9.3*. Более того, из массивов преобразования могут быть исключены или наоборот добавлены новые элементы.

Вызов функции `strtr()` в листинге 9.44 аналогичен вызову функции `htmlspecialchars()`.

Листинг 9.44. Альтернатива функции `htmlspecialchars()`

```
<?php
  $str = strtr($str, get_html_translation_table(HTML_SPECIALCHARS));
?>
```

Функции `htmlspecialchars()` и `htmlentities()` делают HTML-теги видимыми при выводе информации в окно браузера. В ряде случаев удобнее вообще удалить HTML-теги из текста. Для этого предназначена специальная функция `strip_tags()`, которая имеет следующий синтаксис:

```
strip_tags($str [, $allowable_tags])
```

Функция удаляет из строки `$str` HTML-теги, кроме тех, которые указываются в параметре `$allowable_tags`.

> ***ЗАМЕЧАНИЕ***
>
> Особенность функции заключается в том, что теги не удаляются, а заменяются на пробельные символы, что может быть не всегда удобно.

В листинге 9.45 приводится пример использования функции `strip_tags()`. Для того чтобы был виден результат, перед выводом в окно браузера он пропускается через функцию `htmlspecialchars()`.

Листинг 9.45. Использование функции `strip_tags()`

```
<?php
  $str = "<p>Параграф.</p>
            <!-- Comment -->
            Еще немного текста";
  echo htmlspecialchars(strip_tags($str));
```

```
  echo "<br>";
  echo htmlspecialchars(strip_tags($str, "<p>"));
?>
```

Результатом работы скрипта из листинга 9.45 будут следующие строки:

```
Параграф. Еще немного текста
<p>Параграф.</p> Еще немного текста
```

9.6. Экранирование

Группа функций экранирования предназначена для экранирования специальных символов в строках. Список функций данной группы представлен в табл. 9.8.

Таблица 9.8. *Функции экранирования*

Функция	Описание
`addslashes($str)`	Возвращает копию строки `$str`, в которой перед каждым специальным символом добавлен обратный слеш (\). Экранируются одиночная кавычка ('), двойная кавычка ("), обратный слеш (\) и NUL (символ \0)
`stripslashes($str)`	Функция `stripslashes()` является обратной для `addslashes()`, предназначена для удаления обратных слешей
`addcslashes($str, $charlist)`	Возвращает копию строки `$str`, в которой перед каждым символом из перечисленных в `$charlist` добавлен обратный слеш (\)
`stripcslashes($str)`	Функция `stripcslashes()` удаляет символ обратного слеша и является противоположной функции `addcslashes()`
`quoted_printable_decode ($str)`	Декодирует строку `$str`, закодированную методом quoted printable
`quotemeta($str)`	Возвращает копию строки `$str`, в которой перед каждым символом `. \\ + * ? [ ^ ] ( $ )` помещается обратный слеш (\). Функция удобна для экранирования данных в регулярных выражениях

Функция `addslashes()` удобна для экранирования данных, например, при вставке в базу данных (листинг 9.46).

Листинг 9.46. Использование функции `addslashes()`

```
<?php
  $str = "Функция addslashes экранирует символы \ и '";
  echo addslashes($str); // Функция addslashes экранирует символы \\ и \'
?>
```

До PHP версии 6.0 существовал режим "магических кавычек", который можно было включить в конфигурационном файле php.ini, назначив директиве `magic_quotes_gpc` значение `On`. При включенном режиме магических кавычек, специальные символы в строках, полученных от клиента (HTTP-методами GET, POST, а также через cookies), автоматически экранируются. Если режим отключен, автоматического экранирования не происходит, и разработчику необходимо самостоятельно заботиться об этом. Определить, включен режим магических кавычек или нет, можно было при помощи функции `get_magic_quotes_gpc()`, которая возвращала `TRUE`, если режим включен, и `FALSE` в противном случае. В листинге 9.47 приводится типичный способ обработки данных, полученных от HTML-формы методом POST.

Листинг 9.47. Использование функции `get_magic_quotes_gpc()`

```
<?php
  if (!get_magic_quotes_gpc())
  {
    $_POST['name'] = addslashes($_POST['name']);
  }
?>
```

Как видно из листинга 9.68, если режим магических кавычек отключен, элемент `name` суперглобального массива `$_POST` подвергается обработке функции `addslashes()`, если режим отключен — обработка не требуется и даже вредна, т. к. приведет к появлению лишних слешей в элементе `$_POST['name']`. Из-за того что ряд серверов может включать режим обратных кавычек, ряд — отключать, было принято решение, начиная с PHP 6.0, вообще отказаться от такой возможности: функция `get_magic_quotes_gpc()` должна всегда возвращать `FALSE` для совместимости со старым кодом. В дистрибутиве, который использовался для создания этой книги, функции `get_magic_quotes_gpc()` вообще не оказалось. Впрочем, в любом случае всегда можно добавить ее эмуляцию (листинг 9.48).

Листинг 9.48. Эмуляция функции `get_magic_quotes_gpc()`

```
<?php
  if (!function_exists('get_magic_quotes_gpc'))
  {
    function get_magic_quotes_gpc()
    {
      return FALSE;
    }
  }
?>
```

Как видно из листинга 9.48, при помощи функции `function_exists()` проверяется, существует ли функция `get_magic_quotes_gpc()`, и, если она не обнаружена, создается

заглушка, всегда возвращающая значение FALSE (режим магических кавычек отключен).

Функция addcslashes() экранирует все спецсимволы в стиле языка C и имеет следующий синтаксис:

```
addcslashes($str, $charlist)
```

Функция addcslashes() возвращает копию сроки $str, в которой перед каждым символом из перечисленных в параметре $charlist добавлен обратный слеш (\).

> ***ЗАМЕЧАНИЕ***
>
> Следует проявлять внимание при экранировании символв 0, a, b, f, n, r, t и v. Они будут преобразованы в \0, \a, \b, \f, \n, \r, \t и \v. В C все они являются предопределенными escape-последовательностями, в то время как в PHP только \0 (NUL), \r (возврат каретки), \n (перевод строки) и \t (табуляция).

При передаче $charlist вида "\0..\37" будут экранированы символы с ASCII кодами от 0 до 37. Пример использования функции addcslashes() приведен в листинге 9.49.

Листинг 9.49. Использование функции addcslashes()

```
<?php
  $escaped = addcslashes($not_escaped, "\0..\37!@\177..\377");
?>
```

9.7. Локальные настройки (локаль)

В разных странах в силу сложившихся традиций имеются различия в представлении чисел, денежных сумм, времени и даты. Правила их представления для каждой страны называют *локальными настройками* или сокращенно *локалью*. Каждая операционная система, будь то Windows или одна из UNIX-подобных ОС, поддерживает локальные настройки, которые позволяют изменять представление данных. Такой возможностью обладает и PHP. В табл. 9.9 представлены функции, ответственные за работу с локалью.

Таблица 9.9. *Функции для работы с локальными настройками*

Функция	Описание
Setlocale($category, $locale [, ...])	Устанавливает локальные настройки или возвращает текущую локальную настройку. Функция возвращает значение только что установленной локали или FALSE, если операционная система не поддерживает работу с локальными настройками
localeconv()	Возвращает ассоциативный массив с информацией о числовых и денежных форматах текущей локали. Ключи массива и их значения описываются в табл. 9.11
nl_langinfo($item)	Предоставляет доступ к отдельным элементам локали. Если аргумент $item имеет недопустимое значение, возвращает FALSE. Допустимые значения параметра $item описываются в табл. 9.12—9.16

Локальные настройки управляются шестью переменными окружения, список которых представлен в табл. 9.10.

***Таблица 9.10.** Переменные окружения, управляющие локальной настройкой*

Переменная окружения	Описание
LC_COLLATE	Определяет правила сравнения и, следовательно, сортировки строк для местного алфавита
LC_CTYPE	Определяет правила преобразования одиночных символов для местного алфавита. Позволяет правильно распознавать вид символа: цифра, буква, знак, верхний и нижний регистр
LC_MONETARY	Определяет правила национального представления денежных величин
LC_NUMERIC	Определяет правила национального представления чисел с плавающей точкой
LC_TIME	Определяет правила национального представления даты и времени. Задает именование дней недели, месяцев, формат даты
LC_MESSAGES	Определяет формат информационных, диагностических и интерактивных сообщений операционной системы
LC_ALL	Особая переменная окружения, позволяющая устанавливать значения для всех вышеперечисленных переменных

Функция `setlocale()` позволяет установить или возвратить значение каждой переменной окружения из табл. 9.10. В качестве первого аргумента используется константа, совпадающая по имени с одной из переменных окружения из табл. 9.10. Если в качестве второго аргумента выступает `NULL` или 0, функция возвращает текущее значение локали (листинг 9.50).

> ***Замечание***
>
> Если в качестве второго аргумента функции `setlocale()` передается пустая строка, производится попытка установить в качестве значения локали значение, указанное в переменной окружения `LANG` (эта переменная окружения характерна для UNIX-подобных операционных систем).

Листинг 9.50. Получение текущей локали

```
<?php
  // Получение текущей локали
  echo setlocale(LC_ALL, NULL);
?>
```

К сожалению, значения переменных окружения не стандартизированы и в разных операционных системах могут принимать различные форматы, так для Windows с русскоязычной локалью скрипт из листинга 9.50 вернет значение

```
Russian_Russia.1251
```

Для UNIX-подобных операционных систем локаль выглядит, как правило, следующим образом:

```
ru_RU.UTF-8
ru_RU.KOI8-R
ru_RU.CP1251
```

До точки располагается страна и язык, после точки — кодировка. В листинге 9.51 в качестве примера выставляется локаль для представления денежных величин.

Листинг 9.51. Установка локали для денежных величин

```
<?php
  // Получение текущей локали
  echo setlocale(LC_MONETARY, "ru_RU.CP1251");
?>
```

Функция `setlocale()` может принимать несколько вариантов, которые перечисляются через запятую. В листинге 9.52 осуществляется попытка установки русской локали, при этом функции передается несколько вариантов названия локали.

Листинг 9.52. Попытка установки русской локали

```
<?php
  $loc = setlocale(LC_ALL, 'ru', 'ru_RU', "rus", "russian");
  echo "На этой системе русская локаль имеет имя '$loc'";
?>
```

Функция `localeconv()` возвращает ассоциативный массив с информацией о числовых и денежных форматах. Структура результирующего массива описывается в табл. 9.11.

Таблица 9.11. *Значение элементов результирующего массива функции* `localeconv()`

Ключ	Описание
`decimal_point`	Символ десятичной точки
`thousands_sep`	Разделитель групп (тысячи)
`grouping`	Массив, содержащий количество цифр в группах для числовых данных
`int_curr_symbol`	Международное обозначение валюты, например, RUR
`currency_symbol`	Национальное обозначение валюты, например, руб
`mon_decimal_point`	Символ десятичной точки в денежном формате
`mon_thousands_sep`	Разделитель групп в денежном формате
`mon_grouping`	Массив, содержащий количества цифр в группах для денежных данных

Таблица 9.11 *(окончание)*

Ключ	Описание
positive_sign	Знак для положительных чисел
negative_sign	Знак для отрицательных чисел
int_frac_digits	Количество разрядов после точки (международное)
frac_digits	Количество разрядов после точки (национальное)
p_cs_precedes	TRUE, если currency_symbol записывается перед положительным значением, иначе FALSE
p_sep_by_space	TRUE, если currency_symbol отделяется от положительного значения пробелом, иначе FALSE
n_cs_precedes	TRUE, если currency_symbol записывается перед отрицательным значением, иначе FALSE
n_sep_by_space	TRUE, если currency_symbol отделяется от отрицательного значения пробелом, иначе FALSE
p_sign_posn	Для положительных чисел: • 0 — число и обозначение валюты заключаются в скобки; • 1 — знак записывается перед числом и обозначением валюты; • 2 — знак записывается после числа и обозначения валюты; • 3 — знак записывается перед обозначением валюты; • 4 — знак записывается после обозначения валюты
n_sign_posn	Для отрицательных чисел: • 0 — число и обозначение валюты заключаются в скобки; • 1 — знак записывается перед числом и обозначением валюты; • 2 — знак записывается после числа и обозначения валюты; • 3 — знак записывается перед обозначением валюты; • 4 — знак записывается после обозначения валюты

В листинге 9.53 демонстрируется использование функции `localeconv()`.

Листинг 9.53. Использование функции `localeconv()`

```
<?php
  // Устанавливаем локаль
  setlocale(LC_ALL, "ru_RU.UTF-8");
  // Получаем ассоциативный массив с параметрами
  $arr = localeconv();
  // Выводим дамп массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.53 может быть следующий дамп массива `$arr`:

```
Array
(
    [decimal_point] => ,
    [thousands_sep] =>
    [int_curr_symbol] => RUB
    [currency_symbol] => руб
    [mon_decimal_point] => .
    [mon_thousands_sep] =>
    [positive_sign] =>
    [negative_sign] => -
    [int_frac_digits] => 2
    [frac_digits] => 2
    [p_cs_precedes] => 0
    [p_sep_by_space] => 1
    [n_cs_precedes] => 0
    [n_sep_by_space] => 1
    [p_sign_posn] => 1
    [n_sign_posn] => 1
    [grouping] => Array
       (
            [0] => 3
            [1] => 3
        )
    [mon_grouping] => Array
       (
            [0] => 3
            [1] => 3
        )
)
```

Функция `nl_langinfo()`, в отличие от функции `localeconv()`, позволяет напрямую получить доступ к отдельному элементу локали и имеет следующий синтаксис:

```
nl_langinfo($item)
```

Параметр `$item` может принимать значение одной из констант, перечисленных в табл. 9.12—9.16.

> ***Замечание***
>
> Функция `nl_langinfo()` не реализована для операционной системы Windows.

Таблица 9.12. *Константы категории* `LC_CTYPE`

Константа	Описание
`CODESET`	Имя текущей кодировки

Таблица 9.13. *Константы категории* `LC_MONETARY`

Константа	Описание
`INT_CURR_SYMBOL`	Международное обозначение валюты, например, RUR
`CURRENCY_SYMBOL`	Национальное обозначение валюты, например, руб
`CRNCYSTR`	Синоним для `CURRENCY_SYMBOL`
`MON_DECIMAL_POINT`	Символ десятичной точки
`MON_THOUSANDS_SEP`	Разделитель групп (тысячи)
`MON_GROUPING`	Синоним для `MON_THOUSANDS_SEP`
`POSITIVE_SIGN`	Знак для положительных чисел
`NEGATIVE_SIGN`	Знак для отрицательных чисел
`INT_FRAC_DIGITS`	Количество разрядов после точки (международное)
`FRAC_DIGITS`	Количество разрядов после точки (национальное)
`P_CS_PRECEDES`	`TRUE`, если `CURRENCY_SYMBOL` записывется перед положительным значением, иначе `FALSE`
`P_SEP_BY_SPACE`	`TRUE`, если `CURRENCY_SYMBOL` отделяется от положительного значения пробелом, иначе `FALSE`
`N_CS_PRECEDES`	`TRUE`, если `CURRENCY_SYMBOL` записывется перед отрицательным значением, иначе `FALSE`
`N_SEP_BY_SPACE`	`TRUE`, если `CURRENCY_SYMBOL` отделяется от отрицательного значения пробелом, иначе `FALSE`
`P_SIGN_POSN`	Для положительных чисел: • 0 — число и обозначение валюты заключаются в скобки; • 1 — знак записывается перед числом и обозначением валюты; • 2 — знак записывается после числа и обозначения валюты; • 3 — знак записывается перед обозначением валюты; • 4 — знак записывается после обозначения валюты
`N_SIGN_POSN`	Для отрицательных чисел: • 0 — число и обозначение валюты заключаются в скобки; • 1 — знак записывается перед числом и обозначением валюты; • 2 — знак записывается после числа и обозначения валюты; • 3 — знак записывается перед обозначением валюты; • 4 — знак записывается после обозначения валюты

Таблица 9.14. *Константы категории* `LC_NUMERIC`

Константа	Описание
`DECIMAL_POINT`	Символ десятичной точки
`RADIXCHAR`	Синоним для `DECIMAL_POINT`
`THOUSANDS_SEP`	Разделитель групп (тысячи)
`THOUSEP`	Синоним для `THOUSANDS_SEP`

Таблица 9.15. *Константы категории* `LC_TIME`

Константа	Описание
`ABDAY_1`, ..., `ABDAY_7`	Сокращенное название *n*-го дня недели (`ABDAY_1` соответствует воскресенью)
`DAY_1`, ..., `DAY_7`	Полное название *n*-го дня недели (`DAY_1` соответствует воскресенью)
`ABMON_1`, ..., `ABMON_12`	Сокращенное название *n*-го месяца
`MON_1, ..., MON_12`	Полное название *n*-го месяца
`AM_STR`	Обозначение "до полудня"
`PM_STR`	Обозначение "после полудня"
`D_T_FMT`	Строка формата для даты и времени
`D_FMT`	Строка формата для даты
`T_FMT`	Строка формата для времени
`T_FMT_AMPM`	Строка формата для времени в 12-часовом формате
`ERA`	Эра
`ERA_YEAR`	Год в формате с эрой
`ERA_D_T_FMT`	Строка формата для даты и времени с эрой
`ERA_D_FMT`	Строка формата для даты с эрой
`ERA_T_FMT`	Строка формата для времени с эрой

Таблица 9.16. *Константы категории* `LC_ MESSAGES`

Константа	Описание
`YESEXPR`	Регулярное выражение, при совпадении с которым строка рассматривается как ответ "да"
`NOEXPR`	Регулярное выражение, при совпадении с которым строка рассматривается как ответ "нет"
`YESSTR`	Слово "да"
`NOSTR`	Слово "нет"

В листинге 9.54 приводится пример использования функции `nl_langinfo()` — извлекается имя текущей кодировки.

Листинг 9.54. Использование функции `nl_langinfo()`

```
<?php
  echo nl_langinfo(CODESET);
?>
```

9.8. Форматный вывод

Функции данной группы осуществляют вывод информации в окно браузера и ее форматирование. Список функций приводится в табл. 9.17.

> ***ЗАМЕЧАНИЕ***
>
> `echo` не является функцией и относится к языковым конструкциям, наряду с операторами `if`, `for` и т. п.

Таблица 9.17. *Функции форматного вывода*

Функция	Описание
`echo($arg1 [, ...])`	Выводит аргумент `$arg1` и все последующие в окно браузера
`print($arg)`	Выводит аргумент `$arg` в окно браузера
`printf($format [, $args [, ...]])`	Выводит аргументы `$args` в окно браузера, отформатированные в соответствии со строкой `$format`
`fprintf($handle, $format [, $args [, ...]])`	Выводит аргументы `$args` в открытый файл `$handle`, отформатированные в соответствии со строкой `$format`
`sprintf($format [, $args [, ...]])`	Возвращает строку, состоящую из аргументов `$args`, отформатированных в соответствии со строкой `$format`
`vprintf($format, $args)`	Выводит элементы массива `$args` в окно браузера, отформатированные в соответствии со строкой `$format`
`vfprintf($handle, $format, $args)`	Выводит элементы `$args` в открытый файл `$handle`, отформатированные в соответствии со строкой `$format`
`vsprintf($format, $args)`	Возвращает строку, состоящую из элементов массива `$args`, отформатированных в соответствии со строкой `$format`

***Таблица 9.17** (окончание)*

Функция	Описание
number_format($*number* [, $*decimals* [, $*dec_point*, $*thousands_sep*]])	Возвращает отформатированное число $number с $*decimals* знаками после запятой. При этом в качестве десятичной точки будет использован символ $*dec_point*, а тысячи будет разделять символ $*thousands_sep*
money_format($*format*, $*number*)	Возвращает отформатированную денежную сумму $*number* в соответствии со строкой форматирования $*format* и текущей локалью

Конструкция `echo` более подробно обсуждается в *разд. 5.2*. В листинге 9.55 демонстрируется ее использование.

Листинг 9.55. Использование конструкции `echo`

```
<?php
  echo("Hello, world!<br>");
  echo "Hello, world!<br>";
  echo "Hello", " world", "!", "<br>";
?>
```

Функция `print()` похожа на конструкцию `echo`, однако может принимать только один параметр и всегда возвращает `TRUE` (конструкция `echo` ничего не возвращает). Пример типичного использования функции `print()` приводится в листинге 9.56.

Листинг 9.56. Использование функции `print()`

```
<?php
  print("Hello, world!<br>");
?>
```

Конструкция `echo` и функция `print()` выводят значения переменных и константных выражений как есть. Для форматного вывода предназначены функции семейства `printf()`.

> ***Замечание***
>
> Функция `printf()` отличается от `sprintf()` тем, что первая функция выводит результат непосредственно в окно браузера, а вторая возвращает строку, которую можно сохранить в переменной. Функция `fprintf()` результат помещает в открытый файл, дескриптор которого передается ей в качестве первого параметра.

В качестве первого аргумента, функция `printf()` принимает строку форматирования, а в качестве последующих — переменные, определяемые строкой форматирования (количество аргументов функции не ограничено).

Строка форматирования помимо обычных символов может содержать специальные последовательности символов, начинающиеся со знака `%`, которые называют *определителями преобразования*. В примере, представленном в листинге 9.57, определитель преобразования `%d` подставляет в строку число, которое передается в качестве второго аргумента функции.

Листинг 9.57. Использование определителя `%d`

```
<?php
  printf("Первое число — %d", 26); // Первое число — 26
?>
```

> ***ЗАМЕЧАНИЕ***
>
> Помимо функций `printf()`, `sprintf()` и `fprintf()` существуют их аналоги `vprintf()`, `vsprintf()` и `vfprintf()`, которые принимают не переменное число аргументов, а строку форматирования и массив с переменными для определителей преобразования.

Буква `d`, следующая за знаком `%`, определяет тип аргумента (целое, строка и т. д.), поэтому называется *определителем типа*. В табл. 9.18 представлены определители типа, которые допускаются в строке формата функции `printf()`.

Таблица 9.18. *Определители типа функции* `printf()`

Определитель	Описание
`%b`	Определитель целого, которое выводится в виде двоичного числа
`%c`	Спецификатор символа, используется для подстановки в строку формата символов `char`, например, `'a'`, `'w'`, `'0'`, `'\0'`
`%d`	Спецификатор десятичного целого числа со знаком, используется для подстановки целых чисел, например, 0, 100, –45
`%e`	Спецификатор числа в научной нотации, например, число 1200, в данной нотации записывается как `1.2e+03`, а 0.01 как `1e-02`
`%f`	Спецификатор десятичного числа с плавающей точкой, например, 156.001
`%o`	Спецификатор используется для подстановки в строку формата восьмеричного числа без знака
`%s`	Спецификатор используется для подстановки в строку формата строки
`%u`	Спецификатор десятичного целого числа со знаком, используется для подстановки целых чисел без знака
`%x`	Спецификатор используется для подстановки в строку формата шестнадцатеричного числа без знака (строчные буквы для a, b, c, d, e, f)
`%X`	Спецификатор используется для подстановки в строку формата шестнадцатеричного числа без знака (прописные буквы для A, D, C, D, E, F)
`%%`	Двойная последовательность `%%` используется для обозначения одиночного символа `%` в строке вывода

В листинге 9.58 демонстрируется форматный вывод числа 1024 с использованием разнообразных определителей типа.

Листинг 9.58. Работа с определителями типа

```
<?php
  $number = 1024;
  printf("Двоичное число: %b<br>", $number);
  printf("ASCII-эквивалент: %c<br>", $number);
  printf("Десятичное число: %d<br>", $number);
  printf("Число с плавающей точкой: %f<br>", $number);
  printf("Восьмеричное число: %o<br>", $number);
  printf("Строковое представление: %s<br>", $number);
  printf("Шестнадцатеричное число (нижний регистр): %x<br>", $number);
  printf("Шестнадцатеричное число (верхний регистр): %X<br>", $number);
?>
```

Результат работы скрипта:

```
Двоичное число: 1011011101011
ASCII-эквивалент: л
Десятичное число: 5867
Число с плавающей точкой: 5867.000000
Восьмеричное число: 13353
Строковое представление: 5867
Шестнадцатеричное число (нижний регистр): 16eb
Шестнадцатеричное число (верхний регистр): 16EB
```

Использование определителя типа `x` для шестнадцатеричных чисел удобно при формировании цвета в HTML-тегах (листинг 9.59).

Листинг 9.59. Преобразование цвета из десятичного формата в шестнадцатеричный

```
<?php
  $red = 256;
  $green = 256;
  $blue = 100;
  printf("#%X%X%X", $red, $green, $blue); // #FFFF64
?>
```

Между символом `%` и определителем типа может быть расположен *определитель заполнения*. Определитель заполнения состоит из символа заполнения и числа, которое определяет, сколько символов отводится под вывод. Все не занятые параметром символы будут заполнены символом заполнителя. Так в листинге 9.60 под вывод числа 45 отводится 5 символов, т. к. само число занимает лишь 2 символа, три ведущих символа будут содержать символ заполнения.

Листинг 9.60. Форматирование числового вывода

```
<?php
  echo "<pre>";
  printf("% 4d\n",45); // "   45"
  printf("%04d\n",45); // "00045"
  echo "</pre>";
?>
```

Применение определителя типа `f` позволяет вывести число в десятичном формате. Результатом этого является вывод числа с шестью знаками после запятой. Очень часто требуется вывести строго определенное число символов после запятой, например, для вывода денежных единиц, где обязательным требованием является два знака после запятой. В этом случае прибегают к *определителю точности*, который следует сразу за определителем ширины и представляет собой точку и число символов, отводимых под дробную часть числа (листинг 9.61).

Листинг 9.61. Форматирование чисел с плавающей точкой

```
<?php
  printf("%8.2f\n",1000.45684); // 1000.46
  printf("%.2f\n",12.92869);    // 12.93
?>
```

В первом случае под все число отводятся 8 символов, два из которых отводятся под мантиссу числа. Во втором случае ограничение накладывается только на количество цифр после точки.

Если за знаком `%` следует число (а не пробел или 0), то оно воспринимается как *определитель ширины поля* — число символов, отводимых под выводимое значение. При помощи данного поля можно осуществлять выравнивание строк по правому краю (листинг 9.62).

Листинг 9.62. Форматирование строкового вывода

```
<?php
  echo "<pre>";
  printf("%20s\n","Новости");
  printf("%20s\n","Форум");
  printf("%20s\n","Фотогалерея");
  printf("%20s\n","Каталог продукции");
  printf("%20s\n","Контакты");
  echo "</pre>";
?>
```

Результат работы скрипта:

```
            Новости
              Форум
         Фотогалерея
   Каталог продукции
            Контакты
```

В том случае, если после числа следует вывести знак процента (%), его необходимо передавать в качестве аргумента для определителя типа s или использовать последовательность из двух символов процента (%%). Помещение его в строку форматирования без определителя типа приведет к синтаксической ошибке (листинг 9.63).

Листинг 9.63. Ошибочное форматирование

```
<?php
  // printf("%.2f%s\n",12.92869);   // Ошибка
  printf("%.2f%s\n",12.92869,"%");  // 12.93%
  printf("%.2f%%\n",12.92869,"%");  // 12.93%
?>
```

Для форматирования чисел предназначена специальная фукнция number_format(), которая имеет следующий синтаксис:

```
number_format($number [, $decimals [, $dec_point,  $thousands_sep]])
```

Функция возвращает отформатированное число $number с $decimals знаками после запятой. При этом в качестве десятичной точки будет использован символ $dec_point, а тысячи будет разделять символ $thousands_sep (листинг 9.64).

Листинг 9.64. Использование функции number_format()

```
<?php
  $number = 1234.56;
  // английский формат (по умолчанию)
  $english_format_number = number_format($number);
  // 1,234

  // французский формат
  $nombre_format_francais = number_format($number, 2, ',', ' ');
  // 1 234,56

  $number = 1234.5678;
  // английский формат без разделителей групп
  $english_format_number = number_format($number, 2, '.', '');
  // 1234.57
?>
```

Для вывода денежных величин PHP предоставляет специальную функцию `money_format()`, которая имеет следующий синтаксис:

```
money_format($format, $number)
```

Функция возвращает отформатированную денежную величину `$number` в соответствии со строкой форматирования `$format` и текущей локалью. В листинге 9.65 приводится вывод отформатированной строки для американской и русской локалей.

> ***ЗАМЕЧАНИЕ***
>
> Понятие локали более подробно рассматривается в *разд. 9.7*. Функция `money_format()` вызывает функцию `strfmon()` языка C и работает только в тех операционных системах, где такая функция определена. На практике это означает, что функция `money_format()` не будет работать в операционной системе Windows, зато будет работать в UNIX-подобных операционных системах.

Листинг 9.65. Использование функции `money_format()`

```
<?php
  $number = 100.12;

  setlocale(LC_MONETARY, 'ru_RU');
  echo money_format('%i', $number) . "<br>"; // 100.12 RUB

  setlocale(LC_MONETARY, 'en_US');
  echo money_format('%i', $number) . "<br>"; // USD 100.12
?>
```

Точно так же как и в случае функций семейства `printf()`, строка форматирования помимо обычных символов может содержать специальные последовательности символов, начинающиеся со знака процента `%`. После символа процента `%` могут следовать определители преобразования, представленные в табл. 9.19.

***Таблица 9.19.** Определители преобразования*

Определитель	Описание
`%i`	Используется международный денежный формат
`%n`	Используется национальный денежный формат
`%%`	Вставляет символ `%`

В листинге 9.66 демонстрируется разница между национальным и международными форматами на примере русской локали.

Листинг 9.66. Использование определителей преобразования

```
<?php
  $number = 100.12;

  setlocale(LC_MONETARY, 'ru_RU.CP1251');
  echo money_format('%i', $number) . "<br>"; // 100.12 RUB
  echo money_format('%n', $number) . "<br>"; // 100.12 руб
?>
```

Сразу после символа процента `%` могут быть размещены флаги форматирования из табл. 9.20.

***Таблица 9.20.** Флаги форматирования*

Флаг	Описание
`=f`	Символ `=`, за которым следует еще один символ, задает символ заполнения. По умолчанию используется пробел
`^`	Подавляет разбивку чисел, т. е. вместо 100 000 000 000.00 будет выведено 100000000000.00
`+` или `(`	Задает способ форматирования положительных и отрицательных значений. При использовании `+` будут применяться аналоги символов `+` и `–` из текущей локали. Если указана `(`, отрицательные числа будут заключены в скобки. По умолчанию используется `+`
`!`	Подавляет вывод символа валюты
`-`	Выравнивание полей влево, вместо используемого по умолчанию выравнивания вправо

В листинге 9.67 демонстрируется использование флагов форматирования.

Листинг 9.67. Использование флагов форматирования

```
<?php
  $number = -100.12;
  setlocale(LC_MONETARY, 'ru_RU');
  echo money_format('%i', $number) . "<br>";   // -100.12 RUB
  echo money_format('%!(i', $number) . "<br>"; // (100.12)
?>
```

Сразу после флагов может быть указана ширина поля, определяющая количество символов, выделенных под число (листинг 9.68).

Листинг 9.68. Использование ширины поля

```
<?php
  $number = -100.12;
  setlocale(LC_MONETARY, 'ru_RU');
  echo money_format('%=*20i', $number);  // '         -100.12 RUB'
  echo money_format('%=*-20i', $number); // '-100.12 RUB         '
?>
```

Как видно, ширина поля позволяет выделить под денежную величину определенное количество символов. По умолчанию строка выравнивается по правому краю; если указан флаг "минус" (-), выравнивание осуществляется по левому краю.

После ширины поля может быть указан определитель точности, который начинается с символа диеза (#) и определят количество символов до запятой. Этот определитель удобно использовать, когда последовательность цифр должна быть выведена в колонку так, чтобы запятая у каждой цифры находилась на одном уровне (листинг 9.69).

Листинг 9.69. Использование определителя точности

```
<?php
  $arr = array(123.2, 20035.12, 0.3, 10000.0, 5600.3, 12433.2);
  echo "<pre>";
  foreach($arr as $value)
    echo money_format('%10#5i', $value)."<br>";
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.69 будет следующий столбец цифр:

```
   123.20
 20035.12
     0.30
 10000.00
  5600.30
 12433.20
```

Именно при использовании определителя точности, в работу вступает флаг =. В листинге 9.70 все пробелы, которые сгенерированы при помощи определителя точности, заменяются на символ звездочки (*).

Листинг 9.70. Использование флага =

```
<?php
  $arr = array(123.2, 20035.12, 0.3, 10000.0, 5600.3, 12433.2);
  echo "<pre>";
```

```
  foreach($arr as $value)
    echo money_format('%=*10#5i', $value)."<br>";
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.70 будет следующий столбец цифр:

```
  **123.20
  20035.12
  ****0.30
  10000.00
  *5600.30
  12433.20
```

После определителя точности может быть расположен символ точки с последующей за ней цифрой, которая задает количество знаков после десятичного разделителя. В листинге 9.71 выводятся четыре знака после точки.

Листинг 9.71. Задание количества знаков после точки

```
<?php
  $arr = array(123.2, 20035.12, 0.3, 10000.0, 5600.3, 12433.2);
  echo "<pre>";
  foreach($arr as $value)
    echo money_format('%10#5.4i', $value)."<br>";
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.71 будет следующий столбец цифр:

```
   123.2000
 20035.1200
     0.3000
 10000.0000
  5600.3000
 12433.2000
```

9.9. Преобразование кодировок

Локальные настройки, рассмотренные в *разд. 9.8*, позволяют лишь установить системные настройки. Для преобразования кодировки строк предназначен отдельный набор функций. В табл. 9.21 приводятся стандартные функции, позволяющие осуществлять преобразование кодировок.

Таблица 9.21. *Функции преобразования кодировок*

Функция	Описание
`convert_cyr_string(` `$str, $from, $to)`	Преобразует строку `$str` из кодировки `$from` в кодировку `$to`. Параметры `$from` и `$to` могут принимать следующие значения: • `'k'` — (koi8-r); • `'w'` — (windows-1251); • `'i'` — (iso8859-5); • `'a'` — (x-cp866); • `'m'` — (x-mac-cyrillic)
`convert_uuencode($str)`	Преобразует строку `$str` в формат uuencode. Кодирование uuencode переводит строки (включая двоичные символы) в последовательности печатных (7-битных) ASCII-символов, что позволяет безопасно обмениваться данными через сеть. Закодированные данные примерно на 35% больше оригинала
`convert_uudecode($str)`	Преобразует строку `$str` из формата uuencode в обычный вид
`hebrev($hebrew_text [,` `$max_chars_per_line])`	Преобразует текст на иврите `$hebrew_text` из логической кодировки в визуальную. Необязательный параметр `$max_chars_per_line` позволяет задать количество символов, через которые необходимо вставлять переводы строк (при этом разрыва слова не происходит)
`hebrevc($hebrew_text` `[,` `$max_chars_per_line])`	Преобразует текст на иврите `$hebrew_text` из логической кодировки в визуальную с преобразованием перевода строки в HTML-тег перевода строки ` `. Необязательный параметр `$max_chars_per_line` позволяет задать количество символов, через которые необходимо вставлять переводы строк (при этом разрыва слова не происходит)
`base64_encode($str)`	Кодирует строку `$str` в формат MIME base64
`base64_decode($str)`	Декодирует строку `$str` из формата MIME base64 в обычный текст

По историческим причинам в русскоязычном секторе Интернета используется большое число кодировок. Для преобразования строк из одной кодировки в другую предназначена функция `convert_cyr_string()`, которая имеет следующий синтаксис:

```
convert_cyr_string($str, $from, $to)
```

Функция `convert_cyr_string()` преобразует строку `$str` из кодировки `$from` в кодировку `$to`. Значения аргументов `$from` и `$to` — одиночные символы, определяющие кодировку:

- `'k'` — (koi8-r);
- `'w'` — (windows-1251);

- `'i'` — (iso8859-5);
- `'a'` — (x-cp866);
- `'m'` — (x-mac-cyrillic).

> ***Замечание***
>
> В табл. 9.23 приводится более подробное описание каждой из кодировок.

Пример использования функции `convert_cyr_string()` приведен в листинге 9.72, где перекодируется слово "определяющий" из кодировки windows-1251 в koi8-r и обратно.

Листинг 9.72. Перекодирование строки функцией `convert_cyr_string()`

```
<?php
  $str = "определяющий";
  echo "'$str' в koi8-r является '".
       convert_cyr_string($str, "w", "k").
       "'<br>";
?>
```

И вот результат:

```
'определяющий' в koi8-r является 'ПРТЕДЕМСАЭЙК'
```

Функция `base64_encode()` кодирует данные в формате MIME Base64 и имеет следующий синтаксис:

```
base64_encode($str)
```

Эта кодировка была разработана для передачи двоичных данных через транспортные слои, которые не содержат восьмой бит, к примеру, электронная почта. Данные в формате Base64 занимают примерно на 30% больше места, чем оригинал. В листинге 9.73 демонстрируется использование функции `base64_encode()`.

> ***Замечание***
>
> Более подробно о формате Base64 можно почитать в RFC 2045 (**http://www.ysn.ru/docs/cie/RFC/2045/index.htm**).

Листинг 9.73. Использование функции `base64_encode()`

```
<?php
  echo base64_encode("Hello, world!");     // SGVsbG8sIHdvcmxkIQ==
  echo base64_encode("Программирование"); // z/Du4/Dg7Ozo8O7i4O3o5Q==
?>
```

Иногда возникает задача конвертирования символов из формата Unicode. Наиболее часто эта задача встречается при работе с форматом XML.

Примечание

Unicode — это новый стандарт кодирования символов, когда один символ может кодироваться несколькими байтами. Это позволяет в одной кодовой таблице закодировать все символы основных мировых языков и, таким образом, избежать проблем с разночтением. Стандарт Unicode поддерживается тремя формами: UTF-32, UTF-16 и UTF-8. В сети наиболее популярен UTF-8, т. к. он позволяет предавать данные по сети и не содержит управляющих символов, используемых серверами.

Для преобразования кодировок многобайтовых строк предназначена функция `mb_convert_encoding()`.

```
mb_convert_encoding($str, $to [, $from])
```

Функция возвращает строку `$str`, преобразованную из кодировки `$from` в кодировку `$to`. Рассмотрим блок кода, преобразующий текст в формате UTF-16 в формат Windows-1251 (листинг 9.74).

Замечание

Для работы с расширением, поддерживающим многобайтовые строки (функции, начинающиеся с префикса `mb_`), в PHP 5 необходимо подключить соответствующее расширение. Для этого в конфигурационном файле php.ini (C:\Windows\php.ini) необходимо снять комментарий напротив строки `extension=php_mbstring.dll`.

Листинг 9.74. Использование функции `mb_convert_encoding()`

```
<?php
  $data = mb_convert_encoding($data, "Windows-1251", "UTF-16");
?>
```

Стандартных функций явно не достаточно для преобразования всех возможных кодировок, поэтому было разработано специальное расширение iconv, функции которого представлены в табл. 9.22.

***Таблица 9.22.** Функции преобразования кодировок расширения iconv*

Функция	Описание
`iconv($from, $to, $str)`	Меняет кодировку строки `$str` с `$from` на `$to`
`iconv_get_encoding([$type])`	Возвращает значения внутренних переменных, определяющих кодировки входных, результирующих параметров и внутреннего представления. Параметр `$type` может принимать следующие значения • `'input_encoding'` — кодировка входящей строки; • `'output_encoding'` — кодировка результирующей строки; • `'internal_encoding'` — кодировка внутреннего представления Если параметр `$type` не указан или принимает значение `'all'`, функция возвращает ассоциативный массив, содержащий все три значения

***Таблица 9.22** (продолжение)*

Функция	Описание
`iconv_set_encoding($type, $charset)`	Устанавливает значения внутренних переменных, определяющих кодировки входных, результирующих параметров и внутреннего представления. Параметр `$type` может принимать следующие значения • `'input_encoding'` — кодировка входящей строки; • `'output_encoding'` — кодировка результирующей строки; • `'internal_encoding'` — кодировка внутреннего представления. Параметр `$charaset` определяет имя кодировки. Функция возвращает `TRUE` в случае успешной смены значения и `FALSE` в противном случае
`iconv_mime_encode($field_name, $field_value [, $preferences])`	Формирует и возвращает почтовые заголовки в MIME-представлении. Параметр `$field_name` определяет заголовок, `$field_value` — значение, подвергающееся кодировке. Параметр `$preferences` позволяет управлять процессом кодировки
`iconv_mime_decode($encoded_header [, $mode [, $charset]])`	Декодирует почтовый заголовок `$encoded_header` из MIME-представления. В случае успеха возвращает ассоциативный массив, в случае неудачи — `FALSE`. Параметр `$mode` определяет поведение функции, если ей встречается некорректно определенный MIME-заголовок. Можно указать любую комбинацию следующих битовых констант: • `ICONV_MIME_DECODE_STRICT` — указанный заголовок декодируется в полном соответствии с RFC 2047. Этот режим отключен по умолчанию; • `ICONV_MIME_DECODE_CONTINUE_ON_ERROR` — функция попытается проигнорировать любые грамматические ошибки и продолжить процесс декодирования. Необязательный параметр `$charset` указывает кодировку, в которой следует вернуть результат. Если параметр опущен, по умолчанию, будет использоваться кодировка, определенная во внутренней константе `internal_charset` (чтение и установку которой можно осуществлять соответственно при помощи функций `iconv_get_encoding()` и `iconv_set_encoding()`)

***Таблица 9.22** (окончание)*

Функция	Описание
iconv_mime_decode_headers($*encoded_headers* [, $*mode* [, $*charset*]])	Декодирует почтовый заголовок $*encoded_header* из MIME-представления. В случае успеха возвращает ассоциативный массив, в случае неудачи — FALSE. Каждый ключ элемента возвращаемого массива представляет отдельное имя поля, а сам элемент — его значение, если в заголовке несколько полей с одинаковым именем, они помещаются в подчиненный массив с числовыми индексами. Параметры $*mode* и $*charset* аналогичны по значению функции iconv_mime_decode()
iconv_strlen($*str* [, $*charset*])	Возвращает количество символов в строке $*str* с учетом кодировки $*charset*. Если параметр опущен, по умолчанию, будет использоваться кодировка, определенная во внутренней константе internal_charset (чтение и установку которой можно осуществлять соответственно при помощи функций iconv_get_encoding() и iconv_set_encoding())
iconv_strpos($*str*, $*search* [, $*offset* [, $*charset*]])	Возвращает позицию первого вхождения подстроки $*search* в строку $*str* с учетом кодировки $*charset*. Поиск может осуществляться с позиции $*offset*, если этот необязательный параметр указан
iconv_strrpos($*str*, $*search* [, $*charset*])	Возвращает позицию последнего вхождения подстроки $*search* в строку $*str* с учетом кодировки $*charset*
iconv_substr($*str*, $*offset* [, $*length* [, $*charset*]])	Извлекает из строки $*str* подстроку, начинающуюся с символа $*offset* и длиной $*length*. При этом учитывается кодировка $*charset*
ob_iconv_handler($*contents*, $*status*)	Фильтр для управления буфером вывода, позволяет изменять кодировку "на лету". Более подробно эта функция рассматривается в *главе 17*

Функция iconv() позволяет преобразовать кодировку строки и имеет следующий синтаксис:

iconv($*from*, $*to*, $*str*)

Функция возвращает копию строки $*str*, кодировка которой меняется с $*from* на $*to* (листинг 9.75).

Листинг 9.75. Смена кодировки с CP1251 на KOI8-R

```
<?php
  echo iconv("CP1251", "KOI8-R", "Язык программирования PHP");
?>
```

Полный список поддерживаемых кодировок можно обнаружить по ссылке **http://www.gnu.org/software/libiconv/**. В табл. 9.23 приводятся наиболее распространенные кодировки русского сектора Интернета.

Таблица 9.23. *Кодировки, поддерживающие русский язык*

Кодировка	Описание
CP1251	Кириллическая кодировка Windows, благодаря широкому распространению этой операционной системы получила наиболее широкое распространение в русском секторе Интернета
CP866	Кириллическая кодировка DOS, операционной системы, фактически прекратившей свое существование — в настоящий момент кодировка используется только в командной строке Windows. Кодировка проектировалась с тем расчетом, чтобы не занимать коды, отведенные под псевдографику, интенсивно используемую в интерфейсе DOS
ISO-8859-5	Международная кодировка для кириллицы, из-за отсутствия ряда символов, таких как тире, градус, кавычки-елочки, практически не употребляется
KOI8-R	Кириллическая кодировка, разработанная с расчетом, чтобы при удалении восьмого бита в байте, вместо русских букв получались их фонетические аналоги в английском языке. Кодировка имела широкое распространение на ранних стадиях становления Интернета, когда поддержка русских кодировок в программном обеспечении была минимальна
MacCyrillic	Кириллическая кодировка операционных систем Macintosh и Mac OS
UTF-16	Благодаря тому, что под каждый символ отводятся лишь два байта, кодировка позволяет кодировать все существующие языки. Однако в Интернете UTF-16 не получила распространение из-за того, что ряд символов совпадает со служебными управляющими последовательностями, используемыми в протоколах Интернета. Вместо этой кодировки интенсивно применяется кодировка UTF-8
UTF-8	Наиболее распространенная многобайтная кодировка, позволяющая кодировать все языки. Английский текст, управляющие символы и вообще все коды с номерами до 128 кодируются одним байтом в соответствии с ASCII. Остальные символы кодируются последовательностями от двух до четырех байтов. Такой подход позволяет не только избавиться от недостатков кодировки UTF-8, но и значительно уменьшить объем памяти, требуемый для текста

Расширение iconv имеет три внутренние переменные, управляющие используемыми кодировками:

- `input_encoding` — кодировка входящей строки;

- `output_encoding` — кодировка результирующей строки;
- `internal_encoding` — кодировка внутреннего представления.

Выяснить текущее состояние этих переменных позволяет справочная функция `iconv_get_encoding()`, имеющая следующий синтаксис:

```
iconv_get_encoding([$type])
```

В качестве параметра *$type* функция может принимать имя одной из внутренних переменных. В листинге 9.76 возвращается значение переменной `internal_encoding`.

Листинг 9.76. Использование функции `iconv_get_encoding()`

```
<?php
  echo iconv_get_encoding("internal_encoding");
?>
```

Если в качестве параметра функции `iconv_get_encoding()` передается строка `"all"` или параметр вообще отсутствует, функция возвращает ассоциативный массив, в качестве ключей которого выступают имена внутренних переменных (листинг 9.77).

Листинг 9.77. Получение массива со значениями всех внутренних переменных

```
<?php
  $arr = iconv_get_encoding();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.77 может быть следующий дамп массива `$arr`:

```
Array
(
    [input_encoding] => ISO-8859-1
    [output_encoding] => ISO-8859-1
    [internal_encoding] => ISO-8859-1
)
```

Установить новые значения внутренних переменных можно при помощи функции `iconv_set_encoding()`, которая имеет следующий синтаксис:

```
iconv_set_encoding($type, $charset)
```

В качестве параметра *$type* функция принимает внутреннюю переменную, а в качестве *$charset* — имя кодировки. В листинге 9.78 устанавливается входная кодировка KOI8-R, выходная — CP1251 и кодировка внутреннего представления UTF-8.

Листинг 9.78. Использование функции `iconv_set_encoding()`

```
<?php
  // Устанавливаем внутрение переменные
  iconv_set_encoding("input_encoding", "KOI8-R");
  iconv_set_encoding("output_encoding", "CP1251");
  iconv_set_encoding("internal_encoding", "UTF-8");

  // Контролируем состояние внутренних переменных
  $arr = iconv_get_encoding();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.78 будет следующий дамп массива `$arr`:

```
Array
(
    [input_encoding] => KOI8-R
    [output_encoding] => CP1251
    [internal_encoding] => UTF-8
)
```

Установить начальные значения внутренних переменных расширения iconv можно в конфигурационном файле php.ini в секции `[iconv]` (листинг 9.79).

Листинг 9.79. Фрагмент файла php.ini

```
[iconv]
iconv.input_encoding = KOI8-R
iconv.output_encoding = CP1251
iconv.internal_encoding = UTF-8
```

Расширение iconv предоставляет группу функций, позволяющую упростить работу с почтовыми заголовками. Одной из таких функций является `iconv_mime_encode()`, осуществляющая формирование MIME-представление почтового заголовка и имеющая следующий синтаксис:

```
iconv_mime_encode($field_name, $field_value [, $preferences])
```

Параметр *`$field_name`* определяет заголовок, *`$field_value`* — значение, подвергающееся кодировке. Параметр *`$preferences`* представляет собой ассоциативный массив, ключи которого представлены далее:

- `scheme` — метод кодирования поля. Данное поле может принимать два значения: "B" и "Q", где "B" означает Base64-схему кодирования, а "Q" — схему кодирования,

допускающую вывод кавычек (quoted-printable). По умолчанию параметр принимает значение "B";

- `input-charset` — кодировка, в которой представлены первый и второй параметры функции. Если параметр не указан, используется кодировка из внутренней переменной `internal_charset`;
- `output-charset` — кодировка, используемая для формирования MIME-заголовка. Если параметр не указан, используется кодировка из внутренней переменной `output_charset`;
- `line-length` — максимальная длина строк в заголовке. Результирующий заголовок будет "свернут" в несколько строк согласно RFC 2822, если результирующий заголовок будет длиннее указанной в данном параметре величины. Если данный параметр не указан, он принимает значение, равное 76 символам;
- `line-break-chars` — символ конца строки; если не указан, по умолчанию считается, что строка заканчивается `"\r\n"` (CR LF). Следует помнить, что данная строка всегда интерпретируется как ASCII-строка, независимо от используемой кодировки.

В листинге 9.80 приводится пример формирования почтовых заголовков при помощи функции `iconv_mime_encode()`.

Листинг 9.80. Использование функции `iconv_mime_encode()`

```
<?php
  $preferences = array(
    "input-charset" => "CP1251",
    "output-charset" => "KOI8-R",
    "line-length" => 76,
    "line-break-chars" => "\n"
  );

  $preferences["scheme"] = "Q";
  echo iconv_mime_encode("Subject", "Почтовый заголовок", $preferences);

  echo "<br>";

  $preferences["scheme"] = "B";
  echo iconv_mime_encode("Subject", "Почтовый заголовок", $preferences);
?>
```

Результатом выполнения скрипта из листинга 9.80 будут следующие строки:

```
Subject: =?KOI8-R?Q?=F0=CF=DE=D4=CF=D7=D9=CA=20=DA=C1=C7=CF=CC=CF =D7=CF=CB?=
Subject: =?KOI8-R?B?8M/e1M/X2cog2sHHz8zP18/L?=
```

9.10. Сравнение строк

Сравнение строк можно осуществлять при помощи оператора ==, как и любые другие переменные PHP. При сравнении учитывается регистр строк, поэтому для того, чтобы сравнение строк осуществлялось без учета регистра, необходимо выполнить приведение строк к верхнему регистру при помощи функции `strtoupper()` или к нижнему с использованием функции `strtolower()`, как это показано в листинге 9.81.

> ***Замечание***
>
> Синтаксис функций `strtoupper()` и `strtolower()` более подробно рассматривается в *разд. 9.4*.

Листинг 9.81. Сравнение строк

```
<?php
  $str = "строка текста";
  // Строки равны
  if ($str == "строка текста") echo "Строки равны<br>";
  else echo "Строки не равны<br>";
  // Строки не равны
  if (strtoupper($str) == $str) echo "Строки равны<br>";
  else echo "Строки не равны<br>";
  // Строки равны
  if (strtolower($str) == $str) echo "Строки равны<br>";
  else echo "Строки не равны<br>";
?>
```

Помимо оператора равенства ==, язык программирования PHP предоставляет разработчику широкий набор специальных функций сравнения, список которых представлен в табл. 9.24.

Таблица 9.24. *Функции сравнения строк*

Функция	Описание
`strcmp($str1, $str2)`	Сравнивает строки `$str1` и `$str2` без учета регистра. Возвращает 0, если строки равны, отрицательное число, если `$str1` меньше `$str2`, и положительное, если `$str1` больше `$str2`. Сравниваться могут, в том числе, и бинарные данные
`strcoll($str1, $str2)`	Сравнивает строки `$str1` и `$str2` с учетом текущей локали. Функция возвращает 0, если строки равны, отрицательное число, если `$str1` меньше `$str2`, и положительное, если `$str1` больше `$str2`

Таблица 9.24 *(продолжение)*

Функция	Описание
strcasecmp($*str1*, $*str2*)	Сравнивает строки $*str1* и $*str2* без учета регистра. Функция возвращает 0, если строки равны, отрицательное число, если $*str1* меньше $*str2*, и положительное, если $*str1* больше $*str2*. Сравниваться могут, в том числе, и бинарные данные
strncmp($*str1*, $*str2*, $*len*)	Сравнивает первые $*len* символов строк $*str1* и $*str2* без учета регистра. Функция возвращает 0, если строки равны, отрицательное число, если $*str1* меньше $*str2*, и положительное, если $*str1* больше $*str2*. Сравниваться могут, в том числе, и бинарные данные
strncasecmp($*str1*, $*str2*, $*len*)	Сравнивает первые $*len* символов строк $*str1* и $*str2* с учетом регистра. Функция возвращает 0, если строки равны, отрицательное число, если $*str1* меньше $*str2*, и положительное, если $*str1* больше $*str2*. Сравниваться могут, в том числе, и бинарные данные
strnatcmp($*str1*, $*str2*)	Осуществляет "естественное" сравнение строк $*str1* и $*str2* без учета регистра. Функция возвращает 0, если строки равны, отрицательное число, если $*str1* меньше $*str2*, и положительное, если $*str1* больше $*str2*
strnatcasecmp($*str1*, $*str2*)	Осуществляет "естественное" сравнение строк $*str1* и $*str2* с учетом регистра. Функция возвращает 0, если строки равны, отрицательное число, если $*str1* меньше $*str2*, и положительное, если $*str1* больше $*str2*
strspn($*str1*, $*str2* [, $*start* [, $*length*]])	Возвращает количество символов в начале строки $*str1*, соответствующих символам из строки $*str2*. Поиск может начинаться не с нулевого символа начала строки, а с позиции, заданной параметром $*start* (при отрицательном значении отсчет ведется с конца строки) на протяжении $*length* символов
strcspn($*str1*, $*str2* [, $*start* [, $*length*]])	Возвращает количество символов в начале строки $*str1*, не соответствующих символам из строки $*str2*. Поиск может начинаться не с нулевого символа начала строки, а с позиции, заданной параметром $*start* (при отрицательном значении отсчет ведется с конца строки) на протяжении $*length* символов
substr_compare($*main_str*, $*str*, $*offset* [, $*length* [, $*case_insensitivity*]])	Сравнивает строку $*main_str*, начиная с позиции $*offset*, со строкой $*str*. Сравниваются максимум $*length* символов. По умолчанию сравнение осуществляется без учета регистра, однако, если пятый параметр $*case_insensitivity* принимает значение TRUE, регистр учитывается

Таблица 9.24 *(окончание)*

Функция	Описание
similar_text($*str1*, $*str2* [, &$*percent*])	Сравнивает строки $*str1* и $*str2* по алгоритму Оливера и возвращает количество совпадающих в строках символов. Если указывается третий необязательный параметр $*percent*, в него заносится степень схожести строк в процентах
levenshtein($*str1*, $*str2* [, $*cost_ins*, $*cost_rep*, $*cost_del*])	Вычисляет расстояние Левенштейна между строками $*str1* и $*str2*. Под расстоянием Левенштейна понимается минимальное число символов, которое требовалось бы заменить, вставить или удалить для того, чтобы превратить строку $*str1* в $*str2*. Необязательные параметры $*cost_ins*, $*cost_rep* и $*cost_del* позволяют соответственно задать стоимость операции вставки, замены и удаления

Функция `strcmp()` сравнивает две строки и имеет следующий синтаксис:

```
strcmp($str1, $str2)
```

В качестве результата могут возвращаться следующие значения:

- 0, если строки полностью совпадают;
- 1, если строка $*str1* лексикографически больше строки $*str2*;
- –1, если строка $*str1* лексикографически меньше строки $*str2*.

В листинге 9.82 демонстрируется использование функции `strcmp()`.

Листинг 9.82. Сравнение строк функцией `strcmp()`

```
<?php
  $str1 = "ttt";
  $str2 = "ttttttttttt";
  echo "<br>Результат сравнения($str1 , $str2): ";
  echo strcmp(str1,str2)); echo("<br>";
  echo "<br>Результат сравнения($str2 , $str1): ";
  echo strcmp(str2,str1)); echo("<br>";
  echo "<br>Результат сравнения($str1 , $str1): ";
  echo strcmp(str1,str1);
?>
```

Результат работы скрипта из листинга 9.82 выглядит следующим образом:

```
Результат сравнения(ttt , ttttttttttt): -1
Результат сравнения(ttttttttttt , ttt): 1
Результат сравнения(ttt , ttt): 0
```

Если требуется сравнивать строки без учета регистра, можно воспользоваться функцией `strcasecmp()`, синтаксис которой аналогичен синтаксису функции `strcmp()`, но сравнение производится без учета регистра символов.

Функция `strnatcmp()` производит "естественное" сравнение строк, т. е. она сравнивает строки так, как их сравнивал бы человек. Функция имеет следующий синтаксис:

```
strnatcmp($str1, $str2)
```

Если, к примеру, сравниваются файлы с названиями pict1.gif, pict20.gif, pict2.gif, pict10.gif, то обычное сравнение приведет к следующему их расположению:

```
pict1.gif
pict10.gif
pict2.gif
pict20.gif
```

Естественная же сортировка даст результат, который нам более привычен:

```
pict1.gif
pict2.gif
pict10.gif
pict20.gif
```

В листинге 9.83 приводится пример сортировки массива имен файлов при помощи функции `usort()`, которая осуществляет сортировку с применением пользовательских функций. В качестве таких функций выступают `strcmp()` и `strnatcmp()`.

Листинг 9.83. Естественное сравнение строк

```
<?php
  $arr = $natarr = array("pict10.gif",
                         "pict2.gif",
                         "pict20.gif",
                         "pict1.gif");
  echo "обычная сортировка:<br>";
  usort($arr, "strcmp");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
  echo "<br>естественная сортировка:<br>";
  usort($natarr, "strnatcmp");
  echo "<pre>";
  print_r($natarr);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 9.83 представлен на рис. 9.5.

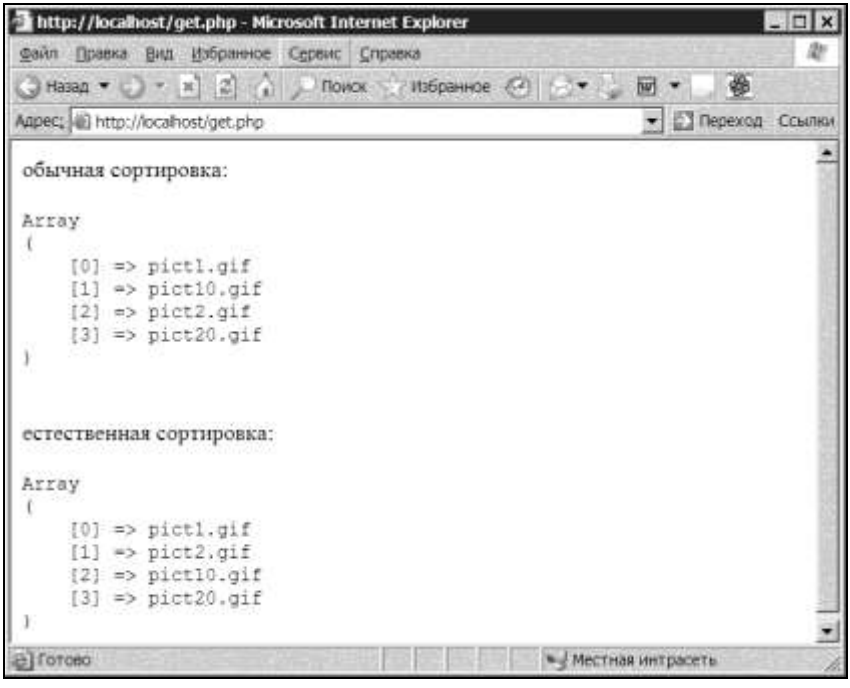


Рис. 9.5. Обычная и естественная сортировка

Функция `similar_text()` определяет схожесть двух строк по *алгоритму Оливера* и имеет следующий синтаксис:

```
similar_text($str1, $str2 [, &$percent])
```

Функция возвращает число символов, совпавших в строках `$str1` и `$str2`. Третий необязательный параметр `$percent` передается по ссылке, и в нем сохраняется процент совпадения строк (листинг 9.84).

Листинг 9.84. Определение схожести строк

```
<?php
  $first  = "Hello, world!";
  $second = "Hello!";
  $var = similar_text($first, $second, &$tmp); // $tmp передаем по ссылке
  echo "Результат выполнения функции
        similar_text() для строк <i>$first</i>
        и <i>$second</i> в количестве символов:
        <br>$var<br> в процентах:<br>$tmp";
?>
```

Результат работы скрипта из листинга 9.84 представлен на рис. 9.6.

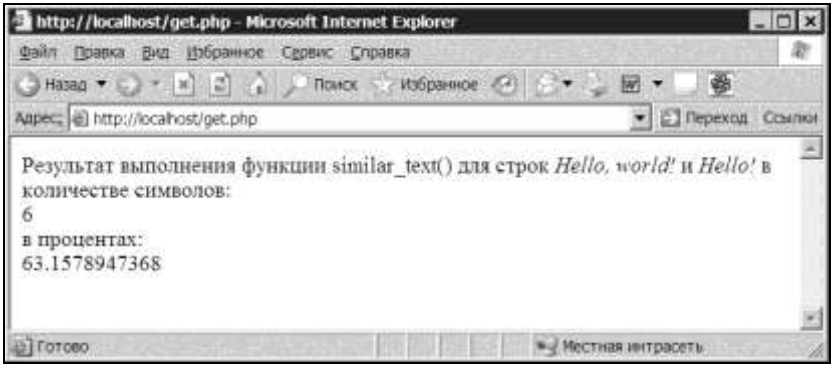


Рис. 9.6. Сравнение строк при помощи функции `similar_text()`

Функция `levenshtein()` выполняет определение *расстояние Левенштейна* двух строк и имеет следующий синтаксис:

```
levenshtein($str1, $str2 [, $cost_ins, $cost_rep, $cost_del])
```

Под *расстоянием Левенштейна* понимается минимальное число символов, которое требовалось бы заменить, вставить или удалить для того, чтобы превратить строку `$str1` в `$str2`.

В листинге 9.85 демонстрируется пример использования функции `levenshtein()`.

Листинг 9.85. Использование функции `levenshtein()`

```
<?php
  $str1 = "Hello, world!";
  $str2 = "Hello!";
  echo levenshtein($str1, $str2); // 7
?>
```

Необязательные параметры `$cost_ins`, `$cost_rep` и `$cost_del` позволяют соответственно задать стоимость операции вставки, замены и удаления (листинг 9.86). Естественно, функция в этом случае становится менее быстродействующей.

> ***Замечание***
>
> Если используются три дополнительных параметра, результат функции часто называют *интегральным показателем сложности трансформации* (ИПСТ).

Листинг 9.86. Второй вариант использования функции `levenshtein()`

```
<?php
  $str1 = "Hello, world!";
  $str2 = "Hello!";
  echo levenshtein($str1, $str2, 3, 3, 3);
?>
```

Число 21 получилось потому, что ИПСТ равен произведению количества символов, необходимых для замены (как видно из листинга 9.86, таких символов необходимо 7), на стоимость одной из операций. В этом примере, поскольку стоимость одинакова, не имеет значения, какую операцию брать. В том случае, если стоимости различны, при вычислении ИПСТ берется наибольшая стоимость, т. е. если будет записано

```
$var = levenshtein($str1, $str2, 2, 3, 6);
```

то функция вернет значение 42.

9.11. Хранение данных

Строки очень часто используются для хранения разнообразных данных. Поэтому PHP имеет целый арсенал функций для упаковки данных в строки и извлечению их. В табл. 9.25 приводится список функций данной группы.

Таблица 9.25. *Функции упаковки данных*

Функция	Описание
`bin2hex($str)`	Осуществляет побайтовое преобразование символов строки `$str` в шестнадцатеричный формат
`pack($format[, $args [, ...]])`	Упаковывает параметры `$args` в двоичную строку. Формат параметров и их количество определяется первым параметром `$format`
`unpack($format, $data)`	Осуществляет обратное функции `pack()` преобразование
`serialize($value)`	Осуществляет упаковку массива или объекта в строку `$value`
`unserialize($str)`	Осуществляет распаковку массива или объекта из строки `$str`, созданной функцией `serialize()`

Функция `bin2hex()` производит побайтовое преобразование символов строки в шестнадцатеричный формат. В качестве единственного параметра функция принимает строку и возвращает строковое шестнадцатеричное представление символов, содержащихся в этой строке. В листинге 9.87 перекодирована в шестнадцатеричный вид фраза `"Hello, php!"`.

Листинг 9.87. Перекодирование в шестнадцатеричный вид

```
<?php
  echo bin2hex("Hello, php!"); // 48656c6c6f2c2070687021
?>
```

Функция `pack()` осуществляет пакетирование данных в двоичную строку и имеет следующий синтаксис:

```
pack($format [, $args [, ...]])
```

Функция упаковывает заданные в ее параметре аргументы в бинарную строку. Формат параметров и их количество задается параметром `$format`, при помощи следующих спецификаторов форматирования:

- `a` (строка, свободные места в поле заполняются символом с кодом 0);
- `A` (строка, свободные места заполняются пробелами);
- `h` (шестнадцатеричная строка, младшие разряды в начале);
- `H` (шестнадцатеричная строка, старшие разряды в начале);
- `c` (знаковый байт);
- `C` (беззнаковый байт);
- `s` (знаковое короткое целое);
- `S` (беззнаковое короткое целое);
- `n` (беззнаковое целое, 16 битов, старшие разряды в конце);
- `v` (беззнаковое целое, 16 битов, младшие разряды в конце);
- `i` (знаковое целое);
- `I` (беззнаковое целое);
- `l` (знаковое длинное целое, 32 бита);
- `L` (беззнаковое длинное целое);
- `N` (беззнаковое длинное целое, 32 бита, старшие разряды в конце);
- `V` (беззнаковое целое, 32 бита, младшие разряды в конце);
- `f` (число с плавающей точкой);
- `d` (число двойной точности);
- `x` (символ с нулевым кодом);
- `X` (возврат назад на 1 байт);
- `@` (заполнение нулевым кодом до заданной абсолютной позиции).

После каждого спецификатора может стоять число, которое говорит о том, сколько символов будет обработано данным спецификатором. Для форматов `a`, `A`, `h` и `H` это число задает количество символов, которые будут помещены в бинарную строку из тех, что находятся в параметре-строке (фактически, определяется размер поля вывода строки). Если используется спецификатор `@`, то в этом случае определяется абсолютная позиция, в которую будут помещены данные. Для всех остальных спецификаторов следующие за ними числа задают количество аргументов, на которые распространяется действие данного формата. Вместо числа можно указать символ `*`, в этом случае спецификатор действует на все оставшиеся данные. Функция возвращает упакованные данные в шестнадцатеричном формате (листинг 9.88).

Листинг 9.88. Паковка данных в двоичный формат

```
<?php
  $bin = pack("nvn*",0x5722,0x1148, 65, 66); // запаковываем
```

```
  $var = bin2hex($bin); // перекодируем из шестнадцатеричного формата
  echo($var);
?>
```

Результатом работы скрипта из листинга 9.88 будет такая последовательность:

```
0x57, 0x22, 0x48, 0x11, 0x00, 0x41, 0x00, 0x42
```

Согласно заданному формату `nvn*`, первое число возвращается как беззнаковое целое со старшими разрядами в конце, второе тоже как беззнаковое целое, только в конце — младшие разряды (поэтому получилось `0x48, 0x11`, а не `0x11, 0x48`), и все остальное до конца возвращается как беззнаковое целое со старшими разрядами в конце.

Следующая пара функций `serialize()` и `unserialize()` позволяют осуществлять упаковку и распаковку, соответственно, массивов и объектов.

> ***ЗАМЕЧАНИЕ***
>
> Сериализация впервые появилась в объектно-ориентированных библиотеках (первой из которых была MFC), потом сериализация стала появляться в объектно-ориентированных языках (Java). Идея сериализации заключается в том, что объекты и массивы очень сложны по своей структуре и на сохранение их путем перебора каждого элемента требуется значительный объем кода — самым простым решением является сохранение таких структур в виде единой закодированной последовательности — байт-коде. В PHP функции сериализации упаковывают данные не в виде байт-кода, а в виде строки.

Синтаксис функции `serialize()` можно представить следующим образом:

```
serialize($value)
```

В качестве аргумента функция принимает массив или объект, возвращая его в виде закодированной строки. Симметричная ей функция `unserialize()` принимает в качестве аргумента закодированную строку, возвращая массив или объект.

```
unserialize($str)
```

В листинге 9.89 демонстрируется использование функций `serialize()` и `unserialize()`.

Листинг 9.89. Использование функций `serialize()` и `unserialize()`

```
<?php
  $poll[0] = 23;
  $poll[1] = 45;
  $poll[2] = 34;
  $poll[3] = 2;
  $poll[4] = 12;
  // Упаковываем массив в строку
  $str = serialize($poll);
  echo $str."<br>";
  // Извлекаем массив из строки
  $arr = unserialize($str);
  print_r($arr);
?>
```

Результатом работы скрипта из листинга 9.88 будут следующие строки:

```
a:5:{i:0;i:23;i:1;i:45;i:2;i:34;i:3;i:2;i:4;i:12;}
Array
(
    [0] => 23
    [1] => 45
    [2] => 34
    [3] => 2
    [4] => 12
)
```

Последние две функции часто применяются при работе с файлами, что будет продемонстрировано в соответствующей главе.

9.12. Работа с путями к файлам и каталогам

Функции данной группы позволяют облегчить работу с путями к файлам и каталогам (табл. 9.26).

***Таблица 9.26.** Функции для работы с путями к файлам и каталогам*

Функция	Описание
`pathinfo($path [, $options])`	Принимает путь к файлу `$path` и возвращает ассоциативный массив, в элементах которого сохраняются каталог, в котором расположен файл, имя файла, его расширение и имя файла без расширения. Если указан необязательный параметр `$options`, функция возвращает строковое значение. Параметр `$options` может принимать следующие константы: • `PATHINFO_DIRNAME` — путь к файлу (каталог); • `PATHINFO_BASENAME` — имя файла; • `PATHINFO_EXTENSION` — расширение файла; • `PATHINFO_FILENAME` — имя файла без расширения
`realpath($path)`	Возвращает абсолютный путь (путь от начала диска) для файла `$path`
`basename($path [, $suffix])`	Извлекает из пути `$path` имя файла. Необязательный параметр `$suffix` позволяет исключить расширение из результирующей строки
`dirname($path)`	Извлекает из пути `$path` каталог, в котором расположен файл

Функция `pathinfo()` принимает путь к файлу `$path` и возвращает ассоциативный массив, в элементах которого сохраняются каталог расположения файла, имя файла, его расширение и имя без расширения (листинг 9.90).

Листинг 9.90. Работа с функцией `pathinfo()`

```
<?php
  $path = pathinfo("C:/www/htdocs/index.html");

  echo "<pre>";
  print_r($path);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.90 будет следующий дамп массива `$path`:

```
Array
(
    [dirname] => C:/www/htdocs
    [basename] => index.html
    [extension] => html
    [filename] => index
)
```

Использовать массив не всегда удобно, поэтому можно вернуть лишь один элемент, передав в качестве второго параметра функции `pathinfo()` соответствующую константу из табл. 9.26 (листинг 9.91).

Листинг 9.91. Использование второго параметра функции `pathinfo()`

```
<?php
  $path = "C:/www/htdocs/index.html";
  echo pathinfo($path, PATHINFO_BASENAME); // index.html
?>
```

Для того чтобы опередить абсолютный путь к файлу, необходимый для функции `pathinfo()`, можно воспользоваться функцией `realpath()`, которая имеет следующий синтаксис:

```
realpath($path)
```

Функция принимает относительный путь *`$path`* и возвращает абсолютный. При работе с функцией следует помнить, что точкой (`.`) обозначается текущий каталог, а двумя точками (`..`) — каталог на уровень выше (листинг 9.92).

> ***ЗАМЕЧАНИЕ***
>
> При разработке Web-приложений чаще требуется путь к текущей странице от DocumentRoot, который можно узнать, обратившись к элементу суперглобального массива `$_SERVER['PHP_SELF']`.

Листинг 9.92. Использование функции `realpath()`

```
<?php
  echo realpath("."); // Путь к текущему каталогу
  echo realpath("../index.php"); // Путь к файлу index.php уровнем выше
?>
```

Для извлечения имени файла существует отдельная функция `basename()`, которая имеет следующий синтаксис:

```
basename($path [, $suffix])
```

Функция принимает путь `$path` и необязательное расширение файла `$suffix` и возвращает имя файла, чей путь был передан в качестве параметра. Если имя файла имеет расширение `$suffix`, оно будет отброшено (листинг 9.93).

Листинг 9.93. Использование функции `basename()`

```
<?php
  $path = "/home/httpd/html/index.html";
  echo basename($path);          // index.html
  echo basename($path, ".html"); // index
?>
```

Функция `dirname()` извлекает из пути каталог и имеет следующий синтаксис:

```
dirname($path)
```

Функция принимает путь `$path` и возвращает только каталог (листинг 9.94).

Листинг 9.94. Использование функции `dirname()`

```
<?php
  $path = "c:/www/html/index.html";
  echo dirname($path); // c:/www/html
?>
```

9.13. Объединение и разбиение строк

Функции данной группы осуществляют объединение подстрок в единую строку, а также разбиение строки на отдельные подстроки (табл. 9.27).

***Таблица 9.27.** Функции объединения и разбиения строк*

Функция	Описание
`str_repeat($str, $number)`	Создает новую строку, состоящую из `$number` повторов строки `$str`

Таблица 9.27 *(продолжение)*

Функция	Описание
str_pad($*str*, $*length* [, $*pad* [, $*pad_type*]])	Увеличивает размер строки $*str* до $*length* символов. По умолчанию новые символы заполняются пробелами, однако символ-заполнитель можно задать в необязательном параметре $*pad*. Необязательный параметр $*pad_type* определяет, с какой стороны добавляются новые символы, и может принимать следующие значения: • STR_PAD_RIGHT — новые символы добавляются в конец строки (по умолчанию); • STR_PAD_LEFT — новые символы добавляются в начало строки; • STR_PAD_BOTH — новые символы добавляются как в конец, так и в начало
chunk_split($*str* [, $*chunklen* [, $*end*]])	Возвращает копию строки $*str*, в которой через каждые $*chunklen* (по умолчанию 76) символов вставляется подстрока $*end* (по умолчанию \r\n)
str_split($*str* [, $*split*])	Преобразует строку $*str* в массив. По умолчанию в качестве элементов массива выступают символы строки. Необязательный параметр $*split* позволяет задать произвольную длину подстрок в символах, образующих элементы массива
strtok($*str*, $*token*)	Разбивает строку $*str* на подстроки, используя в качестве разделителя подстрок последовательность $*token*
str_word_count($*str* [, $*format* [, $*charlist*]])	Если указывается единственный параметр $*str*, возвращается количество слов в строке. Если указывается необязательный параметр $*format*, отдельные слова строки возвращаются в виде массива. Параметр $*format* может принимать следующие значения: • 1 — возвращается массив, содержащий все слова, входящие в строку $*str*; • 2 — возвращается массив, индексами которого являются позиции в строке $*str*, а значениями — соответствующие слова Необязательный параметр $*charlist* позволяет задать дополнительные символы, из которых будет состоять слово
explode($*delimiter*, $*str* [, $*limit*])	Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки $*str*, находящемуся между разделителями, определяемыми аргументом $*delimiter*. Необязательный параметр $*limit* определяет максимальное количество элементов в массиве
implode($*delimiter*, $*arr*)	Объединяет элементы массива строк $*arr* в единую строку, разделяя элементы подстрокой, заданной в параметре $*delimiter*
join()	Синоним для функции implode()

***Таблица 9.27** (окончание)*

Функция	Описание
wordwrap($*str* [, $width [, $*break* [, $*cut*]]])	Вставляет в строке $*str* символы $*break* (по умолчанию \n) через каждые $*width* символов (по умолчанию 75). При этом символ $*break* не разрывает слово, однако если необязательный параметр $*cut* принимает значение TRUE, допускается разрыв слова
parse_str($*str* [, &$*arr*])	Разбивает строку $*str*, имеющую формат строки с GET-запросом, и преобразует отдельные GET-параметры в переменные или в элементы массива $*arr* (если указан второй параметр)
sscanf($*str*, $*format* [, ...])	Разбирает строку $*str* в соответствии со строкой форматирования $*format*. Если передано только два аргумента, функция возвращает массив, в противном случае результаты помещаются в последующие параметры

Функция str_repeat() создает новую строку, состоящую из $number повторов строки $str (листинг 9.95).

Листинг 9.95. Использование str_repeat()

```
<?php
  echo str_repeat("Hello!",3); // Hello!Hello!Hello!
?>
```

Функция str_pad() увеличивает размер строку до определенной длины и имеет следующий синтаксис:

```
str_pad($str, $length [, $pad [, $pad_type]])
```

Функция увеличивает размер строки $*str* до $*length* символов. По умолчанию новые символы заполняются пробелами, однако символ-заполнитель можно задать в необязательном параметре $*pad*. Необязательный параметр $*pad_type* определяет, с какой стороны добавляются новые символы, и может принимать следующие значения:

- STR_PAD_RIGHT — новые символы добавляются в конец строки (по умолчанию);
- STR_PAD_LEFT — новые символы добавляются в начало строки;
- STR_PAD_BOTH — новые символы добавляются как в конец, так и в начало.

В листинге 9.96 демонстрируется использование функции str_pad().

Листинг 9.96. Использование функции str_pad()

```
<?php
  $input = "Alien";
  echo str_pad($input, 10);                    // выводит "Alien     "
```

```
  echo str_pad($input, 10, "-=", STR_PAD_LEFT);  // выводит "-=-=-Alien"
  echo str_pad($input, 10, "_", STR_PAD_BOTH);   // выводит "__Alien___"
  echo str_pad($input, 6 , "___");               // выводит "Alien_"
?>
```

Функция `chunk_split()` возвращает фрагмент строки и имеет следующий синтаксис:

```
chunk_split($str [, $chunklen [, $end]])
```

Функция возвращает копию строки `$str`, в которой через каждые `$chunklen` (по умолчанию 76) символов вставляется подстрока `$end` (по умолчанию `\r\n`). В листинге 9.97 приводится пример использования функции `chunk_split()` — строка разбивается на подстроки длиной 10 символов, каждая строка отделена от другой строки при помощи тега `<br>`.

Листинг 9.97. Использование функции `chunk_split()`

```
<?php
  $text = "Длинная строка, которая при помощи функции chunk_split() будет разбита на части";
  echo chunk_split($text, "10", "<br>");
?>
```

Результат работы скрипта из листинга 9.97 представлен на рис. 9.7.



Рис. 9.7. Результат работы фукцнии `chunk_split()`

Функция `str_split()` преобразует строку в массив и имеет следующий синтаксис:

```
str_split($str [, $split])
```

В качестве первого параметра функция принимает преобразуемую строку `$str`. По умолчанию в качестве элементов массива выступают символы строки. Необязательный параметр `$split` позволяет задать произвольную длину подстрок в символах, образующих элементы массива.

Если `$split` меньше 1, возвращается `FALSE`. Если `$split` больше длины строки `$str`, вся строка будет возвращена в первом и единственном элементе массива. Пример использования функции `str_split()` приводится в листинге 9.98.

Листинг 9.98. Использование функции `str_split()`

```
<?php
  $str = "Потрошим строку";
  $arr1 = str_split($str);
  $arr2 = str_split($str, 3);
  print_r($arr1);
  print_r($arr2);
?>
```

Результатом работы скрипта из листинга 9.98 будут следующие дампы массивов:

```
Array
(
    [0] => П
    [1] => о
    [2] => т
    [3] => р
    [4] => о
    [5] => ш
    [6] => и
    [7] => м
    [8] =>
    [9] => с
    [10] => т
    [11] => р
    [12] => о
    [13] => к
    [14] => у
)
Array
(
    [0] => Пот
    [1] => рош
    [2] => им
    [3] => стр
    [4] => оку
)
```

Функция `strtok()` позволяет разбить строку на подстроки и имеет следующий синтаксис:

```
strtok($str, $token)
```

Функция `strtok()` возвращает строку по частям, а именно: она возвращает часть строки `$str` до разделителя `$token`. При последующих вызовах функции возвращается следующая часть до следующего разделителя, и так до конца строки. При первом вызове функция принимает два аргумента: исходную строку `$str` и разделитель `$token`. Обратите внимание, что при каждом последующем вызове `$str` указывать не следует, иначе будет возвращаться первая часть строки (листинг 9.99).

Листинг 9.99. Использование функции `strtok()`

```
<?php
  $str = "Предложение может содержать\nпробелы,
          запятые и символы\nпереноса";
  $tok = strtok($str," ,\n");
  while($tok)
  {
    echo "word=$tok<br>";
    $tok = strtok(" ,\n");
  }
?>
```

При помощи функции `strtok()` можно извлекать GET-параметры и их значения из строки запроса (листинг 9.100).

Листинг 9.100. Извлечение GET-параметров

```
<?php
  $str = "http://www.softtime.ru/forum/
          read.php?id_forum=1&id_theme=961&id_post=6806";
  $tok = strtok($str,"?&");
  while($tok = strtok("?&"))
  {
    echo "$tok<br>";
  }
?>
```

Результат:

```
id_forum=1
id_theme=961
id_post=6806
```

Функция `str_word_count()` позволяет разбивать строку на отдельные слова и возвращать количество слов в строке. Функция имеет следующий синтаксис:

```
str_word_count($str [, $format [, $charlist]])
```

Функция принимает строку `$str` и необязательный параметр `$format`, определяющий, какую информацию следует возвратить о строке. В случае его отсутствия возвращается число слов в строке. Далее описаны допустимые значения аргумента `$format` и соответствующие им возвращаемые значения:

- 1 — возвращается массив, содержащий все слова, входящие в строку `$str`;
- 2 — возвращается массив, индексами которого являются позиции в строке, а значениями — соответствующие слова.

Необязательный параметр `$charlist` позволяет задать дополнительные символы, из которых будет состоять слово.

Пример использования функции `str_word_count()` приводится в листинге 9.101.

Листинг 9.101. Использование функции `str_word_count()`

```
<?php
  $str = "Hello world!    Hello PHP!";

  $a   = str_word_count($str, 1);
  $b   = str_word_count($str, 2);
  $c   = str_word_count($str);

  echo "<pre>";
  print_r($a);
  print_r($b);
  echo $c;
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 9.101 будут следующие строки

```
Array
(
    [0] => Hello
    [1] => world
    [2] => Hello
    [3] => PHP
)
Array
(
    [0] => Hello
    [6] => world
    [16] => Hello
    [22] => PHP
)
4
```

Функция `explode()`, предназначена для разбивки строки по определенному разделителю и имеет следующий синтаксис:

```
explode($delimiter, $str [, $limit])
```

Функция возвращает массив из строк, каждая из которых соответствует фрагменту исходной строки `$str`, находящемуся между разделителями, определяемым аргументом `$delimiter`.

Необязательный параметр `$limit` определяет максимальное количество элементов в массиве. Оставшаяся (неразделенная) часть будет содержаться в последнем элементе. Пример использования функции `explode()` приводится в листинге 9.102.

Листинг 9.102. Использование функции `explode()`

```
<?php
  $str = "Имя, Фамилия, e-mail";
  $exp_str = explode(",", $str);
  echo "<pre>";
  print_r($exp_str);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 9.102 являются следующие строки:

```
Array
(
  [0] => Имя
  [1] => Фамилия
  [2] => e-mail
)
```

Функция `implode()` является обратной `explode()` функцией и осуществляет объединение элементов массива в строку. Функция имеет следующий синтаксис:

```
implode($delimiter, $arr)
```

Функция `implode()` возвращает строку, которая содержит элементы массива, заданного в параметре `$arr`, между которыми вставляется значение, указанное в параметре `$delimiter`. Пример использования функции приводится в листинге 9.103.

Листинг 9.103. Функция `implode()`

```
<?php
  $arr = array("Сидоров", "Иван", "Петрович");
  echo implode(", ", $arr); // Сидоров, Иван, Петрович
?>
```

Функция `sscanf()` интерпретирует строку `$str` согласно формату `$format` в соответствии со спецификацией для функции `printf()` и имеет следующий синтаксис:

```
sscanf($str, $format [, ...])
```

Обратите внимание, что при указании только двух аргументов *полученные значения возвращаются как массив*.

Пусть имеется строка, в которой находится информация о названии и серийном номере жесткого диска в виде "maxtor/203-5505" и необходимо извлечь из этой строки серийный номер. Для этого можно воспользоваться скриптом из листинга 9.104.

Листинг 9.104. Разбор строки функцией `sscanf()`

```
<?php
  $product = "maxtor/203-5505";
  $str = sscanf($product,"maxtor/%3d-%4d");
  echo("$str[0]-$str[1]");
?>
```

Результат работы скрипта:

```
203-5505
```

Если необязательные аргументы в функции не указываются, то она возвращает массив, и поэтому следует выводить значения элементов массива, как это демонстрируется в листинге 9.104.

При указании необязательных параметров функции `sscanf()` их нужно передавать по ссылке. Пример с использованием необязательных параметров приведен в листинге 9.105.

Листинг 9.105. Использование необязательных параметров в функции `sscanf()`

```
<?php
  $book = "1\tThinking in PHP";
  $str = sscanf($book,"%d\t%s %s %s", &$id, &$first, &$second, &$last);
  echo("book number $id - $first $second $last");
?>
```

Результат работы скрипта:

```
book number 1 - Thinking in PHP
```

Часто в HTML требуется ограничить количество символов на одной строке, т. к. слишком длинное слово или предложение может нарушить дизайн страницы. Для этого предназначена функция `wordwrap()`, которая осуществляет перенос на заданное количество символов с использованием символа разрыва строки. Функция имеет следующий синтаксис:

```
wordwrap($str [, $width [, $break [, $cut]]])
```

Функция разбивает блок текста `$str` на несколько строк, которые завершаются символами `$break` (по умолчанию это перенос строки — `\n`), так, чтобы в одной строке было не более `$width` букв (по умолчанию 75). Поскольку разбиение происходит по границам слов, текст остается вполне читаемым (листинг 9.106).

Листинг 9.106. Разбиение текста функцией `wordwrap()`

```
<?php
  $str = "Здесь может быть любой текст";
  $mod_str = wordwrap($str, 10, "<br>");
  echo($mod_str);
?>
```

Результат работы скрипта:

```
Здесь
может быть
любой
текст
```

Если аргумент `$cut` установлен в `TRUE`, разрыв делается точно в заданной позиции, даже если это приводит к разрыву слова. Поэтому если исходная строка содержит слово длиннее, чем заданная длина, то в этом случае слово будет разорвано (листинг 9.107).

Листинг 9.107. Использование параметра `cut`

```
<?php
  $text = "Очень длинное слоооооооооооооооово.";
  $newtext = wordwrap($text, 8, "<br>", 1);
  echo "$newtext";
?>
```

Результат работы скрипта:

```
Очень
длинное
слоооооо
оооооооо
оово.
```



Глава 10

Регулярные выражения

Регулярные выражения предназначены для описания фрагмента текста, что может использоваться для поиска, замены, извлечения и достаточно сложных процедур преобразования текста. Несмотря на то, что все эти действия больше относятся к строковым функциях, описание регулярных выражений выделено в отдельную главу, т. к. они представляют собой специализированный язык программирования.

Существует несколько диалектов регулярных выражений, среди которых наибольшее распространение получили Perl- и POSIX-регулярные выражения. В данной главе будет рассмотрен синтаксис Perl-регулярных выражений, т. к., начиная с PHP 6.0, регулярные выражения POSIX исключены из ядра и даже стандартных расширений.

10.1. Как изучать регулярные выражения?

Объемные и сложные задачи можно решать долго при помощи простых методов или быстро при помощи сложных. Классические строковые функции быстро осваиваются, но требуют длительного времени при решении более или менее сложных повседневных задач. Регулярные выражения освоить гораздо сложнее, и времени на это уходит больше. Однако повседневные задачи, даже достаточно сложные регулярные выражения, позволяют решать в короткий срок и с минимальными усилиями.

Куда потратить время — каждый из разработчиков решает сам. Однако свободное владение регулярными выражениями дает в программировании ту же свободу действий, что и десятипальцевый слепой метод в наборе текста — вы будете просто излагать свои мысли в коротких и элегантных строках регулярных выражений, а не тратить десятки часов на составление и отладку неуклюжих, состоящих из сотен строк блоков кода. Полученные знания можно применить в десятке программных систем, начиная самыми разными языками программирования (Perl, PHP, C#, Java, JavaScript и т. п.) и заканчивая базами данных (MySQL, Oracle).

Регулярные выражения являются достаточно сложным языком программирования, требующим от разработчиков высокой сосредоточенности, особенно на начальных этапах обучения. Если составление регулярного выражения вызывает у вас затруднение или рабочее регулярное выражение выглядит в ваших глазах каббалистической записью, вы всегда можете обратиться на наш форум, который полностью посвящен регулярным выражениям **http://www.softtime.ru/forum/read.php?id_forum=6**. Авторы и посетители форума с удовольствием помогут разобраться с решением задачи

и объяснят назначения элементов регулярного выражения. Сам форум содержит большое количество готовых решений, накопившихся там за несколько лет.

Изучить регулярные выражения, просто читая их описание, довольно сложно — требуется интенсивная практика. Для начала их следует использовать в повседневных задачах, если таких задач не много, новых задач всегда можно с избытком обнаружить на форуме **http://www.softtime.ru/forum/read.php?id_forum=6**. Множество посетителей форума отточило свое искусство составления регулярных выражений, отвечая на вопросы других посетителей.

Несмотря на то, что данная глава достаточно подробно обсуждает регулярные выражения, рассмотреть все аспекты их использования в ней невозможно. Для более глубокого изучения материала следует обратиться к книге Дж. Фридла "Регулярные выражения". К большому сожалению, выпуск русского издания прекращен, однако по ссылке **http://www.softtime.ru/info/fridl.php** можно загрузить электронную версию книги.

10.2. Синтаксис регулярных выражений

Шаблон, формируемый при помощи регулярных выражений, может состоять как из обычных символов, так и *метасимволов*. В табл. 10.1 приводятся наиболее распространенные метасимволы.

Таблица 10.1. *Метасимволы регулярных выражений*

Специальный символ	Описание
`.`	Соответствует любому одному символу
`[...]`	Соответствует одному символу из тех, что перечислены в квадратных скобках, например, выражение `[0123456789]` соответствует одной цифре, выражение может быть свернуто в последовательность `[0-9]`
`[^...]`	Соответствует одному любому символу, не перечисленному в квадратных скобках, например, выражение `[^0-9]` соответствует любому символу, не являющемуся цифрой
`^`	Позиция в начале строки, например, выражение `^flag` соответствует любой строке, начинающейся со слова `flag`
`$`	Позиция в конце строки, например, выражение `^fl$` соответствует строке, содержащей лишь два символа `fl`, а строке `flag` регулярное выражение уже не будет соответствовать
`|`	Любое из разделяемых выражений, например, выражение `first|second` соответствует любой строке, содержащей либо слово `first`, либо слово `second`
`(...)`	Круглые скобки служат для логического объединения частей регулярного выражения

Если искомая строка сама содержит метасимволы, их следует экранировать. Например, для поиска подстроки "ссылка [9]", где вместо цифры 9 может стоять любая другая цифра, может использоваться регулярное выражение:

```
"ссылка \[[0-9]\]"
```

Таким образом, для поиска квадратных скобок их необходимо экранировать, чтобы интерпретатор регулярных выражений не воспринимал их как начало символьного класса.

Символ точки, а также квадратные скобки всегда обозначают лишь один символ. Однако в приведенном выше примере ссылка может состоять из одной и более цифр. Для решения этой задачи используются *квантификаторы* (табл. 10.2), которые позволяют задать количество повторений символа. Таким образом, приведенное выше регулярное выражение можно обобщить на произвольное количество цифр в квадратных скобках:

```
"ссылка \[[0-9]+\]"
```

Таблица 10.2. *Квантификаторы регулярных выражений*

Квантификатор	Описание
?	Предшествующий символ либо входит в строку один раз, либо вообще в нее не входит
*	Предшествующий символ входит в строку любое число раз, в том числе и 0
+	Предшествующий символ входит в строку один или более число раз
{n}	Предшествующий символ входит в строку n раз
{n,}	Предшествующий символ входит в строку n или более количество раз
{n, m}	Предшествующий символ входит в строку от n до m раз

Знак вопроса, стоящий после последовательностей .* и .+, имеет специальное значение. Обычно регулярные выражения стремятся найти как можно более длинное соответствие, в связи с чем указанные выше последовательности стремятся вернуть всю строку. Однако такое поведение можно изменить, воспользовавшись последовательностями .*? или .+?, которые стремятся найти как можно более короткое соответствие.

Помимо метасимволов и квантификаторов, специальным значением обладает ряд экранированных обратным слешем символов (табл. 10.3).

Таблица 10.3. *Специальные символы регулярных выражений*

Специальный символ	Описание
\b	Позиция, соответствующая границе слова
\B	Позиция, не соответствующая границе слова
\n	Соответствует символу новой строки

Таблица 10.3 *(окончание)*

Специальный символ	Описание
`\r`	Соответствует символу возврата каретки
`\t`	Соответствует символу табуляции
`\f`	Соответствует символу конца файла
`\d`	Соответствует любой десятичной цифре, является аналогом символьного класса `[0-9]`
`\D`	Соответствует любому символу, кроме десятичной цифры, является аналогом символьного класса `[^0-9]`
`\w`	Соответствует любому алфавитно-цифровому символу и символу подчеркивания, т. е. символ, образующий "слово". Является аналогом символьного класса `[a-zA-Z0-9_]`
`\W`	Соответствует всем символам, которые не попадают под определение метасимвола `\w`
`\s`	Соответствует любому пробельному символу
`\S`	Соответствует любому не пробельному символу

Помимо обычных круглых скобок, диалект Perl-регулярных выражений предоставляет разработчику четыре типа позиционных проверок (табл. 10.4), позволяющих выяснить, находится та или иная подстрока справа или слева от текущей позиции.

Таблица 10.4. *Позиционные проверки регулярных выражений*

Позиционная проверка	Описание
`(?<=...)`	Выражение в скобках ... располагается слева
`(?<!...)`	Выражение в скобках ... не может располагаться слева
`(?=...)`	Выражение в скобках ... располагается справа
`(?!...)`	Выражение в скобках ... не может располагаться справа

В Perl-регулярных выражениях само выражение заключается в граничные символы, после которого могут следовать модификаторы регулярного выражения. Рассмотрим три выражения для поиска подстроки `"flag"`:

```
/flag/i
#flag#is
|flag|
```

В первом случае в качестве граничных символов выступает прямой слеш (`/`), во втором — символ диеза (`#`), в последнем случае — прямая черта (`|`). Символы, расположенные после закрывающего граничного символа, являются *модификаторами*. Список модификаторов приводится в табл. 10.5.

Замечание

Если прямая черта (|) выступает в качестве границы регулярного выражения, она не может использоваться внутри выражения как метасимвол, т. к. граничные символы придется экранировать, что снимает их специальное значение.

Таблица 10.5. *Модификаторы Perl-регулярных выражений*

Модификатор	Описание
i	Регулярное выражение не зависит от регистра
m	Если используется данный модификатор, то соответствие ищется в интервале между двумя переводами строк, а не во всем тексте
s	Если используется данный модификатор, то соответствие ищется во всем тексте, а не в интервале между двумя переводами строк
x	При использовании данного модификатора не экранируемые пробелы и символы табуляции игнорируются, если они находятся вне квадратных скобок
e	При использовании данного модификатора в функции `preg_replace()` после стандартных подстановок в заменяемой строке последняя интерпретируется как PHP-код, результат которого используется для замены искомой строки
U	При использовании данного модификатора ищется минимальное по длине соответствие регулярному выражению (без использования данного модификатора ищется максимальное соответствие)

10.3. Функции для работы с регулярными выражениями

Для работы с регулярными выражениями в PHP предусмотрен набор функций, начинающихся с префикса `preg`, краткое описание которых представлено в табл. 10.6.

Таблица 10.6. *Функции для работы с регулярными выражениями*

Функция	Описание
`preg_match($`*`pattern`*`, $`*`str`* `[, &$`*`matches`* `[, $`*`flags`* `[, $`*`offset`*`]]])`	Выполняет проверку на соответствие регулярному выражению `$`*`pattern`* строки `$`*`str`*. В необязательный параметр `$`*`matches`* могут быть помещены результаты поиска, параметр `$`*`flags`* определяет структуру массива `$`*`matches`*. Параметр `$`*`offset`* позволяет указать альтернативные позиции для начала поиска (по умолчанию поиск осуществляется с первой позиции строки `$`*`str`*). Функция возвращает 1, если соответствие найдено, или 0 в противном случае

Таблица 10.6 *(окончание)*

Функция	Описание
`preg_match_all($pattern, $str, &$matches [, $flags [, $offset]])`	Ищет все соответствия регулярному выражению `$pattern` в строке `$str` с размещением результатов поиска в массиве `$matches` (его структуру определяет параметр `$flags`). Параметр `$offset` позволяет указать альтернативные позиции для начала поиска (по умолчанию поиск осуществляется с первой позиции строки `$str`). В случае успешного поиска функция возвращает количество найденных соответствий, в противном случае возвращается `FALSE`
`preg_replace($pattern, $replacement, $str [, $limit [, &$count]])`	Осуществляет замену в строке `$str` по регулярному выражению `$pattern` на подстроку `$replacement`. Количество замен может быть ограничено необязательным параметром `$limit`, в параметре `$count` возвращается реальное количество осуществленных замен
`preg_replace_callback($pattern, $callback, $str [, $limit [, &$count]])`	Выполняет поиск по регулярному выражению `$pattern` и замену в строке `$str` с использованием функции обратного вызова `$callback`. Количество замен может быть ограничено необязательным параметром `$limit`, в параметре `$count` возвращается реальное количество осуществленных замен
`preg_quote($str [, $delimiter])`	Экранирует метасимволы регулярных выражений в строке `$str`. Необязательный параметр `$delimiter` позволяет указать дополнительные символы, которые также должны подвергаться экранированию
`preg_split($pattern, $str [, $limit [, $flags]])`	Разбивает строку `$str` по регулярному выражению `$pattern`. Необязательный параметр `$limit` позволяет ограничить количество возвращаемых фрагментов. Параметр `$flags` позволяет настроить поведение функции
`preg_grep($pattern, $input [, $flags])`	Принимает в качестве параметра `$input` массив и возвращает новый массив с элементами, соответствующими или несоответствующими регулярному выражению `$pattern`

10.4. Функция *preg_match()*

Функция `preg_match()` осуществляет поиск в строке по регулярному выражению и имеет следующий синтаксис:

```
preg_match($pattern, $str [, $matches [, $flags [, $offset]]])
```

В строке `$subject` ищется соответствие регулярному выражению `$pattern`. Если задан необязательный параметр `$matches`, то результаты поиска помещаются в массив. Элемент `$matches[0]` будет содержать часть строки, соответствующую вхождению всего шаблона; `$matches[1]` — часть строки, соответствующую первым круглым скобкам; `$matches[2]` — вторым и т. д. Необязательный флаг `$flag` может принимать

единственное значение `PREG_OFFSET_CAPTURE`, при указании которого изменяется формат возвращаемого массива `$matches`: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом — смещение. Поиск осуществляется слева направо, с начала строки. Дополнительный параметр `$offset` может быть использован для указания альтернативной начальной позиции для поиска. Функция `preg_match()` возвращает количество найденных соответствий. Это может быть 0 (совпадения не найдены) и 1, поскольку `preg_match()` прекращает свою работу после первого найденного совпадения.

В качестве примера использования функции `preg_match()` рассмотрим небольшое Web-приложение, осуществляющее проверку корректности введенного электронного адреса. Будем исходить из того, что адрес должен иметь вид ***something@server.com***. У адреса есть две составляющие — имя пользователя и имя домена, которые разделены символом `@`. В имени пользователя могут присутствовать буквы нижнего и верхнего регистров, цифры, знаки подчеркивания, тире и точки. В качестве разделителя между именем пользователя и именем домена в выражение требуется добавить знак `@`, после чего следует доменное имя, которое тоже может состоять из букв нижнего и верхнего регистра, цифр, тире и точки (знак подчеркивания не допускается в доменном имени). Итак, регулярное выражение, проверяющее имя пользователя, наличие разделителя и доменного имени, выглядит следующим образом:

```
"[-0-9a-z_.]+@[-0-9a-z^.]+"
```

Для проверки доменного имени первого уровня (**.ru**, **.com**) необходимо добавить следующее выражение:

```
"\.[a-z]{2,6}"
```

> ***Замечание***
>
> Домены первого уровня могут содержать до 6 символов (например, travel). Причем символ подчеркивания в доменном имени встречаться не может.

Символ `"."` в регулярных выражениях используется для обозначения любого символа, поэтому для поиска соответствия точки в регулярном выражении этот символ экранируется обратным слешем: `"\."`. (В символьном классе, заключенном в квадратные скобки, экранировать точку не обязательно.)

Объединяя эти строки, можно получить следующее регулярное выражение в формате Perl для проверки корректности адресов электронной почты:

```
"|[-0-9a-z_.]+@[-0-9a-z^.]+\.[a-z]{2,6}|i"
```

> ***Замечание***
>
> Модификатор `i` в регулярном выражении сообщает интерпретатору, чтобы поиск соответствия проводился без учета регистра.

Это регулярное выражение будет соответствовать e-mail, однако если строка должна содержать адрес электронной почты и ничего более, то регулярное выражение необходимо снабдить привязкой к началу (`^`) и концу текста (`$`).

```
"|^[-0-9a-z_.]+@[-0-9a-z^.]+\.[a-z]{2,6}$|i"
```

Окончательно проверка корректности адреса электронной почты может выглядеть так, как это представлено в листинге 10.1.

Листинг 10.1. Проверка корректности адреса электронной почты

```
<form method=post>
  <input size=60 type=text name=name value=<?= $_POST['name']; ?>>
  <input type=submit value='Проверить'>
</form><br>
<?php
// Обработчик HTML-формы
if (isset($_POST['name']))
{
  if (preg_match("|^[-0-9a-z_]+@[-0-9a-z^\.]+\.[a-z]{2,6}$|i",
$_POST['name']))
  {
    echo "e-mail верен";
  }
  else
  {
    echo "e-mail неверен";
  }
}
?>
```

Результат работы скрипта из листинга 10.1 представлен на рис. 10.1.

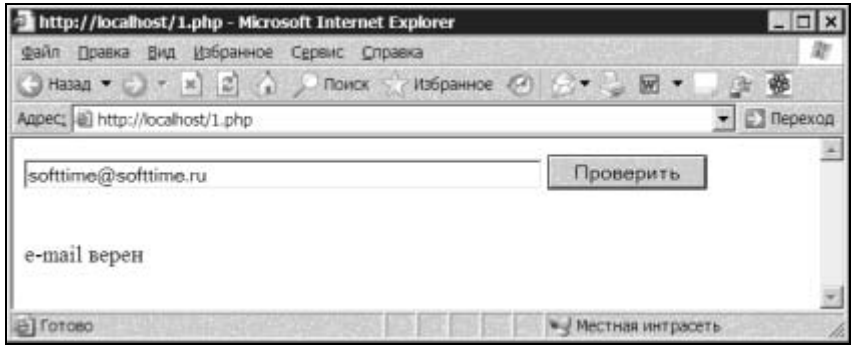


Рис. 10.1. HTML-форма проверки адреса электронной почты

В качестве дополнительного примера создадим HTML-форму, состоящую из двух текстовых полей, в первом из которых вводится количество товарных позиций, а во втором — их цена в формате `###.##`. Обработчик формы будет проверять, является ли введенная в первом поле информация целым числом, а во втором — удовлетворяющим

денежному формату. Если оба числа будут введены корректно, будет выводиться их произведение.

Проверить целое число очень просто, для его проверки введен специальный класс `\d`. Кроме этого, необходимо явно указать начало (`^`) и конец строки (`$`), иначе после того как будет указана одна цифра, можно будет ввести все, что угодно, — регулярное выражение найдет цифру и проигнорирует все остальное. Привязка к началу и концу строки позволяет избежать этого:

```
"|^[\d]*$|"
```

Обработка числа с плавающей точкой состоит из нескольких частей. Целой части числа соответствует рассмотренное в предыдущем разделе выражение `"[\d]*"`. После точки (которую необходимо экранировать) следует дробная часть, которая тоже описывается выражением `"[\d]*"`. Теперь, объединяя фрагменты, можно получить шаблон для чисел с плавающей точкой — `"|^[\d]*\.[\d]*$|"`. Завершающим штрихом будет учет разного формата разделителя целой и дробной части, в качестве которого может выступать как точка (23.56), так и запятая (23,56) — для учета чисел в обоих форматах регулярное выражение нужно изменить следующим образом: `"|^[\d]*[.,][\d]*$|"`. Однако такое выражение не позволит вводить целые числа, которые являются частным случаем чисел с плавающей точкой. Для того чтобы исправить это, необходимо добавить символ `?` после класса `[.,]`.

```
"|^[\d]*[.,]?[\d]*$|"
```

В листинге 10.2 представлено окончательное решение проверки корректности ввода числа.

Листинг 10.2. Проверка чисел на корректность

```
<form method=post>
  Число <input size=60 type=text name=number
               value='<?= htmlspecialchars($_POST['number']); ?>'><br>
  Цена <input size=60 type=text name=price
              value='<?= htmlspecialchars($_POST['price']); ?>'><br>
  <input type=submit value='Проверить'>
</form><br>
<?php
  // Обработчик HTML-формы
  if (isset($_POST['number']) && isset($_POST['price']))
  {
    if (!preg_match("|^[\d]*$|", $_POST['number']))
    {
      exit("Неверен формат числа товарных позиций");
    }
    if (!preg_match("|^[\d]*[.,]?[\d]*$|", $_POST['price']))
    {
```

```
      exit("Неверен формат цены");
    }
    echo number_format($_POST['number']*$_POST['price'], 2, '.', ' ');
  }
?>
```

При помощи функции `preg_match()` можно извлекать фрагменты строки. Например, при помощи регулярного выражения `"|<title>(.*)</title>|siU"` можно извлечь заголовок HTML-страницы. В названии могут содержаться угловые скобки < и >. Поэтому необходимо использовать теги `<title>` и `</title>`, управляя "жадностью" при помощи модификатора `U`. Регулярные выражения по умолчанию являются "жадными", т. е. возвращают самое длинное соответствие из всех возможных. Модификатор `U` позволяет инвертировать "жадность" и требует, чтобы найденное соответствие было самым коротким из всех найденных. Модификатор `U` позволяет менять "жадность" всего выражения, если же часть выражения должна состоять из "жадных" фрагментов, а часть из "не жадных", лучше воспользоваться последовательностью `.*?`, которая возвращает минимально возможное количество символов, в отличие от последовательности `.*`, пытающейся вернуть максимально возможное количество символов. Таким образом, следующие два регулярных выражения эквивалентны по результату

```
"|<title>(.*)</title>|siU"
"|<title>(.*?)</title>|si"
```

В листинге 10.3 представлено окончательное решение для извлечения названия HTML-страницы.

> ***Замечание***
>
> Синтаксис функции `file_get_contents()` более подробно рассматривается в *главе 13*.

Листинг 10.3. Извлечение названия HTML-страницы

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Регулярное выражение
  $pattern = "|<title>(.*?)</title>|si";

  // Извлекаем название HTML-страницы
  if (preg_match($pattern, $content, $out))
  {
    echo $out[1];
  }
?>
```

Так как требуемый результат находится в круглых скобках, следует обращаться ко второму элементу результирующего массива `$out` (`$out[1]`), первый элемент (`$out[2]`) содержит соответствие всему регулярному выражению.

10.5. Функция *preg_match_all()*

Если необходимо найти либо сосчитать все совпадения, следует воспользоваться функцией `preg_match_all()`, которая имеет такой синтаксис:

```
preg_match_all($pattern, $str, $matches [,$flags [,$offset]])
```

Функция ищет в строке `$str` все совпадения с регулярным выражением `$pattern` и помещает результат в массив `$matches` в порядке, определяемом комбинацией флагов `$flags`. Так же как и в случае функции `preg_match()`, можно задать смещение `$offset`, начиная с которого будет осуществляться поиск в строке `$str`.

После нахождения первого соответствия дальнейший поиск будет осуществляться не с начала строки, а от конца последнего найденного вхождения.

Дополнительный параметр `$flags` может комбинировать следующие значения:

- `PREG_PATTERN_ORDER` — если этот флаг установлен, результат будет упорядочен следующим образом: элемент `$matches[0]` содержит массив полных вхождений регулярного выражения, элемент `$matches[1]` — массив вхождений первых круглых скобок, `$matches[2]` — вторых и т. д. То есть если строка содержит три соответствия регулярному выражению, то подстроку для последнего соответствия всему регулярному выражению можно найти в элементе `$matches[0][3]`, а для первых круглых скобок данного соответствия — в элементе `$matches[1][3]`;
- `PREG_SET_ORDER` — если этот флаг установлен, результат будет упорядочен следующим образом: элемент `$matches[0]` содержит первый набор вхождений, элемент `$matches[1]` содержит второй набор вхождений и т. д. В таком случае массив `$matches[0]` содержит первый набор вхождений, а именно: элемент `$matches[0][0]` содержит первое вхождение всего регулярного выражения, элемент `$matches[0][1]` содержит первое вхождение первых круглых скобок, `$matches[0][1]` — вторых и т. д. Аналогично массив `$matches[1]` содержит второй набор вхождений, и так для каждого найденного набора;
- `PREG_OFFSET_CAPTURE` — если этот флаг установлен, для каждой найденной подстроки будет указана ее позиция в исходной строке. Необходимо помнить, что этот флаг меняет формат возвращаемых данных: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом — смещение.

> ***ЗАМЕЧАНИЕ***
>
> Использование значения `PREG_PATTERN_ORDER` одновременно с `PREG_SET_ORDER` бессмысленно.

Функция `preg_match_all()` возвращает количество найденных вхождений шаблона (может быть нулем) или `FALSE`, если во время выполнения возникли какие-либо ошибки.

Пусть имеется текстовый файл index.html и стоит задача подсчитать, сколько в нем содержится одно-, двух-, ..., десятибуквенных слов. Для поиска всех односимвольных слов может использоваться регулярное выражение `"|\b[\w]{1}\b|s"`. Последовательность `\b` является так называемой границей слова — без указания границ слова, регулярное выражение `[\w]{1}` будет соответствовать каждой букве любого слова. Для того чтобы обобщить решение на двух-, трех-, ... и десятисимвольные слова, необходимо в цикле изменять значение в фигурных скобках от 1 до 10 (листинг 10.4).

Листинг 10.4. Подсчет количества слов с различным количеством символов

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Удаляем HTML-теги
  $content = strip_tags($content);

  // Подсчитываем количество слов
  for($i = 1; $i <= 10; $i++)
  {
    $total = preg_match_all("|\b[\w]{".$i."}\b|s",
                  $content,
                  $out,
                  PREG_PATTERN_ORDER);
    echo "Количество слов с $i символами - $total<br>";
  }
?>
```

Для того чтобы продемонстрировать массовое извлечение подстрок при помощи регулярных выражений и функции `preg_match_all()`, создадим новостную ленту, которая выводит 10 последних новостных позиций с новостного портала **www.lenta.ru**. Портал предоставляет всем желающим возможность загрузки новостных позиций в различных форматах. Для создания собственной ленты новостей удобно воспользоваться RSS-каналом, доступном по адресу **http://img.lenta.ru/r/EX/import.rss**.

В XML-файле, который загружается с сайта **www.lenta.ru**, каждая новостная позиция описывается набором тегов, представленным в листинге 10.5.

Листинг 10.5. Фрагмент XML-файла с новостными позициями

```
<item>
<title>Два пенальти принесли &quot;Спартаку&quot; победу над
&quot;Байером&quot;</title>
<link>http://lenta.ru/news/2007/11/08/spartak/</link>
```

```
<description>Московский &quot;Спартак&quot; с победы начал свое выступление
в групповом турнире Кубка УЕФА. Команда Станислава Черчесова
в &quot;Лужниках&quot; принимала немецкий &quot;Байер&quot; из Леверкузена
и добилась победы со счетом 2:1. Оба спартаковских гола были забиты ударами
с пенальти во втором тайме: отличились Роман Павлюченко и Сантос Мо-
царт.</description>
<pubDate>Thu, 08 Nov 2007 20:54:31 +0300</pubDate>
<category>Спорт</category>
</item>
```

Тег `<title>` содержит заголовок новости, тег `<link>` указывает на страницу с подробным содержанием новостной позиции, тег `<description>` предоставляет краткое описание, тег `<pubDate>` традиционно содержит дату размещения новости, а тег `<category>` определяет категорию новостной позиции и может служить фильтром новостных позиций. В листинге 10.6 приводится скрипт, загружающий десять последних новостных позиций и выводящий их на странице.

Листинг 10.6. Вывод последних 10 новостных позиций

```
<?php
  // Ссылка на XML-файл
  $url = "http://img.lenta.ru/r/EX/import.rss";
  // Загружаем файл
  $content = file_get_contents($url);
  // Регулярное выражение
  $pattern = "|<item>[\s]*<title>(.*?)</title>[\s]*".
             "<link>(.*?)</link>[\s]*".
             "<description>(.*?)</description>[\s]*".
             "<pubDate>(.*?)</pubDate>[\s]*".
             "<category>(.*?)</category>|is";
  preg_match_all($pattern, $content, $out);
  // Выводим последние 10 позиций
  for($i = 0; $i < 10; $i++)
  {
    echo "<a href={$out[2][$i]}>{$out[1][$i]}</a><br>".
         "{$out[3][$i]}<br><br>";
  }
?>
```

Результат работы скрипта из листинга 10.5 представлен на рис. 10.2.

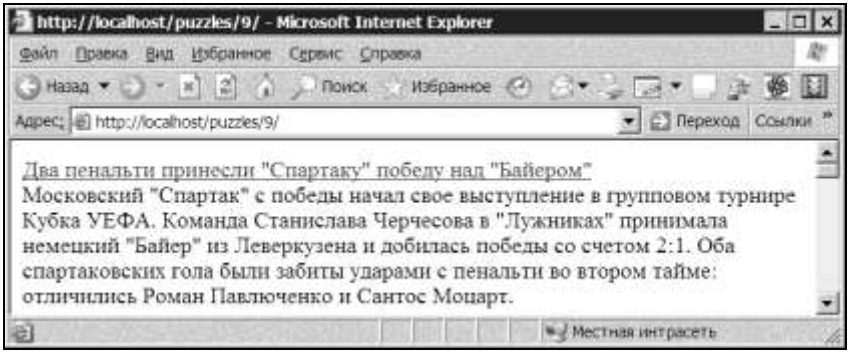


Рис. 10.2. Вывод новостных позиций

10.6. Функция *preg_replace()*

Функция `preg_replace()` осуществляет поиск и замену по регулярному выражению и имеет следующий синтаксис:

```
preg_replace($pattern, $replacement, $str [, $limit])
```

Эта функция ищет в строке `$str` соответствие регулярному выражению `$pattern` и заменяет его на `$replacement`. Необязательный параметр `$limit` задает количество соответствий, которые необходимо заменить. Если этот параметр не указан или равен –1, то заменяются все найденные подстроки.

Параметр `$replacement` может содержать ссылки вида `\\n`; каждая такая ссылка будет заменена на подстроку, соответствующую `n`-ым круглым скобкам. `n` может принимать значения от 0 до 99, причем ссылка `\\0` соответствует вхождению всего шаблона. Выражения в круглых скобках нумеруются слева направо, начиная с единицы.

Если во время выполнения функции были обнаружены совпадения с шаблоном, будет возвращено измененное значение параметра `$str`, в противном случае будет возвращен исходный текст `$str`.

Первые три параметра функции `preg_replace()` могут быть одномерными массивами. В случае если массив использует ключи, при обработке массива они будут взяты в том порядке, в котором расположены в массиве.

В случае если параметры `$pattern` и `$replacement` являются массивами, функция `preg_replace()` поочередно извлекает из обоих массивов по паре элементов и использует их для операции поиска и замены. Если массив `$replacement` содержит больше элементов, чем `$pattern`, вместо недостающих элементов для замены будут взяты пустые строки. Если `$pattern` является массивом, а `$replacement` — строкой, по каждому элементу массива `$pattern` будет осуществлен поиск и замена на `$replacement` (шаблоном будут поочередно все элементы массива, в то время как строка замены остается фиксированной). Вариант, когда `$pattern` является строкой, а `$replacement` — массивом, не имеет смысла.

В листинге 10.7 приводится пример использования функции `preg_replace()`, в котором из файла index.htm удаляются все HTML-теги.

Листинг 10.7. Удаление тегов из HTML-страницы

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Массив регулярных выражений
  $search = array ("'<script[^>]*?>.*?</script>'si",
                   "'<[\/\!]*?[^<>]*?>'si",
                   "'([\r\n])[\s]+'",
                   "'&(quot|#34);'i",
                   "'&(amp|#38);'i",
                   "'&(lt|#60);'i",
                   "'&(gt|#62);'i",
                   "'&(nbsp|#160);'i",
                   "'&(iexcl|#161);'i",
                   "'&(cent|#162);'i",
                   "'&(pound|#163);'i",
                   "'&(copy|#169);'i",
                   "'&#(\d+);'e");
 // Массив замены
 $replace = array ("",
                   "",
                   "\\1",
                   "\"",
                   "&",
                   "<",
                   ">",
                   " ",
                   chr(161),
                   chr(162),
                   chr(163),
                   chr(169),
                   "chr(\\1)");
  // Осуществляем удаление тегов и вывод текста в окно браузера
  echo preg_replace($search, $replace, $content);
?>
```

Для операции удаления HTML-тегов обычно используется специализированная функция `strip_tags()`, синтаксис которой подробно рассмотрен в *разд. 9.5*. Использование

данной функции не всегда удобно, т. к. HTML-теги заменяются пробелами, так, чтобы длина строки не изменялась. Впрочем, очистить текст от лишних пробельных символов можно также при помощи регулярных выражений. Для решения задачи удобно воспользоваться функцией `preg_replace()` и регулярным выражением `"|[\s]+|s"`. В листинге 10.8 приводится пример сжатия текстовой строки после обработки файла функцией `strip_tags()`.

Листинг 10.8. Замена нескольких пробельных символов одним

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Удаляем HTML-теги
  $content = strip_tags($content);

  // Заменяем несколько пробельных символов одним
  echo preg_replace("|[\s]+|s", " ", $content);
?>
```

Класс `\s` объединяет все пробельные символы и эквивалентен набору классов `[ \f\n\r\t\v]`. Скрипт из листинга 10.8 заменяет все пробельные символы пробелом — это может быть неудобно, если для каких-то целей необходимо сохранить переводы строк. В листинге 10.9 приводится альтернативный скрипт, который осуществляет сжатие текста, не затрагивая переводы строк.

Листинг 10.9. Альтернативное решение задачи

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Удаляем HTML-теги
  $content = strip_tags($content);

  // Заменяем несколько пробельных символов
  // одним, не затрагивая переводы строк
  echo preg_replace("|[ \f\t\v]+|s", " ", $content);
?>
```

Если для удаления всех HTML-тегов имеется специализированная функция `strip_tags()`, то в ряде других задач обработки HTML-тегов без регулярных выражений обойтись сложно. Одной из таких задач является удаление HTML-тега изображе-

ния <img>. Для решения этой задачи можно воспользоваться регулярным выражением "|<img[^>]+>|si": замена его пустой строкой приведет к удалению всех изображений из содержимого HTML-страницы (листинг 10.10).

Листинг 10.10. Удаление изображений из HTML-страницы

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Регулярное выражение
  $search = "|<img[^>]+>|si";
  // Замена
  $replace = "";

  // Осуществляем удаление тегов и вывод текста в окно браузера
  echo preg_replace($search, $replace, $content);
?>
```

Тег <img> является одиночным, поэтому после того, как найдено вхождение подстроки "<img", далее необходимо извлечь весь текст до первой закрывающей угловой скобки > — "[^>]+>".

Пусть в тексте требуется заменить все символы точки (.) на многоточие (...), но только в том случае, если точка не стоит в конце сокращений "г.", "рис." и "табл.".

Для того чтобы осуществить замену точки на многоточие в тексте с условием, необходимо проверить, не предшествуют ли точке подстроки "г", "рис" и "табл". Для решения данной задачи удобно воспользоваться негативной ретроспективной проверкой. Такая проверка оформляется в виде круглых скобок, внутри которых имеется последовательность ?<!, после которой указывается проверяемое выражение.

Например, для замены всех точек на многоточие, не затрагивая последовательность "г.", можно воспользоваться регулярным выражением "|(?<!г)\.|is" (листинг 10.11).

Листинг 10.11. Замена точки на многоточие, не затрагивая "г."

```
<?php
  $text = 'На рис. 6 представлен график по данным табл. 8.
           Как видно из рисунка, максимум приходится на 2002 г.,
           поэтому целесообразно сосредоточиться на периоде
           1998-2002 гг., как наиболее показательном.';
  $text = preg_replace("|(?<!г)\.|is", "...", $text);
  echo $text;
?>
```

Результатом работы скрипта из листинга 10.11 будет следующий фрагмент текста:

```
На рис... 6 представлен график по данным табл... 8... Как видно из рисунка,
максимум приходится на 2002 г., поэтому целесообразно сосредоточиться
на периоде 1998-2002 гг., как наиболее показательном...
```

Как видно из результата работы скрипта, все точки подвергаются замене, кроме тех, которым предшествует символ "г". Теперь остается обобщить решение на последовательности "рис" и "табл". К сожалению, в скобках ретроспективных проверок не допускается использование символа `|`, объединение последовательностей `"(?<!г)\."`, `"(?<!г)\."` и `"(?<!г)\."` при помощи `|` также не проходит, т. к. проверка негативная, а не позитивная. Выйти из ситуации позволяет тот факт, что ретроспективные проверки допускают вложение. Поэтому три негативных ретроспективных проверки можно объединить в одной позитивной (листинг 10.12).

Листинг 10.12. Вложенные ретроспективные проверки

```
<?php
  $text = 'На рис. 6 представлен график по данным табл. 8.
           Как видно из рисунка, максимум приходится на 2002 г.,
           поэтому целесообразно сосредоточиться на периоде
           1998-2002 гг., как наиболее показательном.';
  $text = preg_replace("|(?<=((?<!г)(?<!рис)(?<!табл)))\.|is",
                       "...",
                       $text);
  echo $text;
?>
```

Результатом работы скрипта из листинга 10.12 будет следующий фрагмент текста:

```
На рис. 6 представлен график по данным табл. 8... Как видно из рисунка,
максимум приходится на 2002 г., поэтому целесообразно сосредоточиться
на периоде 1998-2002 гг., как наиболее показательном...
```

10.7. Функция *preg_replace_callback()*

Функция `preg_replace_callback()` осуществляет поиск по регулярному выражению и замену с использованием функции обратного вызова и имеет следующий синтаксис:

```
preg_replace_callback($pattern, $callback, $str [, $limit])
```

Поведение этой функции во многом сходно с `preg_replace()`, за исключением того, что вместо параметра *`$replacement`* необходимо указывать функцию *`$callback`*, которой в качестве входящего параметра передается массив найденных вхождений. Функция обратного вызова *`$callback`* возвращает строку, в которой будет произведена замена.

Пусть имеется строка `"pole=1(20),pole2=1(3),pole3=1(1)"`, и необходимо при каждом обращении к скрипту уменьшать значение чисел в скобках на единицу, причем, если значение достигает 0, всю подстроку `poleX=1(Y)` необходимо удалить из подстро-

ки. Для решения этой задачи удобно воспользоваться функцией `preg_replace_callback()` с пользовательской функцией обратного вызова `replace_number()` (листинг 10.13).

Листинг 10.13. Использование функции `preg_replace_callback()`

```
<?php
  // Текст для разбора
  $text = "pole=1(20),pole2=1(3),pole3=1(1)";
  // Функция обратного вызова
  function replace_number($matches)
  {
    // $matches[0] -  полное вхождение шаблона
    // $matches[1] - вхождение первой подмаски,
    // заключенной в круглые скобки, и т. д.

    // Уменьшаем на единицу
    $matches[3]--;
    // Если значение меньше или равно нулю, исключаем поле
    if ($matches[3] <= 0) return "";
    // В противном случае возвращаем подстроку с новым значением
    else return "pole$matches[1]=$matches[2]($matches[3])";
  }
  $text = preg_replace_callback(
              "|pole([\d]*)=([\d]+)\(([\d]+)\)|i",
              "replace_number",
              $text);
  // Нужно теперь удалить сдвоенные запятые
  $text = preg_replace("#([,]+|,$)#", ",", $text);
  // А также завершающую запятую
  $text = preg_replace("|,$|i", "", $text);
  // Выводим результат
  echo $text;
?>
```

При построении различных Web-приложений, главным образом гостевых книг, форумов и чатов часто возникает необходимость защиты дизайна страниц от длинных последовательностей символов, которые могут растянуть дизайн. При помощи регулярных выражений можно легко решить эту проблему, применив функцию `wordwrap()` к словам, количество символов в которых превышает 25 символов (листинг 10.14).

> ***Замечание***
>
> Синтаксис функции `wordwrap()`, которая позволяет разбивать строку на части, более подробно обсуждается в *главе 9*.

Листинг 10.14. Разбивка длинной строки

```
<?php
  $text = "AAAAAAAAAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAAAAAAAAAAAAaaaaaaaaa";

  $text = preg_replace_callback("|(\w{25,})|",
                                "split_text",
                                $text);
  function split_text($matches)
  {
    return wordwrap($matches[1], 25, '<br>', 1);
  }

  echo $text;
?>
```

10.8. Функция *preg_split()*

Функция `preg_split()` разбивает строку по регулярному выражению и имеет следующий синтаксис:

`preg_split(`*`$pattern`*`, `*`$str`*` [, `*`$limit`*` [, `*`$flags`*`]])`

Функция возвращает массив, состоящий из подстрок заданной строки *`$str`*, которая разбита по границам, соответствующим шаблону *`$pattern`*.

В случае если параметр *`$limit`* указан, функция возвращает не более, чем *`$limit`* подстрок, при его отсутствии или равенстве –1 функция действует без ограничений. Последний параметр *`$flags`* может быть произвольной комбинацией следующих флагов (соединение происходит при помощи оператора `|`):

- `PREG_SPLIT_NO_EMPTY` — если этот флаг установлен, функция `preg_split()` вернет только непустые подстроки;
- `PREG_SPLIT_DELIM_CAPTURE` — если этот флаг установлен, выражение, заключенное в круглые скобки в разделяющем шаблоне, также извлекается из заданной строки и возвращается функцией;
- `PREG_SPLIT_OFFSET_CAPTURE` — если этот флаг установлен, для каждой найденной подстроки будет указана ее позиция в исходной строке. Этот флаг меняет формат возвращаемых данных: каждое вхождение возвращается в виде массива, в нулевом элементе которого содержится найденная подстрока, а в первом — смещение.

Продемонстрируем работу функции на примере разбивки объемного текста на предложения. В качестве регулярного выражения лучше использовать символ точки с последующими пробельными символами `"|\.[\s]+|s "` (листинг 10.15).

Листинг 10.15. Разбивка текста на предложения

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");

  // Удаляем HTML-теги
  $content = preg_replace("|[\s]+|s", " ", strip_tags($content));

  $text = preg_split("|[\.!\?][\s]+|s", $content);
  echo "<pre>";
  print_r($text);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 10.15 представлен на рис. 10.3. Как видно, это лишь промежуточный этап, теперь необходимо разбить каждое предложение на отдельные слова.

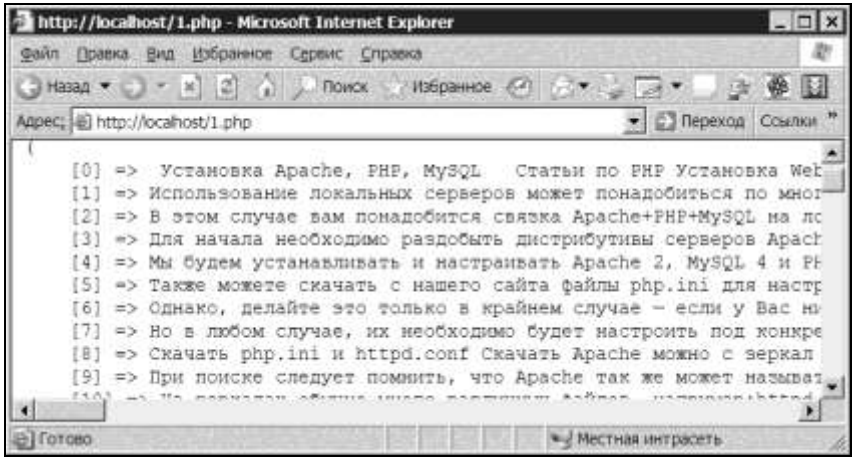


Рис. 10.3. Содержимое массива `$text` с предложениями

В листинге 10.16 представлен модифицированный скрипт, который дополнительно разбивает каждое предложение на отдельные слова и формирует двумерный массив.

Листинг 10.16. Преобразование массива `$text` в двумерный

```
<?php
  // Извлекаем содержимое из файла index.htm
  $content = file_get_contents("index.htm");
```

```
  // Удаляем HTML-теги
  $content = preg_replace("|[\s]+|s", " ", strip_tags($content));

  for($i = 0; $i < count($text); $i++)
  {
    $text[$i] = preg_split("|[-\s,]+|s", $text[$i]);
  }
  echo "<pre>";
  print_r($text);
  echo "</pre>";
?>
```

Результат работы скрипта в конечном варианте представлен на рис. 10.4.

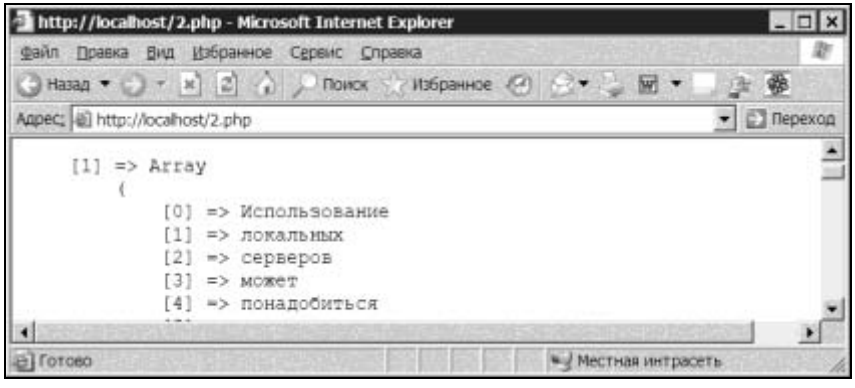


Рис. 10.4. Содержимое массива `$text` с отдельными словами

10.9. Функция *preg_quote()*

Функция `preg_quote()` экранирует в строке метасимволы регулярных выражений и имеет следующий синтаксис:

`preg_quote($str [, $delimiter])`

Функция принимает параметр `$str` и добавляет обратный слеш перед каждым метасимволом: `. \ + * ? [ ^ ] $ ( ) { } = ! < > | :`. Это бывает полезно, если шаблон формируется динамически, например, с участием строки, введенной пользователем.

Символ, указанный в необязательном параметре `$delimiter`, также подвергается экранированию (в этот параметр часто помещают символ границы регулярного выражения).

Данная функция позволяет динамически формировать регулярное выражение, не опасаясь, что символы искомой строки будут восприниматься как метасимволы (листинг 10.17).

Листинг 10.17. Использование функции `preg_quote()`

```
<?php
  $text = "Язык программирования *template*";
  $part = "*template*";
  $text = preg_replace("#".preg_quote($part)."#",
                       "<b>PHP</b>",
                        $text);
  echo $text; // Язык программирования PHP
?>
```



Глава 11

Дата и время

Функции даты и времени позволяют получать текущее время и дату, а также форматировать вывод даты. В PHP 5 и PHP 6 порядок работы с временем и датой несколько изменился, а количество функций этой группы значительно возросло. В частности некоторые функции предназначены для генерации и обслуживания объекта `DateTime`, более подробно рассматриваемого в *главе 26*.

11.1. Формирование даты и времени

Исторически сложилось, что время и дата имеют собственную достаточно запутанную систему счисления, которая значительно отличается от десятичной, используемой в математических и компьютерных вычислениях. Чтобы совместить эти две системы счисления, в компьютерных вычислениях время хранят в UNIXSTAMP-формате, при необходимости форматируя дату для отображения в традиционном формате. В UNIXSTAMP-формате время хранится в секундах, прошедших с 0:00 1 января 1970 года. Время в таком формате позволяет легко сравнивать даты друг с другом, а также прибавлять и вычитать интервалы времени. Функции, формирующие дату в этом формате, представлены в табл. 11.1.

***Таблица 11.1.** Формирование времени и даты*

Функция	Описание
`time()`	Возвращает текущую дату и время в UNIXSTAMP-формате
`microtime([$get_as_float])`	Возвращает текущую дату и время в UNIXSTAMP-формате с микросекундами. Если параметр `$get_as_float` принимает значение `TRUE`, функция возвращает результат в виде числа с плавающей точкой
`mktime([$hour [, $minute [, $second [, $month [, $day [, $year [, $is_dst]]]]]]])`	Возвращает дату и время в UNIXSTAMP-формате с годом `$year`, месяцем `$month`, днем месяца `$day`, часом `$hour`, минутами `$minute`, секундами `$second`. Если параметр `$id_dst` принимает значение 1, считается, что установлено летнее время, если 0 — зимнее, если –1 или параметр не указан вовсе, PHP пытается автоматически определить его

Таблица 11.1 *(окончание)*

Функция	Описание
gmmktime([$*hour* [, $*minute* [, $*second* [, $*month* [, $*day* [, $*year* [, $*is_dst*]]]]]]])	Функция gmmktime() аналогична mktime(), однако возвращается не местное время, а время по Гринвичу
gettimeofday([$*return_float*])	Возвращает текущую дату и время либо в виде ассоциативного массива, либо, если параметр $*return_float* принимает значение TRUE, в виде числа с плавающей точкой
strtotime($*time* [, $*timestamp*])	Разбирает строку с датой и временем и возвращает их в формате UNIXSTAMP

Для получения текущего времени и даты в формате UNIXSTAMP традиционно применяется функция time(). Пример использования функции приводится в листинге 11.1.

Листинг 11.1. Использование функции `time()`

```
<?php
  echo time(); // 1102283188
?>
```

Иногда, особенно для профилирования скриптов, удобнее воспользоваться функцией microtime(), которая возвращает метку времени с микросекундами и имеет следующий синтаксис:

```
microtime([$get_as_float])
```

При вызове без необязательного параметра возвращается строка в формате "msec sec", где sec — это количество секунд, прошедших с 0:00:00 1 января 1970 года, а msec — это микросекунды. Если передан необязательный параметр $get_as_float, равный TRUE, функция microtime() возвращает действительное число, целая часть которого соответствует секундам, а дробная — микросекундам.

В листинге 11.2 приводится пример функции microtime() как с использованием необязательного параметра get_as_float, так и без применения него.

Листинг 11.2. Использование функции `microtime()`

```
<?php
  echo microtime()."<br>"; // 0.82713800 1136125418
  echo microtime(TRUE);    // 1136125418.83
?>
```

Действительное число более удобно при измерениях скорости вычисления скрипта, т. к. не требуется дополнительных преобразований секунд и микросекунд в действительное число (листинг 11.3).

Замечание

Во время первого запуска, скрипт обычно выполняется несколько медленнее, чем последующие, поэтому для получения объективных значений разумно набрать статистику по нескольким последовательным запускам.

Листинг 11.3. Измерение скорости выполнения скрипта

```
<?php
  // Время начала выполнения скрипта
  $begin = microtime(TRUE);

  // Основное тело скрипта
  for($i = 0; $i < 5; $i++)
  {
     // Односекундная задержка
     sleep(1);
  }

  // Время завершения выполнения скрипта
  $end = microtime(TRUE);

  // Общее время выполнения скрипта
  echo "Скрипт выполнялся ".($end - $begin)." секунд";
?>
```

Для формирования произвольной временной метки следует воспользоваться функцией `mktime()`, которая имеет следующий синтаксис:

```
mktime([$hour [,
        $minute [,
        $second [,
        $month [,
        $day [,
        $year [,
        $is_dst]]]]]])
```

Все принимаемые ею аргументы являются необязательными; если не передан ни один из аргументов, функция вернет текущее время. Назначение семи аргументов функции следующее:

- `$hour` — часы;
- `$minute` — минуты;
- `$second` — секунды;
- `$month` — месяц;

- `$day` — день;
- `$year` — год;
- `$is_dst` — аргумент `$is_dst` может быть установлен в 1, если заданной дате соответствует летнее время, 0 в противном случае, или –1 (значение по умолчанию), если неизвестно, действует ли летнее время на заданную дату. В последнем случае PHP пытается определить это самостоятельно.

В листинге 11.4 приводится пример использования функции `mktime()`.

Листинг 11.4. Использование функции `mktime()`

```
<?php
  echo mktime(0, 0, 0, 2, 24, 2005); // 1109192400
?>
```

Параметр `$year` может быть двух- или четырехзначным числом. Значения от 0 до 69 соответствуют 2000—2069, а 70—99 соответствуют 1970—1999 (в большинстве современных систем, где время представляется 32-битным целым со знаком, допустимыми являются значения `$year` между 1901 и 2038).

> ***Замечание***
>
> Ни одна из версий Windows не поддерживает отрицательные метки времени. Поэтому для Windows допустимыми являются значения `year` между 1970 и 2038.

Используя комбинацию функций `time()` и `mktime()`, можно выяснить, например, число дней между какой-либо датой и сегодняшним днем (листинг 11.5).

Листинг 11.5. Вычисление числа дней между сегодняшним днем и 26.12.2002

```
<?php
  // Число дней между сегодняшним днем и 26.12.2002
  echo (time() - mktime(0, 0, 0, 12, 26, 2002))/86400;
?>
```

Еще одной функцией, позволяющей получить время в формате UNIXSTAMP, является функция `gettimeofday()`, которая имеет следующий синтаксис:

```
gettimeofday([$return_float])
```

По умолчанию, функция возвращает ассоциативный массив, ключи которого имеют следующее значение:

- `sec` — время в формате UNIXSTAMP;
- `usec` — микросекунды;
- `minuteswest` — смещение к западу от Гринвича;
- `dsttime` — принимает 1, если временной метке соответствует летнее время и 0 для зимнего.

В листинге 11.6 приводится пример использования функции `gettimeofday()`.

Листинг 11.6. Использование функции `gettimeofday()`

```
<?php
  $arr = gettimeofday();

  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результат выполнения скрипта из листинга 11.6 может выглядеть следующим образом:

```
Array
(
    [sec] => 1208956959
    [usec] => 125002
    [minuteswest] => -240
    [dsttime] => 1
)
```

Как видно из результата, элемент с ключом `minuteswest` принимает значение, соответствующее смещению, равному 4 часам от Гринвича (4 × 60 минут = 240). С учетом летнего времени (`dsttime == 1`) это соответствует третьему часовому поясу (московское время).

Если необязательный параметр `$return_float` функции `gettimeofday()` принимает значение `TRUE`, вместо ассоциативного массива функция возвращает дату и время в виде вещественного числа, целая часть которого соответствует секундам, а дробная — микросекундам. Поэтому скрипт из листинга 11.3 можно переписать с использованием функции `gettimeofday()` (листинг 11.7).

Листинг 11.7. Использование функции `gettimeofday()`

```
<?php
  // Время начала выполнения скрипта
  $begin = gettimeofday(TRUE);

  // Основное тело скрипта
  for($i = 0; $i < 5; $i++)
  {
     // Односекундная задержка
     sleep(1);
  }
```

```
  // Время завершения выполнения скрипта
  $end = gettimeofday(TRUE);

  // Общее время выполнения скрипта
  echo "Скрипт выполнялся ".($end - $begin)." секунд";
?>
```

Функция `strtotime()` позволяет получить время в формате UNIXSTAMP, отталкиваясь от текущей даты. Функция имеет следующий формат:

```
strtotime($time [, $timestamp])
```

Первый параметр `$time` может принимать различные значения, начиная с "now", обозначающее текущее время, заканчивая датой в формате "10 Сентября 2008 года" и интервалами вида "+1 week" ("+1 неделя"), которые позволяют прибавлять к текущей дате указанный интервал. Необязательный параметр `$timestamp` позволяет задать произвольную временную метку вместо текущей даты и времени. В листинге 11.8 приводится пример использования функции `strtotime()`.

> ***ЗАМЕЧАНИЕ***
>
> Функция `date()`, использованная в листинге 11.8, описывается в *разд. 11.3*.

Листинг 11.8. Использование функции `strtotime()`

```
<?php
  // Устанавливаем часовой пояс
  date_default_timezone_set("Europe/Moscow");

  $arr = array("now",               // Сейчас
               "10 September 2000", // 10 Сентября 2000
               "+1 day",            // Завтра
               "+1 week",           // Через неделю
              "+1 week 2 days 4 hours 2 seconds", // Через 9 дн. 4 ч. 2 с.
               "next Thursday", // Следущий вторник
               "last Monday");  // Предыдущий понедельник

  echo "<table border=1>";
  foreach($arr as $time)
  {
    echo "<tr>
            <td>$time</td>
            <td>".date("d.m.Y H:i:s", strtotime($time))."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 11.8 представлен на рис. 11.1.

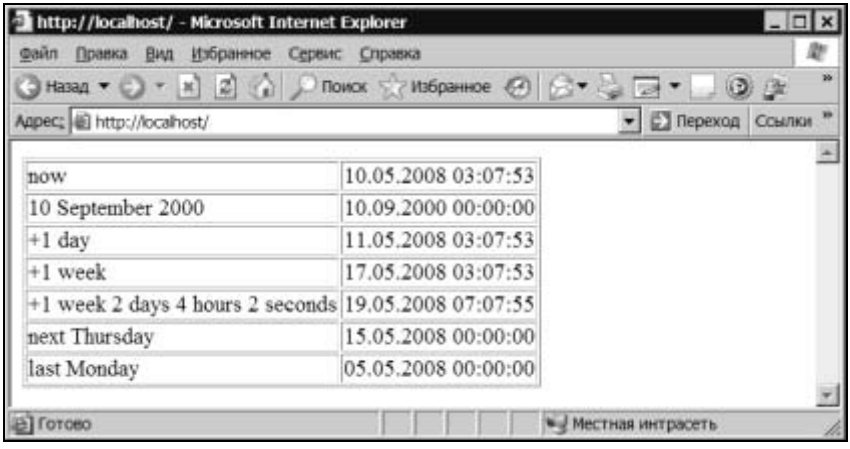


Рис. 11.1. Результат преобразования дат и временных интервалов функцией `strtotime()`

11.2. Географическая привязка

При работе с датами большое значение приобретают часовые пояса, которые различаются для разных географических местностей. Земной шар представляет собой шар (точнее, геоид, но в данном контексте это не очень важно), поэтому он поделен на 24 часовые зоны. За нулевую точку принимается нулевой меридиан, проходящий через небольшой английский городок Гринвич. Поэтому меридиан зачастую называют гринвичским, а время, соответствующее нулевой зоне, — гринвичским временем. Все остальные часовые пояса отсчитываются от этого времени, например, московское время смещено от гринвичского на 3 часа зимой и на 4 летом.

> ***ЗАМЕЧАНИЕ***
>
> Земной шар представляет собой шар (точнее геоид), поэтому если с одной его стороны ночь, то с другой — день.

В табл. 11.2 перечислены функции географической привязки.

Таблица 11.2. *Функции географической привязки*

Функция	Описание
`date_default_timezone_get()`	Возвращает текущую часовую зону
`date_default_timezone_set ($timezone_identifier)`	Устанавливает новую часовую зону `$timezone_identifier`
`timezone_identifiers_list()`	Возвращает массив с названиями часовых зон

***Таблица 11.2** (окончание)*

Функция	Описание
`timezone_abbreviations_list()`	Возвращает массив с названиями часовых зон, смешениями относительно Гринвича, а также указанием, поддерживается в данной области переход на летнее время или нет
`timezone_name_from_abbr($abbr [, $gmt_offset [, $isdst]])`	Возвращает название часовой зоны по аббревиатуре `$abbr` или смещению относительно Гринвича в секундах `$gmt_offset`. Если необязательный параметр `$isdst` принимает значение `TRUE`, зона должна поддерживать переход на летнее время, если `FALSE` — не поддерживать
`date_sunrise($timestamp [, $format [, $latitude [, $longitude [, $zenith [, $gmt_offset]]]]])`	Возвращает время восхода солнца для дня года, задаваемого при помощи параметра `$timestamp`, широты `$latitude` и долготы `$longitude` в формате, определяемом параметром `$format`. Необязательный параметр `$zenith` позволяет задать зенит (90° для точки горизонта), а параметр `$gmt_offset` — смещение в часах от Гринвича — функция обычно самостоятельно вычисляет такое смещение по широте, однако не учитывает летнее время (именно в этом случае и использует параметр `$gmt_offset`)
`date_sunset($timestamp [, $format [, $latitude [, $longitude [, $zenith [, $gmt_offset]]]]])`	Возвращает время заката солнца для дня года, задаваемого при помощи параметра `$timestamp`, широты `$latitude` и долготы `$longitude` в формате, определяемом параметром `$format`. Необязательный параметр `$zenith` позволяет задать зенит (90° для точки горизонта), а параметр `$gmt_offset` — смещение в часах от Гринвича — функция обычно самостоятельно вычисляет такое смещение по широте, однако не учитывает летнее время (именно в этом случае и использует параметр `$gmt_offset`)
`date_sun_info($timestamp, $latitude, $longitude)`	Для дня года, задаваемого при помощи параметра `$timestamp`, широты `$latitude` и долготы `$longitude`, возвращает информацию о времени восхода и захода солнца, а также начале и конце сумерек. Широта и долгота задаются в градусах

Функция `date_default_timezone_set()` и `date_default_timezone_get()` позволяют соответственно установить и возвратить текущую часовую зону. В листинге 11.8 приводится пример использования функций.

Листинг 11.8. Установка часовой зоны

```
<?php
  date_default_timezone_set("Europe/Moscow");
  echo date_default_timezone_get(); // Europe/Moscow
?>
```

Установленная при помощи функции `date_default_timezone_set()` часовая зона учитывается в дальнейшем при работе всех функций, работающих с датой и временем. Часовую зону можно также указать в конфигурационном файле php.ini. Зона по умолчанию задается директивой `date.timezone` из секции `[Date]`.

Часовых зон достаточно много, и привести полный их список не представляется возможным, впрочем, в этом нет необходимости, т. к. PHP предоставляет множество функций для получения названий зон. Первой функцией является функция `timezone_identifiers_list()`, которая возвращает массив со списком часовых зон (листинг 11.9).

Листинг 11.9. Список часовых поясов

```
<?php
  $arr = timezone_identifiers_list();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Далее приводится фрагмент дампа результирующего массива:

```
Array
(
    [0] => Africa/Abidjan
    [1] => Africa/Accra
    ...
    [366] => Europe/Monaco
    [367] => Europe/Moscow
    [368] => Europe/Nicosia
    ...
    [445] => Pacific/Yap
    [446] => UTC
)
```

Более подробного результата можно добиться при помощи функции `timezone_abbreviations_list()`, которая возвращает массив, зоны в которой разделены по географическим группам. Группа географических зон Восточной Европы находится в элементе с ключом `eest` (листинг 11.10).

Листинг 11.10. Список часовых поясов Восточной Европы

```
<?php
  $arr = timezone_abbreviations_list();
  echo "<pre>";
  print_r($arr['eest']);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 11.11 будет следующий дамп массива `$arr`:

```
Array
(
    [0] => Array
        (
            [dst] => 1
            [offset] => 10800
            [timezone_id] => Europe/Helsinki
        )
    ...
    [18] => Array
        (
            [dst] => 1
            [offset] => 10800
            [timezone_id] => Europe/Moscow
        )
    ...
    [31] => Array
        (
            [dst] => 1
            [offset] => 10800
            [timezone_id] => W-SU
        )
)
```

Каждой из часовых зон соответствует один элемент массива, который представляет собой подмассив из трех элементов:

- `timezone_id` — название часовой зоны;
- `offset` — смещение зоны относительно Гринвича;
- `dst` — поле, принимающее значение `TRUE`, если для данной географической области принято переводить часы на летнее время, и `FALSE` в противном случае.

Замечание

В PHP при выводе булевых величин, `TRUE` выводится как 1, а `FALSE` как пустая строка.

Функция `timezone_name_from_abbr()` позволяет искать полные имена часовых зон либо по аббревиатуре, либо по смещению относительно Гринвича. Функция имеет следующий синтаксис:

```
timezone_name_from_abbr($abbr [, $gmt_offset [, $isdst]])
```

В качестве первого параметра `$abbr` функция может принимать сокращенное название часовой зоны. В листинге 11.11 приводится пример использования функции.

Листинг 11.11. Использование функции `timezone_name_from_abbr()`

```
<?php
  echo timezone_name_from_abbr("MSK"); // Europe/Moscow
?>
```

Если неизвестно даже сокращенное название часовой зоны, ее полное название можно получить, указав во втором параметре `$gmt_offset` смещение в секундах относительно Гринвича. Третий параметр `$isdst` должен принимать значение `TRUE`, если в искомой зоне осуществляется перевод часов на летнее время, и `FALSE` в противном случае. В листинге 11.12 приводится пример поиска московского часового пояса по смещению относительно Гринвича. Следует обратить внимание, что в качестве первого параметра передается пустая строка.

Листинг 11.12. Поиск зоны по смещению относительно Гринвича

```
<?php
  echo timezone_name_from_abbr("", 14400, TRUE); // Europe/Moscow
?>
```

PHP предоставляет разработчику функции, позволяющие определять восход и закат солнца для определенной географической точки. Функция `date_sunrise()` позволяет определить время восхода солнца и имеет следующий синтаксис:

```
date_sunrise($timestamp [, $format [, $latitude [, $longitude [, $zenith [,
$gmt_offset]]]]])
```

Функция возвращает время восхода солнца для дня года, задаваемого при помощи параметра `$timestamp` (время задается в формате UNIXSTAMP, количество секунд с нуля часов 1 января 1970 года), широты `$latitude` и долготы `$longitude` в формате, определяемом параметром `$format`, который может принимать следующие значения:

- `SUNFUNCS_RET_STRING` — строковое представление, например, 16:46;
- `SUNFUNCS_RET_DOUBLE` — вещественное представление, например, 16.78243132;
- `SUNFUNCS_RET_TIMESTAMP` — целочисленное представление в формате UNIXSTAMP, например, 1095034606.

Необязательный параметр `$zenith` позволяет задать зенит в градусах (90° для точки горизонта), а параметр `$gmt_offset` — смещение в часах от Гринвича — функция обычно самостоятельно вычисляет такое смещение по широте, однако не учитывает летнее время (именно в этом случае и использует параметр `$gmt_offset`). В листинге 11.13 приводится пример использования функции `date_sunrise()`.

> ***Замечание***
>
> Функция `date_sunset()` принимает аналогичные функции `date_sunrise()` параметры, однако возвращает время заката солнца.

Листинг 11.13. Использование функции `date_sunrise()`

```
<?php
  date_default_timezone_set("Europe/Moscow");
  echo date_sunrise(time(), SUNFUNCS_RET_STRING, 44, 56.3, 90, 4);
?>
```

Функция `date_sun_info()` возвращает время восхода и захода солнца, а также время наступления сумерек, функция имеет следующий синтаксис:

```
date_sun_info($timestamp, $latitude, $longitude)
```

Для дня года, задаваемого при помощи параметра *`$timestamp`* (время задается в формате UNIXSTAMP, количество секунд с нуля часов 1 января 1970 года), широты *`$latitude`* и долготы *`$longitude`*, возвращает информацию о времени восхода и захода солнца, а также начале и конце сумерек. Широта и долгота задаются в градусах. В листинге 11.14 приводится пример использования функции `date_sun_info()`.

Листинг 11.14. Использование функции `date_sun_info()`

```
<?php
  date_default_timezone_set("Europe/Moscow");
  $arr = date_sun_info(time(), 44, 56.3);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

В качестве результата функция `date_sun_info()` возвращает ассоциативный массив:

```
Array
(
    [sunrise] => 1210380811
    [sunset] => 1210433329
    [transit] => 1210407070
    [civil_twilight_begin] => 1210378844
```

```
    [civil_twilight_end] => 1210435296
    [nautical_twilight_begin] => 1210376372
    [nautical_twilight_end] => 1210437768
    [astronomical_twilight_begin] => 1210373555
    [astronomical_twilight_end] => 1210440585
)
```

Элементы результирующего массива представлены в формате UNIXSTAMP и имеют следующие значения:

- `sunrise` — время восхода солнца;
- `sunset` — время заката солнца;
- `transit` — зенит солнца;
- `civil_twilight_begin` — конец утренних гражданских сумерек;
- `civil_twilight_end` — начало вечерних гражданских сумерек;
- `nautical_twilight_begin` — конец утренних навигационных сумерек;
- `nautical_twilight_end` — начало вечерних навигационных сумерек;
- `astronomical_twilight_begin` — конец утренних астрономических сумерек;
- `astronomical_twilight_end` — начало вечерних астрономических сумерек.

11.3. Форматирование даты и времени

Время в формате UNIXSTAMP удобно для компьютерных вычислений, однако плохо воспринимается людьми, поэтому PHP предоставляет разработчикам широкий набор форматирующих функций, а также функций, разбивающих дату и время на отдельные компоненты (табл. 11.3).

Таблица 11.3. *Функции для форматирования даты и времени*

Функция	Описание
`date($format [, $timestamp])`	Форматирует текущую дату в соответствии со строкой, переданной в качестве первого параметра `$format`. Может отформатировать произвольную дату, если она передана в качестве второго необязательного параметра `$timestamp`
`gmdate($format [, $timestamp])`	Функция `gmdate()` аналогична функции `date()`, однако возвращает не текущее локальное время, а время по Гринвичу
`idate($format [, $timestamp])`	Форматирует текущую дату в соответствии со строкой, переданной в качестве первого параметра `$format`. Может отформатировать произвольную дату, если она передана в качестве второго необязательного параметра `$timestamp`. В отличие от функции `date()` возвращает лишь целые значения

***Таблица 11.3** (окончание)*

Функция	Описание
`checkdate($month, $day, $year)`	Проверяет, является ли дата с годом `$year`, месяцем `$month` и днем `$day` корректной, и возвращает `TRUE`, если это так, и `FALSE` в противном случае
`getdate([$timestamp])`	Возвращает текущую дату и время в виде ассоциативного массива. Через необязательный параметр `$timestamp` можно передать произвольную временную метку
`strftime($format [, $timestamp])`	Форматирует текущие дату и время согласно локальным настройкам. Параметр `$format` позволяет задать строку форматирования. Необязательный параметр `$timestamp` позволяет передать на обработку функции произвольную временную метку
`gmstrftime($format [, $timestamp])`	Функция `gmstrftime()` аналогична функции `strftime()`, однако возвращает не текущее локальное время, а время по Гринвичу
`strptime($date, $format)`	Разбирает строку, сгенерированную функцией `strftime()`, и возвращает дату и время в виде ассоциативного массива
`date_parse($date)`	Разбирает строку, сгенерированную функцией `strftime()`, и возвращает дату и время в виде ассоциативного массива
`localtime([$timestamp [, $is_associative]])`	Возвращает текущую дату и время в виде индексного массива (или ассоциативного массива, если параметр `$is_associative` принимает значение `TRUE`). Массив может быть возвращен не только для текущего времени, но и для произвольной временной метки в формате UNIXSTAMP, которая может быть передана через параметр `$timestamp`

Основной функцией для форматирования даты является функция `date()`, которая имеет следующий синтаксис:

```
date($format [, $timestamp])
```

Необязательный параметр `$timestamp` позволяет задать время (в секундах с нуля часов 1 января 1970 года). При его отсутствии функция возвращает текущее время. Параметр `$format` определяет строку форматирования, в которой символы интерпретируются согласно табл. 11.4.

***Таблица 11.4.** Символы форматирования функции `date()`*

Символ	Описание	Пример возвращаемого значения
`a`	Время до (am) и после (pm) полудня в нижнем регистре	am или pm
`A`	Время до (am) и после (pm) полудня в верхнем регистре	AM или PM

Таблица 11.4 *(продолжение)*

Символ	Описание	Пример возвращаемого значения
B	Время в стандарте Swatch Internet	От 000 до 999
c	Дата в формате ISO 8601 (добавлено в PHP 5)	2004-02-12T15:19:21+00:00
d	День месяца, 2 цифры с ведущими нулями	от 01 до 31
D	Сокращенное наименование дня недели, 3 символа	От Mon до Sun
F	Полное наименование месяца, например January или March	От January до December
g	Часы в 12-часовом формате без ведущих нулей	От 1 до 12
G	Часы в 24-часовом формате без ведущих нулей	От 0 до 23
h	Часы в 12-часовом формате с ведущими нулями	От 01 до 12
H	Часы в 24-часовом формате с ведущими нулями	От 00 до 23
i	Минуты с ведущими нулями	От 00 до 59
I	Признак летнего времени	1, если дата соответствует летнему времени, иначе 0
j	День месяца без ведущих нулей	От 1 до 31
l	Полное наименование дня недели	От Sunday до Saturday
L	Признак високосного года	1, если год високосный, иначе 0
m	Порядковый номер месяца с ведущими нулями	От 01 до 12
M	Сокращенное наименование месяца, 3 символа	От Jan до Dec
n	Порядковый номер месяца без ведущих нулей	От 1 до 12
O	Разница со временем по Гринвичу в часах	Например: +0200
r	Дата в формате RFC 2822	Например: Thu, 21 Dec 2000 16:01:07 +0200
s	Секунды с ведущими нулями	От 00 до 59
S	Английский суффикс порядкового числительного дня месяца, 2 символа	st, nd, rd или th. Применяется совместно с j
t	Количество дней в месяце	От 28 до 31

Таблица 11.4 *(окончание)*

Символ	Описание	Пример возвращаемого значения
T	Временная зона на сервере	Примеры: EST, MDT, ...
U	Количество секунд, прошедших с 1 января 1970 года	См. также `time()`
w	Порядковый номер дня недели	От 0 (воскресенье) до 6 (суббота)
W	Порядковый номер недели года по ISO-8601, первый день недели — понедельник	Например: 42 (42-я неделя года)
Y	Порядковый номер года, 4 цифры	Примеры: 1999, 2003
y	Номер года, 2 цифры	Примеры: 99, 03
z	Порядковый номер дня в году (нумерация с 0)	От 0 до 365
Z	Смещение временной зоны в секундах. Для временных зон западнее UTC это отрицательное число, восточнее UTC — положительное	От –43 200 до 43 200

В листинге 11.15 приводится пример использования функции `date()`.

Листинг 11.15. Использование функции `date()`

```
<?php
  date_default_timezone_set("Europe/Moscow");
  echo date("d.m.Y H:i:s"); // 06.12.2008 10:51:53
?>
```

Скрипт из листинга 11.16 выводит приветствие в зависимости от времени суток.

Листинг 11.16. Вывод приветственной фразы в зависимости от времени суток

```
<?php
  date_default_timezone_set("Europe/Moscow");
  $hour = date("G"); // Получаем текущий час
  if ($hour >= 5  && $hour < 11) echo "Доброе утро!";
  if ($hour >= 11 && $hour < 17) echo "Добрый день!";
  if ($hour >= 17 && $hour < 23) echo "Добрый вечер!";
  if ($hour >= 23 || $hour < 5)  echo "Доброй ночи!";
?>
```

Следует отметить, что помимо функции `date()`, возвращающей текущие дату и время (с учетом часовой зоны), разработчикам доступна функция `gmdate()`, которая возвращает время по Гринвичу (листинг 11.17).

Листинг 11.17. Сравнение функций `date()` и `gmdate()`

```
<?php
  date_default_timezone_set("Europe/Moscow");
  echo date("d.m.Y H:i:s"); // 01.01.2008 00:00:00
  echo "<br>";
  echo gmdate("d.m.Y H:i:s"); // 31.12.2007 21:00:00
?>
```

Функция `checkdate()` позволяет проверить, корректна ли дата, и имеет следующий синтаксис:

```
checkdate($month, $day, $year)
```

В качестве первого параметра функция принимает месяц `$month`, второго `$day` — день, в качестве последнего `$year` — год. В листинге 11.18 приводится пример использования функции `checkdate()`.

Листинг 11.18. Использование функции `checkdate()`

```
<?php
  date_default_timezone_set("Europe/Moscow");

  if (checkdate(12, 31, 2000)) // TRUE
    echo "Дата 31.12.2000 корректна<br>";
  else
    echo "Дата 31.12.2000 не корректна<br>";

  if (checkdate(2, 29, 2001)) // FALSE
    echo "Дата 31.12.2000 корректна<br>";
  else
    echo "Дата 31.12.2000 не корректна<br>";
?>
```

Функция `strftime()`, так же как и `date()`, позволяет форматировать дату и имеет следующий синтаксис:

```
strftime($format [, $timestamp])
```

Необязательный параметр `$timestamp` позволяет задать время (в секундах с нуля часов 1 января 1970 года). При его отсутствии функция возвращает текущее время. Параметр `$format` определяет строку форматирования. В форматирующей строке распознаются символы из табл. 11.5.

Таблица 11.5. *Символы форматирования функции* `strftime()`

Символ	Описание
`%a`	Сокращенное название дня недели в текущей локали
`%A`	Полное название дня недели в текущей локали
`%b`	Сокращенное название месяца недели в текущей локали
`%B`	Полное название месяца недели в текущей локали
`%c`	Предпочтительный формат даты и времени в текущей локали
`%C`	Столетие (год, деленный на 100 и округленный до целого, от 00 до 99)
`%d`	День месяца в виде десятичного числа (от 01 до 31)
`%D`	Аналогично `%m/%d/%y`
`%e`	День месяца в виде десятичного числа, если это одна цифра, то перед ней добавляется пробел (от ' 1' до '31')
`%g`	Подобно `%G`, но без столетия
`%G`	Год, 4-значное число, соответствующее номеру недели по ISO (см. `%V`). Аналогично `%Y`, за исключением того, что если номер недели по ISO соответствует предыдущему или следующему году, используется соответствующий год
`%h`	Аналогично `%b`
`%H`	Номер часа от 00 до 23
`%I`	Номер часа от 01 до 12
`%j`	Номер дня в году (от 001 до 366)
`%m`	Номер месяца (от 01 до 12)
`%M`	Минуты
`%n`	Перевод строки "`\n`"
`%p`	Время до (am) и после (pm) полудня или соответствующие строки в текущей локали
`%r`	Время до (am) и после (pm) полудня
`%R`	Время в 24-часовом формате
`%S`	Секунды
`%t`	Символ табуляции "`\t`"
`%T`	Текущее время, аналогично `%H:%M:%S`
`%u`	Номер дня недели от 1 до 7, где 1 соответствует понедельнику
`%U`	Порядковый номер недели в текущем году. Первым днем первой недели в году считается первое воскресенье года
`%V`	Порядковый номер недели в году по стандарту ISO 8601:1988 от 01 до 53, где 1 соответствует первой неделе в году, в которой как минимум 4 дня принадлежат этому году. Первым днем недели считается понедельник. (Используйте `%G` или `%g` для определения соответствующего года)

Таблица 11.5 *(окончание)*

Символ	Описание
%W	Порядковый номер недели в текущем году. Первым днем первой недели в году считается первый понедельник года
%w	Номер дня недели, 0 соответствует воскресенью
%x	Предпочтительный формат даты без времени в текущей локали
%X	Предпочтительный формат времени без даты в текущей локали
%y	Год без столетия (от 00 до 99)
%Y	Год, включая столетие
%Z	Временная зона в виде смещения, аббревиатуры или полного наименования
%%	Символ %

В листинге 11.19 приводится пример использования функции `strftime()`.

Листинг 11.19. Пример использования функции `strftime()`

```
<?php
  date_default_timezone_set("Europe/Moscow");
  echo strftime("%d.%m.%Y %H:%M:%S"); // 06.12.2008 10:51:53
?>
```

Как видно из табл. 11.5, на функцию `strftime()` влияют установки текущей локали. В листинге 11.20 перед вызовом функции устанавливается русская локаль, поэтому в дате выводится сначала день, потом месяц, потом год, вместо последовательности "месяц, день, год", характерной для американской локали.

Листинг 11.20. Установка русской локали перед вызовом функции `strftime()`

```
<?php
  // Устанавливаем часовой пояс
  date_default_timezone_set("Europe/Moscow");
  // Устанавливаем локаль
  setlocale(LC_TIME, "ru_RU.CP1251");
  // Выводим дату и время в русской локали
  echo strftime("%c"); // 06/12/2008 10:51:53
?>
```

Функция `getdate()` возвращает текущую время и дату в виде ассоциативного массива и имеет следующий синтаксис:

```
getdate([$timestamp])
```

Через необязательный параметр `$timestamp` функции можно передать время (в секундах с нуля часов 1 января 1970 года), в случае отсутствия данного параметра функция работает с текущим временем. Как результат работы, `getdate()` возвращает ассоциативный массив, содержащий ключи, описание которых представлено в табл. 11.6.

Таблица 11.6. *Ключи ассоциативного массива, возвращаемого функцией* `getdate()`

Ключ	Описание	Диапазон
`"seconds"`	Секунды	От 0 до 59
`"minutes"`	Минуты	От 0 до 59
`"hours"`	Часы	От 0 до 23
`"mday"`	Порядковый номер дня месяца	От 1 до 31
`"wday"`	Порядковый номер дня	От 0 (воскресенье) до 6 (суббота)
`"mon"`	Порядковый номер месяца	От 1 до 12
`"year"`	Порядковый номер года, 4 цифры	Примеры: 1999, 2003
`"yday"`	Порядковый номер дня в году (нумерация с 0)	От 0 до 366
`"weekday"`	Полное наименование дня недели	От Sunday до Saturday
`"month"`	Полное наименование месяца	от January до December
`0`	Количество секунд, прошедших с 1 января 1970 года.	Платформозависимое, в большинстве случаев от –2 147 483 648 до 2 147 483 647

В листинге 11.21 приводится пример использования функции `getdate()`.

Листинг 11.21. Использование функции `getdate()`

```
<?php
  $today = getdate();
  echo "<pre>";
  print_r($today);
  echo "</pre>";
?>
```

Результатом работы функции из листинга 11.22 будет следующий дамп массива `$today`:

```
Array
(
    [seconds] => 26
    [minutes] => 35
```

```
    [hours] => 1
    [mday] => 6
    [wday] => 1
    [mon] => 12
    [year] => 2004
    [yday] => 340
    [weekday] => Monday
    [month] => December
    [0] => 1102286126
)
```

Функция `strptime()` разбирает строку, сгенерированную функцией `strftime()`, и возвращает дату и время в виде ассоциативного массива. Функция имеет следующий синтаксис:

```
strptime($date, $format)
```

В качестве первого параметра `$date` функция принимает дату и время в виде строки, в качестве второго параметра *`$format`* — строку форматирования, описывающую строку *`$date`*. Как результат работы, `strptime()` возвращает ассоциативный массив, содержащий ключи, описание которых представлено в табл. 11.7.

Таблица 11.7. *Ключи ассоциативного массива, возвращаемого функцией* `strptime()`

Ключ	Описание	Диапазон
`"tm_sec"`	Секунды	От 0 до 59
`"tm_min"`	Минуты	От 0 до 59
`"tm_hour"`	Часы	От 0 до 23
`"tm_mday"`	Порядковый номер дня месяца	От 1 до 31
`"tm_mon"`	Порядковый номер месяца	От 0 до 11
`"tm_year"`	Порядковый номер года, начиная с 1900	Примеры: 108 (для 2008)
`"tm_wday"`	День недели, начиная с воскресенья (0)	От 0 до 6
`"tm_yday"`	День в году	От 0 до 365
`"month"`	Полное наименование месяца	От January до December
`"unparsed"`	Часть строки `$date`, которая не удовлетворяет формату `$format`	—

В листинге 11.22 приводится пример совместного использования функций `strftime()` и `strptime()`.

> ***Замечание***
>
> Функция `strptime()` не поддерживается в операционной системе Windows.

Листинг 11.22. Использование функции `strptime()`

```
<?php
  date_default_timezone_set("Europe/Moscow");

  $format = '%d.%m.%Y %H:%M:%S';
  $strf = strftime($format);

  $arr = strptime($strf, $format);

  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 11.22 будет следующий дамп массива `$arr`:

```
Array
(
    [tm_sec] => 53
    [tm_min] => 51
    [tm_hour] => 10
    [tm_mday] => 5
    [tm_mon] => 11
    [tm_year] => 108
    [tm_wday] => 6
    [tm_yday] => 340
    [unparsed] =>
)
```

Еще одной функцией, предназначенной для разбора строки, сгенерированной функцией `strftime()`, является функция `date_parse()`, которая имеет следующий синтаксис:

```
date_parse($date)
```

Функция принимает в качестве единственного параметра отформатированную дату, а возвращает ассоциативный массив, содержащий ключи, описание которых представлено в табл. 11.8.

Таблица 11.8. *Ключи ассоциативного массива, возвращаемого функцией* `date_parse()`

Ключ	Описание	Диапазон
`"year"`	Года, 4 числа	Например, 1999 или 2008
`"month"`	Месяцы	От 1 до 12

***Таблица 11.8** (окончание)*

Ключ	Описание	Диапазон
"hour"	Часы	От 0 до 23
"minute"	Минуты	От 0 до 59
"second"	Секунды	От 0 до 59
"fraction"	Доли секунды	Например: 0.5
"warning_count"	Количество предупреждений	—
"warnings"	Массив с предупреждениями	—
"error_count"	Количество ошибок	—
"errors"	Массив с ошибками	—
"is_localtime"	Являются ли текущие дата и время локальными	—

В листинге 11.23 приводится пример использования функции `date_parse()`.

Листинг 11.23. Использование функции `date_parse()`

```
<?php
  // Устанавливаем часовой пояс
  date_default_timezone_set("Europe/Moscow");

  $arr = date_parse("2006-12-31 10:00:00.5");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 11.23 будет следующий дамп массива `$arr`:

```
Array
(
    [year] => 2006
    [month] => 12
    [day] => 31
    [hour] => 10
    [minute] => 0
    [second] => 0
    [fraction] => 0.5
    [warning_count] => 0
    [warnings] => Array
```

```
        (
        )

    [error_count] => 0
    [errors] => Array
        (
        )

    [is_localtime] =>
)
```

Как видно из результатов, ни ошибок, ни предупреждений результирующий массив не содержит, однако стоит только использовать некорректную дату, например, "2006-12-32 10:00:00.5", в массив будет добавлено предупреждение.

```
Array
(
    [year] => 2006
    [month] => 12
    [day] => 3
    [hour] => 10
    [minute] => 0
    [second] => 0
    [fraction] => 0.5
    [warning_count] => 0
    [warnings] => Array
        (
        )

    [error_count] => 1
    [errors] => Array
        (
            [9] => Unexpected character
        )

    [is_localtime] =>
)
```

Еще одной функцией, предназначенной для получения компонентов даты и времени в виде ассоциативного массива, является функция `localtime()`, которая имеет следующий синтаксис:

```
localtime([$timestamp [, $is_associative]])
```

Функция возвращает текущую дату и время в виде индексного массива (или ассоциативного массива, если параметр `$is_associative` принимает значение `TRUE`). Массив

может быть возвращен не только для текущего времени, но и для произвольной временной метки в формате UNIXSTAMP, которая может быть передана через параметр `$timestamp`. В табл. 11.9 представлены ключи результирующего массива, возвращаемого функцией `localtime()`.

***Таблица 11.9.** Ключи ассоциативного массива, возвращаемого функцией `localtime()`*

Ключ	Описание	Диапазон
`"tm_sec"`	Секунды	От 0 до 59
`"tm_min"`	Минуты	От 0 до 59
`"tm_hour"`	Часы	От 0 до 23
`"tm_mday"`	День месяца	От 0 до 30
`"tm_mon"`	Месяцы	От 0 до 11
`"tm_year"`	Порядковый номер года, начиная с 1900	Примеры: 108 (для 2008)
`"tm_wday"`	День недели, начиная с воскресенья (0)	От 0 до 6
`"tm_yday"`	День в году	От 0 до 365
`"tm_isdst"`	Летнее время	1 (да) или 0 (нет)

В листинге 11.24 приводится пример использования функции `localtime()`.

Листинг 11.24. Использование функции `localtime()`

```
<?php
  date_default_timezone_set("Europe/Moscow");

  $arr = localtime();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";

  $arr = localtime(time(), TRUE);
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 11.24 могут быть следующие дампы:

```
Array
(
    [0] => 6
    [1] => 43
```

```
    [2] => 3
    [3] => 10
    [4] => 4
    [5] => 108
    [6] => 6
    [7] => 130
    [8] => 1
)
Array
(
    [tm_sec] => 6
    [tm_min] => 43
    [tm_hour] => 3
    [tm_mday] => 10
    [tm_mon] => 4
    [tm_year] => 108
    [tm_wday] => 6
    [tm_yday] => 130
    [tm_isdst] => 1
)
```

Глава 12

Математические функции

В данной главе будут рассмотрены математические функции, которые являются неотъемлемым элементом любого языка программирования.

12.1. Предопределенные константы

Язык программирования PHP предоставляет разработчикам ряд предопределенных констант, полезных в математических вычислениях. Список констант и их значения представлены в табл. 12.1.

Замечание

Префикс `M` в именах предопределенных констант из табл. 12.1 означает "Mathematical" (Математический).

Таблица 12.1. *Предопределенные константы для математических вычислений*

Константа	Значение	Математическое значение
`M_PI`	3.14159265358979323846	Число π
`M_E`	2.7182818284590452354	Экспонента *e*
`M_LOG2E`	1.4426950408889634074	$\log_2(e)$
`M_LOG10E`	0.43429448190325182765	$\lg(e)$
`M_LN2`	0.69314718055994530942	$\ln(2)$
`M_LN10`	2.30258509299404568402	$\ln(10)$
`M_PI_2`	1.57079632679489661923	$\pi/2$
`M_PI_4`	0.78539816339744830962	$\pi/4$
`M_1_PI`	0.31830988618379067154	$1/\pi$
`M_2_PI`	0.63661977236758134308	$2/\pi$
`M_SQRTPI`	1.77245385090551602729	$\sqrt{\pi}$
`M_2_SQRTPI`	1.12837916709551257390	$2/\sqrt{\pi}$
`M_SQRT2`	1.41421356237309504880	$\sqrt{2}$

***Таблица 12.1** (окончание)*

Константа	Значение	Математическое значение
M_SQRT3	1.73205080756887729352	$\sqrt{3}$
M_SQRT1_2	0.70710678118654752440	$1/\sqrt{2}$
M_LNPI	1.14472988584940017414	$\ln(\pi)$
M_EULER	0.57721566490153286061	Постоянная Эйлера, *C*

Константы из табл. 12.1 в любой момент можно вызывать в PHP-скрипте без предварительного объявления (листинг 12.1). В вычислениях лучше отдавать предпочтение предопределенным константам — результат получается точнее, чем при использовании пользовательских констант.

> ***Замечание***
>
> Число π возвращает также функция `pi()`, которая более подробно рассматривается в *разд. 12.7*.

Листинг 12.1. Обращение к предопределенной константе `M_PI`

```
<?php
  echo M_PI; // 3.14159265359
?>
```

12.2. Поиск максимума и минимума

Для поиска максимального и минимального значений среди последовательности целых и вещественных чисел предназначены соответственно функции `max()` и `min()`, синтаксис которых представлен в табл. 12.2.

***Таблица 12.2.** Функции поиска максимума и минимума*

Функция	Описание
`max($value1, [$value2 [, $value3...]])`	Возвращает максимальное значение среди параметров
`min($value1, [$value2 [, $value3...]])`	Возвращает минимальное значение среди параметров

Как видно из синтаксиса, возможны два варианта использования функций: с одним параметром и несколькими. Если функции передается один аргумент, он обязательно должен быть массивом, и в этом случае возвращается минимальный элемент этого массива (листинг 12.2).

Листинг 12.2. Максимальное значение элементов массива

```
<?php
  // Формируем массив из 5 чисел
  $arr = array(2.05, 45, 2.31, 22.5, 1);
  echo max($arr); // 45
?>
```

Если функция принимает несколько числовых аргументов, то они сравниваются между собой и возвращается наименьшее (листинг 12.3).

Листинг 12.3. Минимальное значение элементов массива

```
<?php
  echo min(1, 2, -1, -5, -7, 8); // -7
?>
```

Интересно отметить, что в качестве элементов могут выступать массивы, в этом случае элементы массивов сравниваются поочередно, начиная с первого. Если первый неодинаковый элемент оказывается большим (или меньшим), массив признается большим (или меньшим). В листинге 12.4 приводится пример поиска максимального массива.

Листинг 12.4. Поиск максимального массива

```
<?php
  // Поиск максимального массива
  $arr = max(array(2, 3, 3),
             array(2, 5, 10),
             array(2, 1, 2));
  // Выводим дамп максимального массива
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Как уже упоминалось ранее при сравнении строк в числовом контексте, PHP пытается привести содержимое строки к числу и, если это не удается, приводит строку к нулю. Поэтому в подавляющем большинстве случаев строки в качестве параметров функций `max()` и `min()` выступают как нулевые значения. Однако в качестве результата возвращается строка, а не 0 (листинг 12.5).

Листинг 12.5. Использование строк в качестве аргументов функций `max()` и `min()`

```
<?php
  echo max(0, 'hello');  // 0
  echo max('hello', 0);  // hello
  echo max('hello', -1); // -1
?>
```

12.3. Генерация случайных чисел

В табл. 12.3 представлены функции, предназначенные для генерации случайных чисел.

> ***ЗАМЕЧАНИЕ***
>
> В *разд. 6.15* описываются функции `array_rand()` и `shuffle()`, предназначенные для получения случайных последовательностей при работе с массивами.

Таблица 12.3. *Функции генерации случайных чисел*

Функция	Описание
`rand([$min, $max])`	Генерирует случайно число в интервале от `$min` до `$max`. Если параметры `$min` и `$max` отсутствуют, возвращается число, лежащее в пределах от 0 до `RAND_MAX` (32 768)
`mt_rand([$min, $max])`	Более быстрая версия функции `rand()`, обладающая более равномерным распределением случайных значений (что важно с точки зрения криптографии)
`srand([$seed])`	До версии PHP 4.2 генератор случайных чисел необходимо было инициировать при помощи функции `srand()`, в противном случае каждый раз генерировалась одинаковая последовательность. В более поздних версиях необходимость в этом отпала, и функцию можно считать устаревшей. Необязательный параметр `$seed` позволяет задать инициализирующее значение, как правило, привязанное к текущему времени
`mt_srand([$seed])`	Синоним для функции `srand()`
`getrandmax()`	Возвращает максимальное значение, которое можно получить, используя для генерации случайных чисел функцию `rand()`
`mt_getrandmax()`	Возвращает максимальное значение, которое можно получить, используя для генерации случайных чисел функцию `mt_rand()`
`lcg_value()`	Генерирует и возвращает случайное дробное число, лежащее в диапазоне от 0 до 1

Функция `rand()` генерирует случайное число и имеет следующий синтаксис:

```
rand([$max, $min])
```

Оба аргумента функции необязательны; если же они указаны, то обозначают интервал, в котором должно лежать генерируемое случайное число. Если параметры `$min` и `$max` отсутствуют, возвращается число, лежащее в пределах от 0 до `RAND_MAX`. В листинге 12.6 приводится пример использования функции `rand()`.

> ***Замечание***
>
> Значение `RAND_MAX` принимает значение 32 768. Константа `RAND_MAX` является внутренней константой, поэтому к ней невозможно обратиться непосредственно, для получения ее значения из скрипта следует воспользоваться функцией `getrandmax()`.

Листинг 12.6. Использование функции `rand()`

```
<?php
  echo rand();      // 18889
  echo rand(0,100); // 70
?>
```

В версиях PHP ниже 4.2 для корректной работы данной функции перед ее использованием нужно проинициализировать генератор случайных чисел функцией `srand()` (листинг 12.7). В современных версиях PHP в этом нет необходимости.

Листинг 12.7. Генерация случайного числа

```
<?php
  srand((float) microtime()*1000000);
  echo rand();
?>
```

Функция `getrandmax()` возвращает максимальное значение, которое можно получить, используя для генерации случайных чисел функцию `rand()` (листинг 12.8).

Листинг 12.8. Использование функции `getrandmax()`

```
<?php
  echo getrandmax(); // 32767
?>
```

Функция `lcg_value()` генерирует и возвращает случайное дробное число, лежащее в диапазоне от 0 до 1 (листинг 12.9).

Листинг 12.9. Использование функции `lcg_value()`

```
<?php
  echo lcg_value(); // 0.064963268024138
?>
```

12.4. Преобразование значений между различными системами счисления

Функции данной группы предназначены для преобразования значений из одной системы счисления в другую. Полный список функций и их синтаксис представлены в табл. 12.4.

Таблица 12.4. *Функции преобразования между различными системами счисления*

Функция	Описание
`base_convert($number, $frombase, $tobase)`	Переводит число `$number` из системы счисления по основанию `$frombase` в систему по основанию `$tobase`. `$frombase` и `$tobase` принимают значения только от 2 до 36 включительно. В строке `$number` цифры обозначают сами себя, символ `a` соответствует числу 11, `b` — числу 12 и т. д. Символ `z` обозначает число 36
`decbin($number)`	Преобразует десятичное число `$number` в двоичное
`decoct($number)`	Преобразует десятичное число `$number` в восьмеричное
`dechex($number)`	Преобразует десятичное число `$number` в шестнадцатеричное
`bindec($binary)`	Преобразует двоичное число `$binary` в десятичное
`hexdec($hex)`	Преобразует восьмеричное число `$hex` в десятичное
`octdec($octal)`	Преобразует восьмеричное число `$octal` в десятичное
`deg2rad($number)`	Преобразует градусы `$number` в радианы
`rad2deg($number)`	Преобразует радианы `$number` в градусы

ЗАМЕЧАНИЕ

Десятичные, двоичные, восьмеричные и шестнадцатеричные числа более подробно обсуждаются в *разд. 3.3*.

Функция `base_convert()` преобразует число из одной системы счисления в другую и имеет следующий синтаксис:

```
base_convert($number, $frombase, $tobase)
```

Функция переводит число `$number` из системы счисления по основанию `$frombase` в систему по основанию `$tobase`. Параметры `$frombase` и `$tobase` принимают значения только от 2 до 36 включительно. В строке `$number` цифры обозначают сами себя, символ `a` соответствует числу 11, `b` — числу 12 и т. д. Символ `z` обозначает число 36. Для того чтобы, к примеру, перевести число 300 из десятичной системы в шестнадцатеричную, аргумент `$frombase` указывается равным 10, а аргумент `$tobase` должен быть равен соответственно 16 (листинг 12.10).

Листинг 12.10. Перевод чисел между разными системами счисления

```
<?php
  echo base_convert("30", 10, 16);  // 1E
  echo base_convert("F", 16, 2);    // 1111
  echo base_convert("1111", 2, 16); // F
  echo base_convert("3", 10, 2);    // 11
?>
```

Использовать столь универсальную функцию, как `base_convert()`, не всегда удобно, поэтому PHP предоставляет разработчикам ряд специализированных функций, которые рассматриваются далее.

Функция `decbin()` преобразует десятичное число в двоичное и имеет следующий синтаксис:

```
decbin($number)
```

В качестве аргумента функция принимает число в десятичной системе счисления `$number`, а возвращает его в двоичном представлении (листинг 12.11).

Листинг 12.11. Преобразование десятичного числа в двоичное

```
<?php
  echo decbin(30); // 11110
?>
```

Функция `decoct()` преобразует десятичное число в восьмеричное и имеет следующий синтаксис:

```
decoct($number)
```

В качестве аргумента функция принимает число в десятичной системе счисления `$number`, а возвращает его в восьмеричном представлении (листинг 12.12).

Листинг 12.12. Преобразование десятичного числа в восьмеричное

```
<?php
  echo decoct(200); // 310
?>
```

Функция `dechex()` преобразует десятичное число в шестнадцатеричное и имеет следующий синтаксис:

```
dechex($number)
```

В качестве аргумента функция принимает число в десятичной системе счисления `$number`, а возвращает его в шестнадцатеричном представлении (листинг 12.13).

Листинг 12.13. Преобразование десятичного числа в шестнадцатеричное

```
<?php
  echo dechex(200); // C8
?>
```

Функция `bindec()` преобразует двоичное число в десятичное и имеет следующий синтаксис:

```
bindec($binary)
```

В качестве аргумента функция принимает число в двоичной системе счисления `$binary` (строка из нулей и единиц), а возвращает его в десятичном представлении (листинг 12.14).

Листинг 12.14. Преобразование двоичного числа в десятичное

```
<?php
  echo bindec("11011"); // 27
?>
```

Функция `hexdec()` преобразует шестнадцатеричное число в десятичное и имеет следующий синтаксис:

```
hexdec($hex)
```

В качестве аргумента функция принимает число в шестнадцатеричной системе счисления `$hex`, а возвращает его в десятичном представлении (листинг 12.15).

Листинг 12.15. Преобразование шестнадцатеричного числа в десятичное

```
<?php
  echo hexdec(0xFF0E0); // 1044704
?>
```

Функция `octdec()` преобразует восьмеричное число в десятичное и имеет следующий синтаксис:

```
octdec($octal)
```

В качестве аргумента функция принимает число в восьмеричной системе счисления `$octal`, а возвращает его в десятичном представлении (листинг 12.16).

Листинг 12.16. Преобразование восьмеричного числа в десятичное

```
<?php
  echo octdec(1123544545); // 156158309
?>
```

Функция `deg2rad()` преобразует градусы в радианы и имеет следующий синтаксис:

```
deg2rad($number)
```

В качестве аргумента функция принимает градусы `$number`, а возвращает радианы (листинг 12.17).

Листинг 12.17. Преобразование градусов в радианы

```
<?php
  echo deg2rad(90.00); // 1.5707963267949
?>
```

Функция `rad2deg()` выполняет обратную функции `deg2rad()` задачу — преобразует радианы в градусы и имеет следующий синтаксис:

```
rad2deg($number)
```

В качестве аргумента функция принимает радианы `$number`, а возвращает градусы (листинг 12.18).

Листинг 12.18. Преобразование радиан в градусы

```
<?php
  echo rad2deg(1.5707963267949); // 90
?>
```

12.5. Округление чисел

Функции данной группы предоставляют разработчику возможность округления чисел по различным правилам (в том числе отличным от правил математического округления). В табл. 12.5 представлен список функций округления.

Таблица 12.5. *Функции округления чисел*

Функция	Описание
`abs($number)`	Возвращает модуль числа `$number`
`round($value [, $precision])`	Округляет число `$value`, если указан необязательный параметр `$precision` — округление осуществляется до `$precision` знака после запятой
`ceil($value)`	Округляет число `$value` до ближайшего целого числа, причем результат всегда больше `$value`
`floor($value)`	Округляет число `$value` до ближайшего целого числа, причем результат всегда меньше `$value`
`fmod($x, $y)`	Возвращает остаток деления двух чисел `$x` и `$y`

Функция `abs()` возвращает модуль числа и имеет следующий синтаксис:

```
abs($number)
```

Если аргумент `$number` является отрицательным числом, его знак меняется на положительный, если же в качестве аргумента передается положительное число — оно возвращается без изменений (листинг 12.19).

Листинг 12.19. Использование функции `abs()`

```
<?php
  echo abs(-4.5); // 4.5
  echo abs(6);    // 6
?>
```

Функция `round()` производит математическое округление числа и имеет следующий синтаксис:

```
round($value [, $precision])
```

В качестве первого аргумента функция принимает число `$value` и возвращает округленное число. Второй необязательный параметр `$precision` позволяет задать количество знаков после запятой, которое должно остаться в результате, если оно не указано, считается, что округление должно быть произведено до целых. В листинге 12.20 приводится пример работы с функцией `round()`.

Листинг 12.20. Использование функции `round()`

```
<?php
  echo round(3.4);          // 3
  echo round(3.5);          // 4
  echo round(3.6);          // 4
  echo round(3.6, 0);       // 4
  echo round(1.95583, 2);   // 1.96
  echo round(1241757, -3); // 1242000
  echo round(5.045, 2);     // 5.05
  echo round(5.055, 2);     // 5.06
?>
```

Функция `ceil()` производит округление до ближайшего целого числа, причем результат всегда больше округляемого числа. Функция имеет следующий синтаксис:

```
ceil($value)
```

В листинге 12.21 приводится пример использования функции `ceil()`.

> ***Замечание***
>
> В отличие от функции `round()`, функция `ceil()` производит округление только до целых чисел.

Листинг 12.21. Использование функции `ceil()`

```
<?php
  echo ceil(4.3);    // 5
  echo ceil(9.999);  // 10
  echo ceil(-4.7);   // -4
  echo ceil(-9.999); // -9
?>
```

Функция `floor()` производит округление до ближайшего целого числа, причем результат всегда больше округляемого числа. Функция имеет следующий синтаксис:

```
floor($value)
```

В листинге 12.22 приводится пример использования функции `floor()`.

> ***Замечание***
>
> В отличие от функции `round()`, функция `floor()` производит округление только до целых чисел.

Листинг 12.22. Использование функции `floor()`

```
<?php
  echo floor(4.3);     //  4
  echo floor(9.999);   //  9
  echo floor(-4.7);    // -5
  echo floor(-9.999);  // -10
?>
```

Функция `fmod()` возвращает остаток деления двух чисел и имеет следующий синтаксис:

```
fmod($x, $y)
```

Остаток определяется как `x - i*y`, где `i` — максимально возможное целое число. В листинге 12.23 приводится пример использования функции `fmod()`.

Листинг 12.23. Использование функции `fmod()`

```
<?php
  echo fmod(10, 3);    // 1
  echo fmod(5.7, 1.3); // 0.5
?>
```

12.6. Логарифмические и степенные функции

В табл. 12.6 приводится список логарифмических и степенных функций, доступных разработчикам.

Таблица 12.6. *Логарифмические и степенные функции*

Функция	Описание
`pow($base, $exp)`	Возвращает число `$base` в степени `$exp`
`exp($number)`	Возвращает экспоненту в степени `$number`
`expm1($number)`	Возвращает экспоненту в степени (`$number` – 1)
`sqrt($number)`	Возвращает корень квадратный числа `$number`
`log($number [, $base])`	Возвращает логарифм числа `$number` по основанию `$base`, если параметр `$base` не указан — возвращается натуральный логарифм
`log10($number)`	Возвращает десятичный логарифм числа `$number`
`log1p($number)`	Возвращает натуральный логарифм от суммы `$number` + 1

Функция `pow()` производит возведение числа в произвольную степень и имеет следующий синтаксис:

```
pow($base, $exp)
```

Функция возвращает число `$base` в степени `$exp`. Пример использования функции приводится в листинге 12.24. Допускается как дробная, так и отрицательная степень, если степень не может быть вычислена, функция `pow()` возвращает `FALSE`.

Листинг 12.24. Возведение в степень

```
<?php
  echo pow(2, 8)   ."<br>";  // 256
  echo pow(2, 1.3) ."<br>";  // 2.46228882669
  echo pow(2, -1)  ."<br>";  // 0.5
  echo pow(0, 0)   ."<br>";  // 1
  echo pow(-1, 10) ."<br>";  // -1
  echo pow(-1, 5.5)."<br>";  // NAN
?>
```

В последнем примере производится попытка вычислить дробную степень от отрицательного числа, в результате чего функция возвращает константу `NAN` — признак того, что результат является неопределенным числом. Для идентификации таких значений существует специальная функция `is_nan()`, синтаксис которой более подробно рассматривается в *разд. 12.8*.

Для получения экспоненциальной степени предназначена специализированная функция `exp()`, которая имеет следующий синтаксис:

```
exp($number)
```

В листинге 12.25 приводится пример работы с функцией `exp()` для вычисления e^2.

Листинг 12.25. Вычисление e^2

```
<?php
  echo exp(2); // 7.3890560989307
?>
```

Впрочем, вычислить экспоненциальную степень можно также при помощи функции `pow()` и предопределенной константы `M_E` (листинг 12.26).

Листинг 12.26. Альтернативный способ вычисления e^2

```
<?php
  echo pow(M_E, 2); // 7.38905609893
?>
```

В состав PHP входят еще более специализированные функции, например, `expm1()`, которая вычисляет экспоненциальную степень числа и вычитает единицу: $e^n - 1$. Функция имеет следующий синтаксис:

```
expm1($number)
```

Особенностью функции является тот факт, что результат вычисляется очень точно, даже если значение `$number` лежит очень близко к нулю. В листинге 12.27 приводится пример, позволяющий сравнить функцию `expm1()` и выражение `exp($number) - 1`.

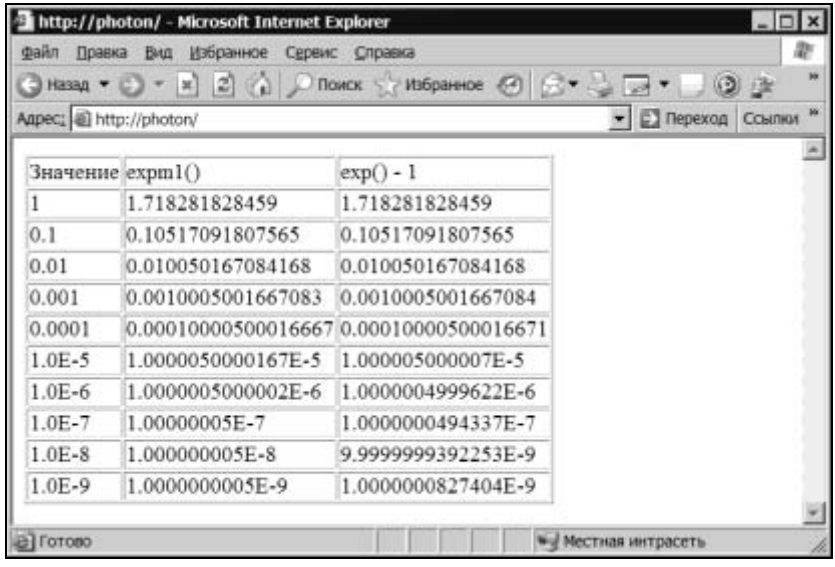
> ***ЗАМЕЧАНИЕ***
>
> В версии PHP, работающей под операционной системой Windows, вместо функции `expm1()` вставлена заглушка, которая использует функцию `exp()`, поэтому вместо точных вычислений вблизи нуля, функция `expm1()` выдает результаты, аналогичные выражению `exp($number) - 1`.

Листинг 12.27. Сравнение функций `expm1()` и `exp()`

```
<?php
  $arr = array(1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001,
               0.0000001, 0.00000001, 0.000000001);
  echo "<table border='1'>";
  echo "<tr>
          <td>Значение</td>
          <td>expm1()</td>
          <td>exp() - 1</td>
```

```
        </tr>";
  foreach($arr as $number)
  {
    echo "<tr>
            <td>$number</td>
            <td>".expm1($number)."</td>
            <td>".(exp($number) - 1)."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Начиная где-то со значений меньших 0.00001, выражение `exp($number) - 1` выдает ошибочные значения, в то время как функция `expm1()` выдает точные (рис. 12.1).



Значение	expm1()	exp() - 1
1	1.718281828459	1.718281828459
0.1	0.10517091807565	0.10517091807565
0.01	0.010050167084168	0.010050167084168
0.001	0.0010005001667083	0.0010005001667084
0.0001	0.00010000500016667	0.00010000500016671
1.0E-5	1.0000050000167E-5	1.000005000007E-5
1.0E-6	1.0000005000002E-6	1.0000004999622E-6
1.0E-7	1.00000005E-7	1.0000000494337E-7
1.0E-8	1.000000005E-8	9.9999999392253E-9
1.0E-9	1.0000000005E-9	1.0000000827404E-9

Рис. 12.1. Сравнение функций `expm1()` и `exp()` в окрестностях нуля

Функция `sqrt()` вычисляет квадратный корень числа и имеет следующий синтаксис:

`sqrt($`*number*`)`

В листинге 12.28 приводится пример работы с функцией `sqrt()`.

Листинг 12.28. Вычисление квадратного корня при помощи функции `sqrt()`

```
<?php
  echo sqrt(4);  // 2
  echo sqrt(-4); // NAN
?>
```

Как видно из листинга 12.28, при попытке вычисления корня квадратного из отрицательного числа возвращается константа `NAN`, сигнализирующая о том, что результат не является неопределенным числом. Квадратный корень может быть вычислен также при помощи функции `pow()`, для этого достаточно использовать в качестве степени значение 0.5 (листинг 12.29).

Листинг 12.29. Вычисление квадратного корня при помощи функции `pow()`

```
<?php
  echo pow(4, 0.5);  // 2
?>
```

Функция `log()` возвращает логарифм и имеет следующий синтаксис:

```
log($number [, $base])
```

Функция возвращает логарифм числа `$number` по основанию `$base`, если параметр `$base` не указан — возвращается натуральный логарифм. В листинге 12.30 приводится пример использования функции `log()`.

Листинг 12.30. Вычисление натурального логарифма

```
<?php
  echo log(15); // 2.7080502011
?>
```

Помимо натурального логарифма в математических вычислениях используется десятичный логарифм. В связи с этим в PHP введена специализированная функция для его вычисления, которая имеет следующий синтаксис:

```
log10($number)
```

В листинге 12.31 приводится пример использования функции `log10()`.

Листинг 12.31. Вычисление десятичного логарифма

```
<?php
  echo log10(15); // 1.17609125906
?>
```

Функция `log1p()`, так же как и функция `expm1()`, предназначена для точных вычислений вблизи единицы и имеет следующий синтаксис:

```
log1p($number)
```

Функция возвращает натуральный логарифм от суммы `$number + 1`. Дело в том, что функция `log()` может возвращать точное значение только для выражения `log(1)`, в окрестностях единицы значения из-за накапливающихся ошибок с плавающей точкой очень не точны — именно для таких вычислений и предназначена функция `log1p()`. В листинге 12.32 приводится пример, позволяющий сравнить функцию `log1p()` и выражение `log($number + 1)`.

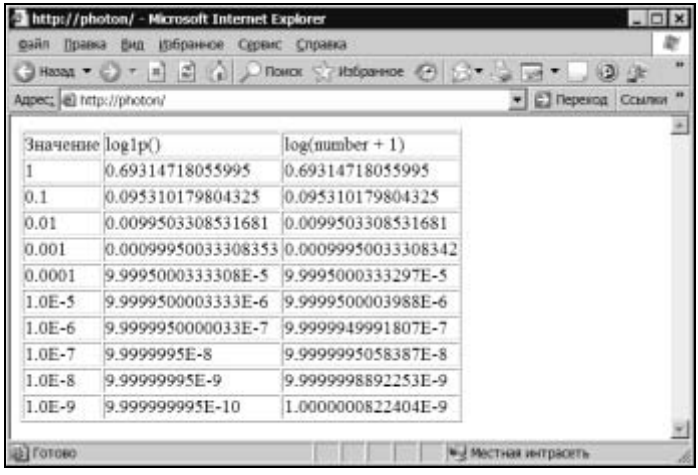
Замечание

В версии PHP, работающей под операционной системой Windows, вместо функции `log1p()` вставлена заглушка, которая использует функцию `log()`, поэтому вместо точных вычислений вблизи единицы, функция `log1p()` выдает результаты аналогичные выражению `log($number + 1)`.

Листинг 12.32. Сравнение функций `log1p()` и `log()`

```
<?php
  $arr = array(1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001,
               0.0000001, 0.00000001, 0.000000001);
  echo "<table border=1>";
  echo "<tr>
          <td>Значение</td>
          <td>log1p()</td>
          <td>log(number + 1)</td>
        </tr>";
  foreach($arr as $number)
  {
    echo "<tr>
            <td>$number</td>
            <td>".log1p($number)."</td>
            <td>".log($number + 1)."</td>
          </tr>";
  }
  echo "</table>";
?>
```

На рис. 12.2 представлен результат работы скрипта из листинга 12.31 в операционной системе Linux.



Значение	log1p()	log(number + 1)
1	0.69314718055995	0.69314718055995
0.1	0.095310179804325	0.095310179804325
0.01	0.0099503308531681	0.0099503308531681
0.001	0.00099950033308353	0.00099950033308342
0.0001	9.9995000333308E-5	9.9995000333297E-5
1.0E-5	9.9999500003333E-6	9.9999500003988E-6
1.0E-6	9.9999950000033E-7	9.9999949991807E-7
1.0E-7	9.9999995E-8	9.9999995058387E-8
1.0E-8	9.99999995E-9	9.9999998892253E-9
1.0E-9	9.999999995E-10	1.0000000822404E-9

Рис. 12.2. Сравнение функций `log1p()` и `log()` в окрестностях нуля

12.7. Тригонометрические функции

Язык программирования PHP предоставляет разработчикам классические и гиперболические тригонометрические функции, полный список которых представлен в табл. 12.7.

Таблица 12.7. *Тригонометрические функции*

Функция	Описание
`pi()`	Возвращает число π
`sin($value)`	Возвращает синус угла `$value`, заданного в радианах
`sinh($value)`	Возвращает гиперболический синус числа `$value`
`cos($value)`	Возвращает косинус угла `$value`, заданного в радианах
`cosh($value)`	Возвращает гиперболический косинус числа `$value`
`tan($value)`	Возвращает тангенс угла `$value`, заданного в радианах
`tanh($value)`	Возвращает гиперболический тангенс числа `$value`
`asin($value)`	Возвращает арксинус `$value` — результат возвращается в радианах
`asinh($value)`	Возвращает обратный гиперболический синус числа `$value`
`acos($value)`	Возвращает арккосинус `$value` — результат возвращается в радианах
`acosh($value)`	Возвращает обратный гиперболический косинус числа `$value`
`atan($value)`	Возвращает арктангенс `$value` — результат возвращается в радианах
`atan2($y, $x)`	Возвращает арктангенс частного `$y` и `$x`
`atanh($value)`	Возвращает обратный гиперболический тангенс числа `$value`
`hypot($x, $y)`	Возвращает длину одной из сторон треугольника по длинам двух других сторон

Функция `pi()` возвращает число π и имеет следующий синтаксис:

```
pi()
```

В листинге 12.33 приводится пример использования функции.

Листинг 12.33. Вывод числа π

```
<?php
  echo pi(); // 3.14159265359
?>
```

Вместо функции `pi()` в выражениях можно использовать предопределенную константу `M_PI`, которая также возвращает число π (листинг 12.34).

Листинг 12.34. Использование предопределенной константы `M_PI`

```
<?php
  echo M_PI; // 3.14159265359
?>
```

Функция `sin()` возвращает синус аргумента, заданного в радианах. Функция имеет следующий синтаксис:

```
sin($value)
```

В листинге 12.35 приводится пример использования функции `sin()`.

Листинг 12.35. Вычисление синуса при помощи функции `sin()`

```
<?php
  echo sin(M_PI_2); // 1
?>
```

`M_PI_2` является предопределенной константой библиотеки PHP для работы с математическими функциями.

Тригонометрические функции принимают значения в радианах, поэтому, если удобнее вести вычисления в градусах, необходимо перед передачей аргумента функции `sin()` преобразовывать значение аргумента в радианы (листинг 12.36).

> ***ЗАМЕЧАНИЕ***
>
> Функция `deg2rad()` описывается в *разд. 12.4*.

Листинг 12.36. Совместное использование функций `sin()` и `deg2rad()`

```
<?php
  echo sin(deg2rad(90)); // 1
?>
```

Функция `sinh()` возвращает гиперболический синус и имеет следующий синтаксис:

```
sinh ($value)
```

Функция вычисляет выражение по формуле `(exp($value) - exp(-$value))/2`. В листинге 12.37 приводится пример использования функции `sinh()`.

Листинг 12.37. Использование функции `sinh()`

```
<?php
  echo sinh(1);              // 1.17520119364
  echo (exp(1) - exp(-1))/2; // 1.17520119364
?>
```

Функция `cos()` возвращает косинус аргумента, заданного в радианах. Функция имеет следующий синтаксис:

```
cos($value)
```

В листинге 12.38 приводится пример использования функции `cos()`.

Листинг 12.38. Вычисление косинуса при помощи функции `cos()`

```
<?php
  echo cos(M_PI); // -1
?>
```

Функция `cosh()` возвращает гиперболический косинус и имеет следующий синтаксис:

```
cosh ($value)
```

Функция вычисляет выражение по формуле `(exp($value) + exp(-$value))/2`. В листинге 12.39 приводится пример использования функции `cosh()`.

Листинг 12.39. Использование функции `cosh()`

```
<?php
  echo cosh(1);              // 1.54308063482
  echo (exp(1) + exp(-1))/2; // 1.54308063482
?>
```

Функция `tan()` возвращает тангенс аргумента, заданного в радианах. Функция имеет следующий синтаксис:

```
tan($value)
```

Функция вычисляет выражение по формуле `sin($value)/cos($value)`. В листинге 12.40 приводится пример использования функции `tan()`.

Листинг 12.40. Вычисление тангенса при помощи функции `tan()`

```
<?php
  echo tan(M_PI_4);             // 1
  echo sin(M_PI_4)/cos(M_PI_4); // 1
?>
```

Функция `tanh()` возвращает гиперболический тангенс и имеет следующий синтаксис:

`tanh ($value)`

Функция вычисляет выражение по формуле `sinh($value)/cosh($value)`. В листинге 12.41 приводится пример использования функции `tanh()`.

Листинг 12.41. Использование функции `tanh()`

```
<?php
  echo tanh(1);         // 0.761594155956
  echo sinh(1)/cosh(1); // 0.761594155956
?>
```

Функция `asin()` возвращает арксинус аргумента и имеет следующий синтаксис:

`asin($value)`

В листинге 12.42 приводится пример использования функции `asin()`.

Листинг 12.42. Вычисление арксинуса при помощи функции `asin()`

```
<?php
  echo asin(M_PI_4);      // 0.903339110767
  echo sin(asin(M_PI_4)); // 0.785398163397 == M_PI_4
  echo asin(M_PI_2);      // NAN
?>
```

Функция `asinh()` возвращает обратный гиперболический синус числа и имеет следующий синтаксис:

`asinh($value)`

В листинге 12.43 приводится пример использования функции `asinh()`.

Листинг 12.43. Использование функции `asinh()`

```
<?php
  echo asinh(M_PI_4);       // 0.721225488727
  echo sinh(asinh(M_PI_4)); // 0.785398163397 == M_PI_4
?>
```

Функция `acos()` возвращает арккосинус аргумента и имеет следующий синтаксис:

`acos($value)`

В листинге 12.44 приводится пример использования функции `acos()`.

Листинг 12.44. Вычисление арккосинуса при помощи функции `acos()`

```
<?php
  echo acos(M_PI_4);      // 0.667457216028
  echo cos(acos(M_PI_4)); // 0.785398163397 == M_PI_4
  echo acos(M_PI_2);      // NAN
?>
```

Функция `acosh()` возвращает обратный гиперболический косинус числа и имеет следующий синтаксис:

```
acosh($value)
```

В листинге 12.45 приводится пример использования функции `acosh()`.

Листинг 12.45. Использование функции `acosh()`

```
<?php
  echo acosh(M_PI_2);       // 1.02322747855
  echo cosh(acosh(M_PI_2)); // 1.57079632679 == M_PI_2
?>
```

Функция `atan()` возвращает арктангенс числа и имеет следующий синтаксис:

```
atan($value)
```

В листинге 12.46 приводится пример использования функции `atan()`.

Листинг 12.46. Вычисление арктангенса при помощи функции `atan()`

```
<?php
  echo atan(M_PI_4);      // 0.665773750028
  echo tan(atan(M_PI_4)); // 0.785398163397 == M_PI_4
?>
```

Функция `atan2()` вычисляет арктангенс частного двух чисел и имеет следующий синтаксис:

```
atan2($y, $x)
```

Функция вычисляет выражение по формуле `atan($x/$y)`. В листинге 12.47 приводится пример использования функции `atan2()`.

Листинг 12.47. Использование функции `atan2()`

```
<?php
  echo atan2(1, 2); // 0.463647609001
  echo atan(1/2);   // 0.463647609001
?>
```

Функция `atanh()` возвращает обратный гиперболический арктангенс числа и имеет следующий синтаксис:

```
atanh($value)
```

В листинге 12.48 приводится пример использования функции `atanh()`.

Листинг 12.48. Использование функции `atanh()`

```
<?php
  echo atanh(M_PI_4);       // 1.05930617082
  echo tanh(atanh(M_PI_4)); // 0.785398163397 == M_PI_4
?>
```

Функция `hypot()` возвращает длину одной из сторон треугольника по длинам двух других сторон и имеет следующий синтаксис:

```
hypot($x, $y)
```

Функция вычисляет выражение по формуле `sqrt($x * $x + $y * $y)`. В листинге 12.49 приводится пример использования функции `hypot()`.

Листинг 12.49. Использование функции `hypot()`

```
<?php
  $x = 1;
  $y = 2;
  echo hypot($x, $y);       // 2.2360679775
  echo sqrt($x*$x + $y*$y); // 2.2360679775
?>
```

12.8. Информационные функции

Информационные функции позволяют ответить на вопрос, является ли содержимое переменным числом (целым или вещественным), бесконечно оно или конечно. Список функций данного раздела представлен в табл. 12.8.

ЗАМЕЧАНИЕ

Часть функций из табл. 12.8 описывается также в *разд. 3.13*.

***Таблица 12.8.** Информационные функции*

Функция	Описание
`is_int($value)`	Возвращает `TRUE`, если переменная `$value` является целым числом, в противном случае возвращается `FALSE`
`is_integer($value)`	Синоним для функции `is_int()`

***Таблица 12.8** (окончание)*

Функция	Описание
is_long($*value*)	Синоним для функции is_int()
is_float($*value*)	Возвращает TRUE, если переменная $*value* является вещественным числом, в противном случае возвращается FALSE
is_real($*value*)	Синоним для функции is_float()
is_double($*value*)	Синоним для функции is_float()
is_numeric($*value*)	Возвращает TRUE, если переменная $*value* является числом или строкой, содержащей число, в противном случае возвращается FALSE
is_nan($*value*)	Возвращает TRUE, если переменная $*value* является неопределенным числом, в противном случае возвращается FALSE
is_infinite($*value*)	Возвращает TRUE, если выражение $*value* выражается бесконечной величиной, в противном случае возвращается FALSE
is_finite($*value*)	Возвращает TRUE, если выражение $*value* выражается конечной величиной, в противном случае возвращается FALSE

Функция is_int() проверяет, является ли переданная ей переменная целым числом, и имеет следующий синтаксис:

is_int($*value*)

Функция возвращает TRUE, если переменная $*value* является целым числом, в противном случае возвращается FALSE (листинг 12.50).

Листинг 12.50. Использование функции is_int()

```
<?php
  if (is_int(23)) // TRUE
    echo "23 — целое число<br>";
  else
   echo "23 — не целое число<br>";

  if (is_int(23.5)) // FALSE
    echo "23.5 — целое число<br>";
  else
   echo "23.5 — не целое число<br>";

  if (is_int("23")) // FALSE
```

```
    echo "\"23\" — целое число<br>";
  else
    echo "\"23\" — не целое число<br>";
?>
```

В результате выполнения скрипта из листинга 12.50 в окно браузера будут выведены следующие строки:

```
23 — целое число
23.5 — не целое число
"23" — не целое число
```

Следует обратить внимание, что функция `is_int()` возвращает `FALSE` не только для дробных (вещественных) чисел, но и строк, даже если они содержат целое число.

Функция `is_float()` проверяет, является ли переданная ей переменная вещественным числом, и имеет следующий синтаксис:

```
is_float($value)
```

Функция возвращает `TRUE`, если переменная `$value` является вещественным числом, в противном случае возвращается `FALSE` (листинг 12.51).

Листинг 12.51. Использование функции `is_float()`

```
<?php
  $arr = array(123.24, "123.24", 123.24e306, 123);
  foreach($arr as $value)
  {
    if (is_float($value))
      echo "Переменная $value имеет тип float<br>";
    else
      echo "Переменная $value имеет тип отличный от float<br>";
  }
?>
```

В результате выполнения скрипта из листинга 12.51 в окно браузера будут выведены следующие строки:

```
Переменная 123.24 имеет тип float
Переменная 123.24 имеет тип отличный от float
Переменная 1.2324E+308 имеет тип float
Переменная 123 имеет тип отличный от float
```

Следует обратить внимание на тот факт, что функция `is_float()` возвращает `FALSE` не только для строк, но и для целых чисел.

Функция `is_numeric()` проверяет, является ли переданная ей переменная числом (целым или вещественным) или строкой, содержащей число. Функция имеет следующий синтаксис:

```
is_numeric($value)
```

Возвращает TRUE, если переменная $*value* является числом или строкой, содержащей число, в противном случае возвращается FALSE (листинг 12.52).

Листинг 12.52. Использование функции is_numeric()

```
<?php
  $arr = array(123.24, "123.24", 123.24e306, 123, "hello");
  foreach($arr as $value)
  {
    if (is_numeric($value))
      echo "Переменная $value является числом<br>";
    else
      echo "Переменная $value не является числом<br>";
  }
?>
```

В результате выполнения скрипта из листинга 12.52 в окно браузера будут выведены следующие строки:

```
Переменная 123.24 является числом
Переменная 123.24 является числом
Переменная 1.2324E+308 является числом
Переменная 123 является числом
Переменная hello не является числом
```

Функция is_nan() проверяет, является ли переданное ей в качестве параметра число неопределенным (NAN), и имеет следующий синтаксис:

```
is_nan($value)
```

Функция возвращает TRUE, если переменная $*value* является неопределенным числом, в противном случае возвращается FALSE. В листинге 12.53 приводится пример использования функции is_nan(). Следует подчеркнуть, что функция возвращает TRUE только в том случае, когда ожидается числовой контекст, поэтому не следует применять функцию для проверки корректности ввода числа — для этого предназначены функции is_number(), is_int() и is_float().

Листинг 12.53. Использование функции is_nan()

```
<?php
  if (is_nan(123)) // FALSE
    echo "Переменная является неопределенным числом<br>";
  else
    echo "Переменная является обычным числом<br>";

  if (is_nan(asin(M_PI_2))) // TRUE
```

```
    echo "Переменная является неопределенным числом<br>";
  else
    echo "Переменная является обычным числом<br>";

  if (is_nan(pow(-1, 5.5))) // TRUE
    echo "Переменная является неопределенным числом<br>";
  else
    echo "Переменная является обычным числом<br>";

  if (is_nan(NAN)) // FALSE
    echo "Переменная является неопределенным числом<br>";
  else
    echo "Переменная является обычным числом<br>";
?>
```

В результате выполнения скрипта из листинга 12.53 в окно браузера будут выведены следующие строки:

```
Переменная является обычным числом
Переменная является неопределенным числом
Переменная является неопределенным числом
Переменная является обычным числом
```

Неопределенное значение `NAN` может участвовать в выражениях, любое выражение со значением `NAN` имеет результатом `NAN` (листинг 12.54).

> ***Замечание***
>
> Интересно отметить, что при попытке деления числа на `NAN`, интерпретатор PHP генерирует предупреждение "Warning: Division by zero" ("Предупреждение: деление на ноль"). Таким образом, неопределенное число ведет себя как 0, что является не совсем корректным. Возможно, в будущих версиях PHP такое поведение неопределенных чисел будет исправлено.

Листинг 12.54. Использование неопределенных чисел в выражениях

```
<?php
  echo pow(-1, 5.5);     // NAN
  echo 1 + pow(-1, 5.5); // NAN
?>
```

Функции `is_infinite()` и `is_finite()` позволяют определить, является ли переданные им числа конечными или бесконечными. Функции имеют следующий синтаксис:

```
is_infinite($value)
is_finite($value)
```

Функция `is_infinite()` возвращает `TRUE`, если `$value` выражается бесконечной величиной, в противном случае возвращается `FALSE`. Функция `is_finite()` возвращает `TRUE`, если `$value` выражается конечной величиной, в противном случае возвращается `FALSE`. Число считается бесконечным, если оно выходит за границы разряда. Например, в листинге 12.55 для числа `123.24e308` функция `is_infinite()` вернет `TRUE`, т. к. это число превышает максимально допустимое для типа `float` значение $\pm 1.7 \times 10^{308}$ *(см. разд. 3.2)*.

Листинг 12.55. Использование функции `is_infinite()`

```
<?php
  if (is_infinite(123)) // FALSE
    echo "Число бесконечно<br>";
  else
    echo "Число конечно<br>";

  if (is_infinite(123.24e308)) // TRUE
    echo "Число бесконечно<br>";
  else
    echo "Число конечно<br>";
?>
```

В результате выполнения скрипта из листинга 12.55 в окно браузера будут выведены следующие строки:

```
Число конечно
Число бесконечно
```

При попытке вывести "бесконечное" число в окно браузера вместо такого числа будет выводиться значение `INF` или `-INF`, если бесконечность отрицательная (листинг 12.56).

Листинг 12.56. Попытка вывода "бесконечного" числа

```
<?php
  echo -123.24e308; // -INF
  echo pow(0, -1);  // INF
  echo 123.24e308;  // INF
?>
```

Интересно отметить, что деление на ноль при помощи выражения `pow(0, -1)` допускается, а попытка деления на ноль при помощи оператора деления `/` вызывает предупреждение "Warning: Division by zero" ("Предупреждение: деление на ноль") и возвращает `FALSE` вместо бесконечности `INF`. Такое поведение является немного непоследовательным и, возможно, будет скорректировано в будущих версиях PHP.

Бесконечные величины (INF) в отличие от неопределенных чисел (NAN) ведут себя как математические бесконечности, например, деление любых конечных значений на бесконечные величины имеет результатом ноль (листинг 12.57).

Листинг 12.57. Использование бесконечных чисел в выражениях

```
<?php
  echo 1/pow(0, -1);          // 0
  echo 123.24e306/123.24e308; // 0
  echo 123.24e308/123.24e306; // INF
?>
```

12.9. Вычисления с повышенной точностью

В данном разделе рассматриваются функции для вычислений с повышенной точностью. Дело в том, что точность обычных математических функций ограничивается архитектурой, что в свою очередь диктуется скоростью вычисления. В случаях, когда скорость не имеет значения, вычисления можно осуществлять с наперед заданной точностью. Однако следует помнить, что это может приводить к значительному увеличению времени вычисления, т. к. дробные числа будут храниться не в регистрах математического сопроцессора, а в обычных строках, которые в свою очередь будут расположены в более медленной оперативной памяти. В табл. 12.9 приводится список функций для вычислений с повышенной точностью, которые обеспечиваются библиотекой BCMath (по умолчанию библиотека встроена в ядро PHP).

ЗАМЕЧАНИЕ

Числа в функциях с повышенной точностью задаются в виде строк, результат также возвращается в виде строки.

Таблица 12.9. *Функции для вычислений с повышенной точностью*

Функция	Описание
bcadd($*left*, $*right* [, $*scale*])	Возвращает сумму двух чисел $*left* и $*right* с точностью, определяемой параметром $*scale* (количество знаков после точки)
bcsub($*left*, $*right* [, $*scale*])	Вычитает из $*left* число $*right* с точностью, определяемой параметром $*scale* (количество знаков после точки)
bcdiv($*left*, $*right* [, $*scale*])	Делит число $*left* на $*right* с точностью, определяемой параметром $*scale* (количество знаков после точки)
bcmul($*left*, $*right* [, $*scale*])	Возвращает произведение $*left* на $*right* с точностью, определяемой параметром $*scale* (количество знаков после точки)
bcpow($*left*, $*right* [, $*scale*])	Возвращает число $*left* в степени $*right* с точностью, определяемой параметром $*scale* (количество знаков после точки)

***Таблица 12.9** (окончание)*

Функция	Описание
`bcmod($left, $right)`	Возвращает остаток целочисленного деления числа `$left` на `$right`
`bcpowmod($left, $right, $modulus [, $scale])`	Функция аналогична выражению `bcmod(bcpow($x, $y), $mod)`
`bcsqrt($operand [, $scale])`	Вычисляет корень квадратный числа `$operand` с точностью, определяемой параметром `$scale` (количество знаков после точки)
`bccomp($left, $right [, $scale])`	Сравнивает числа `$left` и `$right` с точностью `$scale`; возвращает 0, если числа равны, 1, если `$left` больше `$right`, и –1 в противном случае
`bcscale($scale)`	Изменяет заданную по умолчанию точность вычислений на `$scale` для всех математических функций библиотеки BCMath. По умолчанию эта величина принимает нулевое значение

Функция `bcadd()` осуществляет сложение двух чисел с наперед заданной точностью. Функция имеет следующий синтаксис:

```
bcadd($left, $right [, $scale])
```

Возвращает сумму двух чисел `$left` и `$right` с точностью, определяемой параметром `$scale` (количество знаков после точки). В листинге 12.58 приводится пример использования функции `bcadd()`.

Листинг 12.58. Сложение чисел с повышенной точностью, функция `bcadd()`

```
<?php
  $a = "1.234";
  $b = "5";

  echo bcadd($a, $b);    // 6
  echo bcadd($a, $b, 4); // 6.2340
?>
```

Как видно из листинга 12.58, по умолчанию сложение осуществляется с точностью до нуля знаков после точки. Однако точность можно увеличить при помощи третьего параметра функции. Для того чтобы явно не указывать точность при каждой операции с функциями повышенной точности, можно задать точность по умолчанию при помощи функции `bcscale()` (листинг 12.59).

Листинг 12.59. Установка точности по умолчанию при помощи функции `bcscale()`

```
<?php
  $a = "1.234";
  $b = "5";

  // Устанавливаем точность
  bcscale (4);
  // Осуществляем сложение
  echo bcadd($a, $b); // 6.2340
?>
```



Глава 13

Файлы и каталоги

Web-приложения на PHP, работающие с различными данными, в основном используют две возможности для хранения информации: реляционные базы данных и локальную файловую систему. Файлы полезны для хранения простых (к примеру, информация о настройках) и различных неструктурированных данных (изображения, бинарные файлы и т. п.).

Файл представляет собой последовательность байтов, хранящуюся на каком-либо физическом носителе информации. Каждый файл имеет абсолютный путь, по которому определяется его местонахождение. В качестве разделителя пути в Windows может использоваться как прямой (/), так и обратный (\) слеш. В UNIX-подобных операционных системах используется только прямой слеш.

Каталогом называют файл специального вида, который хранит список других файлов и каталогов, входящих в его состав. Обрабатывать каталог при помощи файловых функций не получится — за этим следит операционная система. Как правило, для обработки каталогов предназначен специальный набор функций.

Помимо этих двух основных видов файлов многие файловые системы поддерживают жесткие и символические ссылки. *Жесткой ссылкой* называют дополнительную точку доступа к файлу. Дело в том, что файлы в файловой системе различаются не по названиям, а по уникальным номерам: связь названия файла и его номера называется жесткой ссылкой. Обычно такая ссылка одна, однако ничто не мешает создать их несколько, получив, таким образом, доступ к файлу из разных точек файловой системы. Файл не удаляется, пока не будет удалена последняя жесткая ссылка. *Символическая ссылка* (ее еще называют мягкой) представляет собой файл специального вида, указывающий на другой файл. Внешне она выглядит точно так же, как жесткая ссылка, однако при удалении файла, ссылка просто будет указывать на несуществующий файл.

> ***Замечание***
>
> Разные файловые системы могут поддерживать и другие виды файлов, однако Web-разработчики с ними, как правило, дело не имеют, ограничиваясь лишь вышеперечисленными видами фалов.

13.1. Создание файлов

PHP предоставляет разработчикам большое количество функций для работы с файлами. Первыми рассмотрим функции, позволяющие создать файлы (табл. 13.1).

Замечание

Функции `fopen()` и `tmpfile()` предназначены не столько для создания файла, сколько для его открытия, т. е. получения дескриптора, который позволяет осуществлять манипуляции с содержимым файла.

Таблица 13.1. *Функции создания файлов*

Функция	Описание
`fopen($filename, $mode [, $use_include_path [, $context]])`	Открывает файл с именем `$filename` в режиме `$mode`. Ряд режимов позволяет создать файл, ряд требует, чтобы файл уже существовал. Необязательный параметр `$use_include_path` позволяет задать путь для поиска файла. В случае успеха функция возвращает дескриптор открытого файла, в случае неудачи — `FALSE`. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу
`fclose($handle)`	Закрывает файл, открытый ранее при помощи функции `fopen()`. В качестве единственного параметра `$handler` функция принимает дескриптор открытого файла. Возвращает `TRUE` при успешном закрытии файла и `FALSE` в противном случае
`touch($filename [, $time [, $atime]])`	Функция `touch()` предназначена для изменения времени последней модификации файла `$time` и времени последнего доступа `$atime`. Если файл не существует, он создается, поэтому функция часто используется для создания файлов. Подробнее функция обсуждается в *разд. 13.5*
`tempnam($dir, $prefix)`	Создает в каталоге `$dir` файл с уникальным именем, при этом в имени файла используется префикс `$prefix`. В случае успеха возвращает имя нового файла, в случае неудачи — `FALSE`
`tmpfile()`	Создает временный файл с уникальным именем и возвращает его дескриптор. Файл автоматически удаляется после его закрытия. Если создать файл не удалось, функция возвращает `FALSE`

Один скрипт может оперировать множеством файлов. Чтобы их как-то отличать друг от друга, используется либо имя файла, либо его дескриптор. Имя файла может содержать полный путь, начиная от корня диска, такой путь называется *абсолютным*:

```
/var/home/www/htdocs/text.txt
C:\www\htdocs\text.txt
```

Замечание

В операционных системах Windows разделы обозначаются символами английского алфавита, буквы A и B традиционно резервируются под гибкие диски, остальные буквы используются для именования разделов жесткого диска. В UNIX-подобных операционных системах любой путь начинается с корневого раздела (root-раздела) `/`, к каталогам которого монтируются физические разделы. Таким образом, если в Windows может быть несколько корневых точек, обозначенных буквами, в UNIX-подобных операционных системах такая точка всегда одна.

В различных операционных системах используются разные разделители между каталогами и файлами. В UNIX-подобных операционных системах в качестве разделителя используется прямой слеш `/`, в Windows — обратный `\`. Обратный слеш традиционно служит для экранирования в строках *(см. разд. 3.6)*, поэтому его использование доставляет массу неудобств Windows-разработчикам. К сожалению, исправить ситуацию не представляется возможным — такая нотация используется свыше 20 лет во всех операционных системах компании Microsoft, начиная с DOS. В языке PHP эта проблема частично разрешена — при формировании путей к файлам допускается применять как прямой `/`, так и обратный слеши `\`. Однако при использовании обратного слеша следует помнить, что его необходимо удваивать в строках. В листинге 13.1 приводится пример формирования абсолютного Windows-пути, обе строки полностью эквивалентны.

Листинг 13.1. Использование прямого и обратного слеша при формировании пути

```
<?php
  $windows = "C:\\www\\htdocs\\text.txt";
  $php     = "C:/www/htdocs/text.txt";
?>
```

Как видно из листинга 13.1, удобнее оперировать прямыми слешами `/`, именно они и будут использоваться при дальнейшем изложении.

Помимо абсолютного пути различают *относительный* путь, который выстраивается относительно текущего каталога (как правило, каталога, в котором расположен скрипт).

ЗАМЕЧАНИЕ

Изменить текущий каталог можно при помощи функции `chdir()`, которая описывается в *разд. 13.8*.

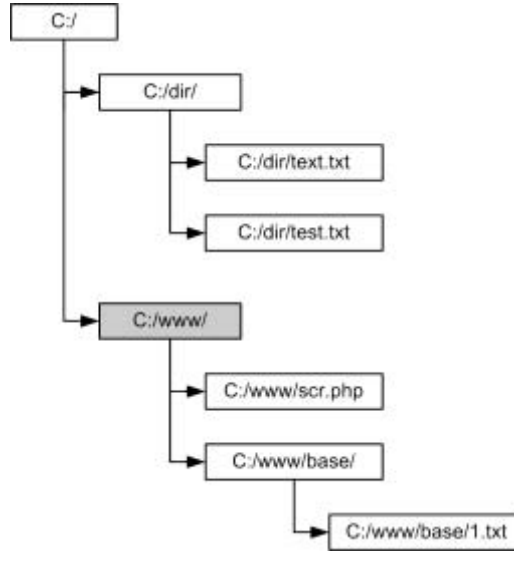


Рис. 13.1. Гипотетическая структура файловой системы

На рис. 13.1 представлена гипотетическая структура файловой системы, при этом текущий каталог C:/www/ выделен серым цветом. Для доступа к нижележащим каталогам достаточно указать путь относительно текущего каталога. В листинге 13.2 приводятся абсолютный и относительный пути к файлу C:/www/base/1.txt, который можно использовать в скрипте C:/www/scr.php.

Листинг 13.2. Абсолютный и относительный путь к файлу 1.txt

```
<?php
  // Абсолютный путь к файлу 1.txt
  $absolute = "C:/www/base/1.txt";
  // Относительный путь к файлу 1.txt
  $relative = "base/1.txt";
?>
```

Следует отметить, что относительный путь не предваряет ни буква раздела, ни символ корневого раздела / — он начинается либо с имени каталога, либо с имени файла.

Как видно из рис. 13.1, для того чтобы использовать относительный путь для доступа к файлу C:/dir/text.txt, необходимо подняться на один уровень выше и спуститься в каталог dir. Для формирования таких путей используется последовательность из двух точек . ., обозначающая родительский каталог (листинг 13.3).

> ***Замечание***
>
> Помимо родительского каталога, обозначаемого двумя точками (..), большинство файловых систем поддерживает текущий каталог, который обозначается одной точкой.

Листинг 13.3. Использование родительского каталога . .

```
<?php
  // Абсолютный путь к файлу text.txt
  $absolute = "C:/dir/text.txt";
  // Относительный путь к файлу text.txt
  $relative = "../dir/1.txt";
?>
```

Если необходимо подняться еще выше, можно использовать несколько последовательностей . ., например, ../../../result/text.txt.

Абсолютный и относительный пути позволяют оперировать именами файлов в пределах файловой системы. В Web также используется *сетевой* путь, который начинается с имени протокола, например, **http://www.softtime.ru/index.php**.

Теперь, когда правила формирования путей к файлам определены, создадим файл в текущем каталоге. Для создания пустого файла удобнее всего воспользоваться функцией `touch()` (листинг 13.4).

Листинг 13.4. Использование функции `touch()`

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Создаем пустой файл с именем $filename
  touch($filename);
?>
```

В случае успешного выполнения скрипта из листинга 13.4 в каталоге, где он расположен, появится новый файл text.txt. Функция `touch()` является типичной функцией, которая ориентируется на название файла. Однако при доступе к файлам более распространен дескрипторный вариант — имя файла связывается с дескриптором (целым числом, уникальным в пределах скрипта), который и используется в дальнейшем для доступа к этому файлу. Процесс связывания имени файла с дескриптором называется "открытием" файла и осуществляется при помощи функции `fopen()`, которая имеет следующий синтаксис:

```
fopen($filename, $mode [, $use_include_path [, $context]])
```

Функция открывает файл с абсолютным, относительным или сетевым путем `$filename` в режиме `$mode`. Если файл отсутствует в текущем каталоге, его поиск может продолжиться в каталогах, заданных в параметре `$use_include_path` (если этот параметр используется). При успешном выполнении функция возвращает дескриптор открытого файла, в противном случае возвращается `FALSE`.

В табл. 13.2 приводится описание режимов `$mode`, в которых может открываться файл. Функция `fopen()` позволяет создавать файлы, однако рассматривать это следует как частный случай, большинство режимов ориентировано на чтение и запись в файл при помощи функций, более подробно рассматривающихся в *разд. 13.3*.

Таблица 13.2. *Режимы открытия файла функцией* `fopen()`

Режим	Чтение	Запись	Файловый указатель	Очистка файла	Создать, если файла нет	Ошибка, если файл есть
`r`	Да	Нет	В начале	Нет	Нет	Нет
`r+`	Да	Да	В начале	Нет	Нет	Нет
`w`	Нет	Да	В начале	Да	Да	Нет
`w+`	Да	Да	В начале	Да	Да	Нет
`a`	Нет	Да	В конце	Нет	Да	Нет
`a+`	Да	Да	В конце	Нет	Да	Нет
`x`	Нет	Да	В начале	Нет	Да	Да
`x+`	Да	Да	В начале	Нет	Да	Да

Как видно из таблицы, для создания файлов подходят любые режимы, кроме `r` и `r+`. При попытке открыть несуществующий файл в одном из этих режимов функция `fopen()` возвращает `FALSE`, в случае остальных режимов функция пытается создать несуществующий файл. В листинге 13.5 приводится пример создания файла с именем text.txt при помощи функции `fopen()`.

Листинг 13.5. Создание файла при помощи функции `fopen()`

```
<?php
  // Имя файла
  $filename = "text.txt";
  // Получаем дескриптор открытого файла
  $fd = fopen($filename, "w");
  // Если файл успешно открыт, сообщаем об этом и закрываем его
  if (!$fd)
  {
    echo "Файл успешно создан<br>";
    fclose($fd);
  }
?>
```

Закрывать файл при помощи функции `fclose()` необязательно, т. к. все открытые дескрипторы закрываются автоматически после завершения работы скрипта. Однако лучше все-таки использовать функцию `fclose()`, особенно при записи информации в файл: информация записывается в файл не побайтово, а блоками по мере заполнения буфера записи. Содержимое буфера сбрасывается на диск либо при его заполнении, либо при использовании функции `fflush()` *(см. разд. 13.3)*, либо при закрытии файла функцией `fclose()`. Если работа скрипта будет неожиданно остановлена, информация из буфера, не сброшенная на жесткий диск, пропадет.

Часто файлы используются как временные хранилища информации, в этом случае удобно прибегать к созданию временных файлов. Функция `tempnam()` позволяет создать в каталоге файл с уникальным именем и имеет следующий синтаксис:

```
tempnam($dir, $prefix)
```

Функция создает файл с уникальным именем в каталоге `$dir` и с префиксом `$prefix`. В качестве результата функция `tempnam()` возвращает имя только что созданного файла. Если создать файл по каким-то причинам не удается, функция возвращает `FALSE`.

В листинге 13.6 скрипт создает в текущем каталоге новый файл с уникальным именем при каждом нажатии на кнопку.

Листинг 13.6. Создание файлов с уникальными именами

```
<form method='post'>
  <input type='submit' value='Записать'>
```

```
</form><br>
<?php
  // Обработчик HTML-формы
  if (isset($_POST))
  {
    // Создаем файл со случайным именем
    $filename = tempnam("./", "fl");
    // Открываем файл
    $fd = fopen($filename, "w");
    // Закрываем файл
    fclose($fd);
  }
?>
```

Помимо функции `tempnam()`, PHP предоставляет в распоряжение разработчика функцию `tmpfile()`, которая также создает временный файл с уникальным именем, но возвращает дескриптор открытого файла. После того как файл закрывается, файл автоматически удаляется. В отличие от функции `tmpfile()`, файл, созданный функцией `tempnam()`, остается существовать и после вызова функции.

13.2. Манипулирование файлами

Помимо операции создания файлов очень часто возникают задачи, связанные с их перемещением, переименованием и удалением. Функции, ответственные за эти операции, представлены в табл. 13.3.

***Таблица 13.3.** Функции манипуляции файлами*

Функция	Описание
`copy($source, $destination)`	Копирует файл с именем `$source` в файл с именем `$destination`. В случае успешного копирования функция возвращает `TRUE`, в противном случае возвращает `FALSE`
`unlink($filename)`	Удаляет файл с именем `$filename`. В случае успешного удаления функция возвращает `TRUE`, в противном случае возвращает `FALSE`
`rename($oldname, $newname)`	Переименовывает файл с именем `$oldname`, назначая ему новое имя `$newname`. В случае успешного переименования функция возвращает `TRUE`, в противном случае возвращает `FALSE`

В листинге 13.7 приводится пример использования функции `copy()`, которая копирует файл /etc/my.cnf в текущий каталог. Если на момент копирования файл my.cnf существовал в текущем каталоге, он будет перезаписан без предупреждений.

Листинг 13.7. Использование функции `copy()`

```
<?php
  if (copy("/etc/my.cnf", "my.cnf")) echo "Файл успешно скопирован";
  else echo "Не удалось скопировать файл";
?>
```

Если файл необходимо переместить, то его можно удалить из точки копирования при помощи функции `unlink()` (листинг 13.8).

Листинг 13.8. Перемещение файла

```
<?php
  // Копируем файл
  if (!copy("/etc/my.cnf", "my.cnf")) exit "Не удалось переместить файл";
  // Удаляем исходный файл
  if (unlink("/etc/my.cnf")) echo "Файл успешно перемещен";
  else echo "Не удалось переместить файл";
?>
```

Впрочем, операцию по переносу файла можно осуществить в один прием при помощи функции `rename()`, которая предназначена для переименования файлов, однако вполне может осуществлять их перенос — для этого достаточно указать новый путь, оставив имя файла без изменения (листинг 13.9).

Листинг 13.9. Использование функции `rename()`

```
<?php
  // Перенос файла
  if (rename("/etc/my.cnf", "my.cnf")) echo "Файл успешно перемещен";
  else echo "Не удалось переместить файл";
?>
```

Функции `copy()` и `rename()` могут копировать не только локальные файлы, но и сетевые. В листинге 13.10 демонстрируется копирование файла **http://www.softtime.ru/index.php**.

Листинг 13.10. Копирование файла из сети

```
<?php
  if (copy("http://www.softtime.ru/index.php", "index.htm"))
    echo "Файл успешно скопирован";
  else echo "Не удалось скопировать файл";
?>
```

Загруженный из сети файл не будет содержать исходных PHP-кодов, лишь HTML-представление (даже если скрипт и копируемый файл находятся на одном сервере). Это связано с тем, что обращение идет к Web-серверу, а не к файловой системе. По этой же причине при помощи функции `copy()` или любой другой функции невозможно загрузить файл на сервер.

Для загрузки файлов на сервер используются HTML-форма и элемент управления типа `file` *(см. разд. 8.10)*. Сервер помещает загруженный файл во временный каталог (путь к файлу во временном каталоге можно получить при помощи элемента `$_FILES['filename']['tmp_name']`), из которого скрипт может его скопировать в любое другое место. Для проверки факта загрузки файла на сервер и перемещения его предназначены специальные функции, описание которых представлено в табл. 13.4.

***Таблица 13.4.** Функции, обслуживающие загруженные на сервер файлы*

Функция	Описание
`is_uploaded_file` (`$`*`filename`*)	Возвращает `TRUE`, если файл был загружен на сервер, и `FALSE` в противном случае. В качестве аргумента функция принимает элемент массива `$_FILES['`*`filename`*`']['`*`tmp_name`*`']`, содержащий имя загруженного файла во временном каталоге
`move_uploaded_file` (`$`*`filename`*, `$`*`destination`*)	Перемещает файл из временного каталога в каталог назначения. В качестве первого аргумента `$`*`filename`* зачастую используется элемент `$_FILES['`*`filename`*`']['`*`tmp_name`*`']`, второй аргумент может быть произвольным. Если необходимо сохранить исходное имя файла, можно воспользоваться элементом `$_FILES['`*`filename`*`']['`*`name`*`']`. В отличие от функции `copy()` функция `move_uploaded_file()` оперирует лишь загруженными файлами

В листинге 13.11 приводится пример использования функций `is_uploaded_file()` и `move_uploaded_file()`.

Листинг 13.11. Использование функций `is_uploaded_file()` и `move_uploaded_file()`

```
<?php
  // Проверяем, загружен ли файл
  if (is_uploaded_file($_FILES['filename']['tmp_name']))
  {
    // Файл успешно загружен, перемещаем его в текущий каталог
    if (move_uploaded_file($_FILES['filename']['tmp_name'],
                           $_FILES['filename']['name']))
      echo "Файл успешно загружен<br>";
    else
      echo "Файл не удалось загрузить<br>";
```

```
  }
?>
<form action="upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="filename"><br>
<input type="submit" value="Загрузить"><br>
</form>
```

13.3. Чтение и запись файлов

Предыдущий раздел посвящен функциям, манипулирующим файлом как единым целым. Функции текущего раздела позволяют оперировать содержимым файла: читать, записывать и перезаписывать. В табл. 13.5 представлены функции, позволяющие осуществлять чтение и запись.

> ***ЗАМЕЧАНИЕ***
>
> Большинство функций из табл. 13.5 в качестве первого аргумента принимает файловый дескриптор `$handler`, который можно получить, "открыв" файл при помощи функции `fopen()`, порядок работы с которой более подробно рассматривается в *разд. 13.2*.

Таблица 13.5. *Функции чтения и записи файлов*

Функция	Описание
`fopen($filename, $mode [, $use_include_path [, $context]])`	Открывает файл с именем `$filename` в режиме `$mode`. Ряд режимов позволяет создать файл, ряд требует, чтобы файл уже существовал. Необязательный параметр `$use_include_path` позволяет задать путь для поиска файла. В случае успеха функция возвращает дескриптор открытого файла, в случае неудачи — `FALSE`. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу
`fclose($handle)`	Закрывает файл, открытый ранее при помощи функции `fopen()`. В качестве единственного параметра `$handle` функция принимает дескриптор открытого файла. Возвращает `TRUE` при успешном закрытии файла и `FALSE` в противном случае
`feof($handle)`	Возвращает `TRUE`, если файловый указатель `$handler` указывает на конец файла, и `FALSE` в противном случае. Более подробно функция описывается в *разд. 13.3.1*
`fgetc($handle)`	Считывает и возвращает в качестве результата из открытого файла `$handle` один символ. Более подробно функция описывается в *разд. 13.3.1*

***Таблица 13.5** (продолжение)*

Функция	Описание
`fgets($handle [, $length])`	Считывает и возвращает в качестве результата из открытого файла `$handle` строку. Чтение заканчивается по достижению строки длины `$handle` – 1 (по умолчанию `$handle` принимает значение 1000), по достижению символа перевода строки или символа конца файла. Более подробно функция описывается в *разд. 13.3.1*
`fgetss($handle [, $length [, $allowable_tags]])`	Аналогична функции `fgets()`, однако из прочитанной строки удаляются все HTML-теги. Необязательный параметр `$allowable_tags` может содержать допустимые HTML-теги, которые не должны отбрасываться
`fread($handle, $length)`	Считывает и возвращает в качестве результата из открытого файла `$handle` строку длиной `$length` символов. В отличие от функции `fgets()` переводы строк не прекращают чтение. Более подробно функция описывается в *разд. 13.3.1*
`fscanf($handle, $format [, ...])`	Аналогична функции форматного разбора строки `sscanf()`, рассмотренной в *разд. 9.13*, однако чтение и разбор строки по форматному правилу `$format` осуществляется не из строки, а из открытого файла `$handler`. Если в функцию переданы только два первых параметра, обработанные значения будут возвращены в виде массива. В противном случае, если были переданы необязательные аргументы, функция вернет количество присвоенных значений
`fpassthru($handle)`	Читает содержимое открытого файла `$handle` от текущей позиции до конца файла и выводит в окно браузера
`file_get_contents($filename [, $flags [, $context [, $offset [, $maxlen]]]])`	Читает файл с именем `$filename` в режиме `$flags` и возвращает его содержимое в виде строки. Необязательный параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу. Если это не требуется, вместо него можно передать `NULL`. Параметр `$offset` задает позицию, начиная с которой выполняется чтение, а `$maxlen` — количество символов, которые следует прочитать. Более подробно функция описывается в *разд. 13.3.1*
`file($filename [, $flags [, $context]])`	Читает файл с именем `$filename` и возвращает его содержимое в виде массива, каждый элемент которого соответствует отдельной строке. Необязательный параметр `$flags` позволяет задать режим чтения файла. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу. Более подробно функция описывается в *разд. 13.3.1*

Таблица 13.5 *(окончание)*

Функция	Описание
`readfile($filename [, $use_include_path [, $context]])`	Читает файл с именем `$filename` и выводит его содержимое в окно браузера. Необязательный параметр `$use_include_path` позволяет задать путь для поиска файла. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу
`fwrite($handle, $str [, $length])`	Записывает в открытый файл `$handle` содержимое строки `$str`. Необязательный параметр `$length` позволяет задать количество байтов, предназначенных для записи (по достижению этого количества запись останавливается). В случае успеха функция возвращает количество записанных байтов, в случае неудачи — `FALSE`. Более подробно функция описывается в *разд. 13.3.2*
`fputs()`	Синоним для функции `fwrite()`
`file_put_contents($filename, $data [, $flags [, $context]])`	Записывает в файл с именем `$filename` данные из параметра `$data` (либо строка, либо одномерный массив). Необязательный параметр `$flags` определяет режим записи файла. Параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу. Более подробно функция описывается в *разд. 13.3.2*
`fflush ($handle)`	Сбрасывает информацию, предназначенную для записи в файл из оперативной памяти на жесткий диск. Использование этой функции предотвращает потерю записанной в файл информации при нештатном завершении скрипта. Более подробно функция описывается в *разд. 13.3.3*
`flock($handle, $operation [, $wouldblock])`	Устанавливает для открытого файла с дескриптором `$handle` режим блокировки `$operation`. Необязательный параметр `$wouldblock` позволяет выяснить успешность операции. Более подробно функция описывается в *разд. 13.3.6*
`ftruncate($handle, $size)`	Уменьшает размер файла с дескриптором `$handle` до `$size` байт. Возвращает `TRUE` в случае успеха и `FALSE` в противном случае. Более подробно функция описывается в *разд. 13.3.5*

При открытии файла при помощи функции `fopen()` он может существовать или отсутствовать; существующий файл может содержать информацию или нет. Режимы открытия файла при помощи функции `fopen()` представлены в табл. 13.2.

Файл можно представить как последовательность байтов, по которой перемещается файловый указатель (рис. 13.2). Чтение и запись производится с текущего положения файлового указателя от начала к концу. Чтение и запись файлов в обратном порядке не допускается.



Рис. 13.2. Последовательное перемещение файлового указателя по файлу

При помощи режимов `r`, `w` и `x` файл открывается с установленным в начало файловым указателем. Это позволяет читать содержимое файла с начала, однако запись в файл также производится с начальной позиции, и все старое содержимое файла перезаписывается. Для того чтобы записать файл, не стирая его содержимое, файл следует открыть в режиме `a`, при котором файловый указатель помещается в конец файла. Несмотря на то, что режим `a+` допускает чтение, сразу после открытия файла прочитать ничего не удастся, т. к. файловый указатель находится в конце.

13.3.1. Чтение файлов

Функции чтения, читая порцию информации из файла, перемещают файловый указатель в конец прочитанного блока, поэтому, последовательно применяя их к одному и тому же файлу, можно просмотреть его содержимое от начала до конца.

В листинге 13.12 демонстрируется скрипт, открывающий текстовый файл text.txt, читающий из него символ за символом (при помощи функции `fgetc()`) и выводящий полученные символы в окно браузера.

Листинг 13.12. Чтение файла

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Последовательно читаем байты из файла, пока не достигнем конца файла
  while(($char = fgetc($fd)) !== FALSE)
  {
    echo $char;
  }
  // Закрываем файл
  fclose($fd);
?>
```

Как уже упоминалось ранее, оператор равенства = возвращает результат присвоения, поэтому цикл while() действует до тех пор, пока функция fgetc() не достигнет конца файла и не вернет значение FALSE. Сравнение следует осуществлять при помощи оператора неэквивалентности !==, т. к. оператор неравенства != остановит цикл, если будет встречен символ, который автоматически приводится к FALSE (например, 0).

Помимо способа остановки цикла, продемонстрированного в листинге 13.12, часто используется альтернативный вариант, основанный на явной проверке положения файлового указателя. Для такой проверки предназначена функция feof(), которая возвращает TRUE, если файловый указатель указывает на конец файла, и TRUE в противном случае. В листинге 13.13 приводится содержимое листинга 13.12, переписанное с применением функции feof().

Листинг 13.13. Использование функции `feof()`

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Последовательно читаем байты из файла, пока не достигнем конца файла
  while(!feof($fd))
  {
    $char = fgetc($fd);
    echo $char;
  }
  // Закрываем файл
  fclose($fd);
?>
```

При открытии файла при помощи функции fopen() следует обратить внимание на режим во втором параметре функции (см. табл. 13.2). Примеры из листингов 13.12 и 13.13 будут работать лишь в том случае, если открытие файла осуществляется в режимах r, r+, w+, a+ и x+, при попытке открыть файл в режимах w, a и x чтение из файла будет невозможным. Впрочем, режим x+ создает пустой несуществующий файл, и читать из него нечего, режим w+ стирает предыдущее содержимое, а a+ устанавливает файловый дескриптор в конец файла, и чтение невозможно. Поэтому при чтении из файла разумно открывать его в режимах r и r+.

К побайтовому чтению файлов прибегают достаточно редко. Так как при помощи PHP чаще решаются прикладные задачи, разработчики чаще работают с текстовыми файлами, которые удобнее читать построчно. Для этого используют функцию fgets(), которая имеет следующий синтаксис:

```
fgets($handle [, $length])
```

Функция считывает из открытого файла $handler строку. Чтение заканчивается по достижению строки длиной $handle – 1 (по умолчанию $handle принимает значение 1000), по достижению символа перевода строки или символа конца файла.

В случае успеха функция возвращает прочитанную строку, в случае неудачи — FALSE. В листинге 13.14 приводится пример использования функции fgets() для чтения текстового файла text.txt.

Листинг 13.14. Использование функции fgets()

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  while(!feof($fd))
  {
    $str = fgets($fd, 1024);
    echo "$str<br>";
  }
  // Закрываем файл
  fclose($fd);
?>
```

Какое бы большое значение не принимал второй параметр функции fgets(), чтение из файла будет прекращено при встрече первого перевода строки. Если такое поведение не удовлетворяет требованиям разработчика, можно воспользоваться функцией fread(), лишенной такой особенности. Функция имеет следующий синтаксис:

```
fread($handle, $length)
```

Функция fread() читает из открытого файла с дескриптором $handle блок информации длиной $length. В листинге 13.15 приводится пример использования функции fread().

Листинг 13.15. Использование функции fread()

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  while(!feof($fd))
  {
    $str = fread($fd, 1024);
    echo "$str<br>";
  }
  // Закрываем файл
  fclose($fd);
?>
```

Если заранее известен размер файла, то при помощи функции `fread()` можно прочитать содержимое файла за один раз (листинг 13.16).

Листинг 13.16. Чтение файла за один раз

```
<?php
  // Открываем файл на чтение
  $fd = fopen("text.txt", "r");
  if (!$fd) exit("Невозможно открыть файл text.txt");
  // Читаем содержимое файла за один раз
  $str = fread($fd, filesize($fd));
  echo $str;
  // Закрываем файл
  fclose($fd);
?>
```

Для определения размера файла в листинге 13.16 использовалась функция `filesize()`, которая более подробно рассматривается в *разд. 13.5*. Вместо точного размера файла в качестве второго параметра функции `fread()` можно подставить размер, заведомо превышающий размер файла. Однако следует проявлять аккуратность при загрузке сетевого файла — второй параметр определяет размер сетевого пакета, и при слишком большой его величине передача файла захлебывается. Наиболее оптимальным размером при загрузке файла через сеть считается размер 4096 байт (листинг. 13.17).

Листинг 13.17. Загрузка файла из сети

```
<?php
  // Открываем файл на чтение
  $fd = fopen("http://www.softtime.ru", "r");
  if (!$fd) exit("Невозможно открыть сетевой файл");
  // Построчно читаем данные из файла, пока не достигнем конца файла
  $str = "";
  while(!feof($fd))
  {
    $str .= fread($fd, 4096);
  }
  // Выводим полученную строку
  echo $str;
  // Закрываем файл
  fclose($fd);
?>
```

Листинги 13.16 и 13.17 демонстрируют типичную задачу — получение содержимого файла в переменную. Эта задача настолько распространена, что для ее решения в PHP

введена специализированная функция `file_get_contents()`, которая имеет следующий синтаксис:

```
file_get_contents($filename [, $flags [,
                  $context [, $offset [, $maxlen]]]])
```

Функция читает файл с именем `$filename` и возвращает его содержимое в виде строки. В отличие от большинства функций данного раздела, в качестве первого аргумента выступает путь к файлу, а не дескриптор, полученный при помощи функции `fopen()`.

Второй необязательный параметр `$flags` позволяет задать режимы работы функции. Допустимые значения этого параметра представлены в табл. 13.6. Можно указать несколько параметров, соединив их оператором `|`. Необязательный параметр `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу. Если это не требуется, вместо него можно передать `NULL`. Необязательные параметры `$offset` и `$maxlen` задают соответственно начальную позицию и количество символов, которые следует прочитать.

> ***Замечание***
>
> Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

Таблица 13.6. *Возможные значения параметра* `$flags` *функции* `file_get_contents()`

Значение	Описание
`FILE_USE_INCLUDE_PATH`	Поиск файла в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini
`FILE_TEXT`	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
`FILE_BINARY`	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла

В листинге 13.18 демонстрируется пример, полностью аналогичный по результату скрипту из листинга 13.16.

> ***Замечание***
>
> Функция `file_get_contents()`, как правило, используется для работы с небольшими файлами. Это связано с тем, что на один скрипт отводится ограниченный объем памяти — 8 или 16 Мбайт в зависимости от значения директивы `memory_limit` в конфигурационном файле php.ini. При попытке чтения файла объемом больше `memory_limit` работа скрипта будет остановлена и возвращена ошибка "Fatal error: Allowed memory size of 8388608 bytes exhausted" ("Критическая ошибка: израсходовано 8388608 байт зарезервированной памяти").

Листинг 13.18. Использование функции `file_get_contents()`

```
<?php
  $str = file_get_contents("text.txt");
  echo $str;
?>
```

Помимо локальных файлов, функция `file_get_contents()` способна к чтению сетевых файлов. В листинге 13.19 приводится альтернативная реализация скрипта из листинга 13.17.

Листинг 13.19. Чтение сетевых файлов при помощи функции `file_get_contents()`

```
<?php
  $str = file_get_contents("http://www.softtime.ru");
  echo $str;
?>
```

Если полученное при помощи функции `file_get_contents()` содержимое файла необходимо разбить построчно, можно воспользоваться функцией `explode()` (листинг 13.20).

Листинг 13.20. Получение содержимого файла в виде

```
<?php
  // Получаем содержимое файла
  $contents = file_get_contents("text.txt");
  // Разбиваем содержимое файла на отдельные строки
  $arr = explode("\n", $contents);
  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 13.20 будет массив `$arr`, каждый элемент которого будет соответствовать строке файла. Для получения содержимого файла в виде массива в PHP предусмотрена специальная функция `file()`, которая имеет следующий синтаксис:

`file($`*`filename`* `[, $`*`flags`* `[, $`*`context`*`]])`

Функция читает файл с именем `$`*`filename`* и возвращает его содержимое в виде массива, каждый элемент которого соответствует отдельной строке. Необязательный параметр `$`*`flags`* позволяет задать режим чтения файла и может принимать значения из табл. 13.7. Параметр `$`*`context`* позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу.

Замечание

Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

Таблица 13.7. *Возможные значения параметра* `$flags` *функции* `file()`

Значение	Описание
`FILE_USE_INCLUDE_PATH`	Поиск файла в каталогах, указанных в директиве `include_path` конфигурационного файла php.ini
`FILE_IGNORE_NEW_LINES`	Не помещать в конец каждого элемента результирующего массива пустой строки
`FILE_SKIP_EMPTY_LINES`	Игнорировать пустые строки
`FILE_TEXT`	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
`FILE_BINARY`	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла.

В листинге 3.21 приводится альтернативный вариант для скрипта из листинга 3.19.

Листинг 3.21. Использование функции `file()`

```
<?php
  // Получаем содержимое файла в виде массива
  $arr = file ("text.txt", FILE_IGNORE_NEW_LINES |
                           FILE_SKIP_EMPTY_LINES |
                           FILE_BINARY);
  // Выводим дамп массива $arr
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

13.3.2. Запись файлов

Для записи в файл его необходимо открыть (при помощи функции `fopen()`) в любом режиме, кроме `r`. Впрочем, для записи нового файла традиционно используются режимы `w` или `w+`, которые помещают файловый указатель в начало файла. Записывать информацию в файл можно при помощи функции `fwrite()`, которая имеет следующий синтаксис:

```
fwrite($handle, $str [, $length])
```

Функция записывает в открытый файл `$handle` содержимое строки `$str`. Необязательный параметр `$length` позволяет задать количество байтов, предназначенных для записи (по достижению этого количества запись останавливается). В случае успеха функция возвращает количество записанных байтов, в случае неудачи — `FALSE`. В листинге 13.22 демонстрируется скрипт, записывающий в файл text.txt строку "Hello, world!".

Листинг 13.22. Запись строки в файл

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "w");
  if (!$fd) exit("Невозможно открыть файл на запись");
  // Записываем строку "Hello, world!"
  fwrite($fd, "Hello, world!");
  // Закрываем файл
  fclose($fd);
?>
```

Помимо режимов, представленных в табл. 13.2, существуют дополнительные режимы `b` и `t` для бинарного и текстовых режимов открытия файлов. Эти режимы не являются самостоятельными и применяются только с одним из основных режимов, представленных в табл. 13.2.

Эти флаги введены из-за того, что в разных операционных системах используются различные обозначения конца строки. В UNIX-подобных операционных системах это последовательность `\n`, в Windows — `\r\n`, в Mac OS — `\n\r`. *Режим трансляции* `t` позволяет автоматически транслировать UNIX-перевод строки `\n` в перевод строки текущей операционной системы. *Бинарный режим* `b` отключает такое поведение и записывает файлы как есть.

> ***ЗАМЕЧАНИЕ***
>
> Большинство программ корректно обрабатывает любые виды переводом строки. Однако существуют исключения, например Блокнот в Windows признает только Windows-переводы строки `\r\n`, UNIX-перевод строки `\n` отображается как нечитаемый символ (квадратик).

В повседневной практике рекомендуется придерживаться бинарного режима `b`, т. к. режим трансляции зачастую приводит к искажению бинарных файлов и созданию приложений, не переносимых между операционными системами.

Различие между режимом трансляции и бинарным режимом удобно отслеживать в операционной системе Windows. Пусть имеется строка `"Hello\nworld!"`, содержащая UNIX-перевод строки `\n`. Запись такой строки в бинарном режиме (листинг 13.23) создаст файл длиной в 12 символов.

Листинг 13.23. Запись файла в бинарном режиме

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
  if (!$fd) exit("Невозможно открыть файл на запись");
```

```
  // Записываем строку в файл
  fwrite($fd, "Hello\nworld!");
  // Закрываем файл
  fclose($fd);
  // Подсчитываем количество символов в файле
  $str = file_get_contents("text.txt");
  echo strlen($str); // 12
?>
```

Запись строки `"Hello\nworld!"` в режиме трансляции создаст файл длиной 13 символов, т. к. UNIX-перевод строки `\n` будет преобразован в `\r\n` (листинг 13.24).

Листинг 13.24. Запись файла в режиме трансляции

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wt");
  if (!$fd) exit("Невозможно открыть файл на запись");
  // Записываем строку в файл
  fwrite($fd, "Hello\nworld!");
  // Закрываем файл
  fclose($fd);
  // Подсчитываем количество символов в файле
  $str = file_get_contents("text.txt");
  echo strlen($str); // 13
?>
```

Помимо записи файла при помощи файлового дескриптора, допускается перезапись всего содержимого файла посредством специализированной функции `file_put_contents()`, которая имеет следующий синтаксис:

```
file_put_contents($filename, $data [, $flags [, $context]])
```

Функция записывает в файл с именем *`$filename`* данные из параметра *`$data`* (либо строку, либо одномерный массив). Необязательный параметр *`$flags`* определяет режим записи файла и может принимать значения из табл. 13.8. Параметр *`$context`* позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу.

> ***Замечание***
>
> Константы `FILE_TEXT` и `FILE_BINARY` являются взаимоисключающими и не могут использоваться одновременно.

Таблица 13.8. *Возможные значения параметра* `$flags` *функции* `file_put_contents()`

Значение	Описание
FILE_USE_INCLUDE_PATH	Поиск файла в каталогах, указанных в директиве include_path конфигурационного файла php.ini
FILE_APPEND	Данные не перезаписывают содержимое файла, а добавляются в конец файла
LOCK_EX	Запрашивает исключительный доступ к файлу для записи информации, сторонние скрипты не смогут открыть файл для записи
FILE_TEXT	Данные возвращаются в кодировке UTF-8 (если этот режим включен в конфигурационном файле php.ini)
FILE_BINARY	Чтение осуществляется в бинарном режиме, без преобразования содержимого файла.

В листинге 13.25 демонстрируется альтернативный вариант скрипта из листинга 13.20.

Листинг 13.25. Использование функции `file_put_contents()`

```
<?php
  file_put_contents("text.txt", "Hello world!");
?>
```

В качестве второго аргумента функции допускается одномерный массив, элементы которого перед записью объединяются в строку.

Листинг 13.26. Запись одномерного массива в файл

```
<?php
  // Формируем массив
  $arr[] = "Hello ";
  $arr[] = "world!";
  // Записываем массив в файл
  file_put_contents("text.txt", $arr);
?>
```

Содержимое листинга 13.26 может быть переписано при помощи комбинации функций `file_put_contents()` и `implode()` (листинг 13.27). Такая избыточная конструкция очень часто встречалось в ранних скриптах на PHP, поэтому поведение функции `file_put_contents()` было расширено для поддержки массивов.

Листинг 13.27. Избыточный вариант записи одномерного массива в файл

```
<?php
  // Формируем массив
  $arr[] = "Hello ";
  $arr[] = "world!";
  // Записываем массив в файл
  file_put_contents("text.txt", implode("", $arr));
?>
```

13.3.3. Обязательно ли закрывать файлы?

Во всех предыдущих примерах, дескриптор, полученный при помощи функции `fopen()`, после использования закрывался при помощи функции `fclose()`. Это не обязательная процедура, т. к. все дескрипторы, открытые скриптом автоматически, закрываются после его штатного завершения. Тем не менее, рекомендуется явно закрывать открытые ранее дескрипторы. Такая рекомендация связана с особенностью файлового ввода/вывода.

При записи в файл с использованием функции `fwrite()` информация записывается не на жесткий диск, а в буфер, расположенный в оперативной памяти. По мере заполнения буфера информация сбрасывается на жесткий диск. Информация также сбрасывается на жесткий диск при явном закрытии файла с помощью функции `fclose()`. Такой механизм создан для ускорения операций ввода/вывода: если функция `fwrite()` будет дожидаться записи каждого байта на жесткий диск, файловые операции будут работать крайне медленно. Существующий механизм позволяет не дожидаться окончания операции записи и продолжать выполнение остальных операторов скрипта.

Недостатком при таком подходе является потеря информации, помещенной в буфер при нештатном завершении скрипта. Вероятность того, что часть информации не будет сброшена на жесткий диск, возрастает тем больше, чем дольше выполняется скрипт. Можно сильно уменьшить вероятность повреждения файла из-за того, что информация не была своевременно сброшена на диск, если своевременно закрыть его при помощи функции `fclose()`.

Удостовериться в таком поведении можно при помощи несложного скрипта, представленного в листинге 13.28. В скрипте открывается файл text.txt, в который записывается строка `"Hello, world!"`. После вызова функции `fwrite()` при помощи функции `sleep()` устанавливается задержка в один час. Если с момента запуска скрипта отслеживать размер файла text.txt, то можно заметить, что размер файла некоторое время остается равным 0 и лишь спустя небольшой промежуток времени изменяется.

Листинг 13.28. Задержка при записи информации на жесткий диск

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
```

```
  // Записываем строку в файл
  fwrite($fd, "Hello, world!");
  // Устанавливаем задержку на час
  sleep(3600);
  // Закрываем файл
  fclose($fd);
?>
```

Если поместить вызов функции `fclose()` до вызова функции `sleep()`, размер файла изменяется моментально, т. к. скрипт вынужден сбросить содержимое буфера ввода/вывода на диск.

Для гарантированной записи информации на жесткий диск файл не обязательно закрывать. Существует специализированная функция `fflush()`, которая заставляет сбрасывать буфер. Функция принимает в качестве единственного параметра дескриптор открытого файла:

```
fflush($handle)
```

В листинге 13.29 демонстрируется использование функции `fflush()`.

Листинг 13.29. Принудительная запись информации на жесткий диск

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
  // Записываем строку в файл
  fwrite($fd, "Hello world!");
  // Сбрасываем буфер перед длительной операцией
  fflush($fd);
  // Устанавливаем задержку на час
  sleep(3600);
  // Закрываем файл
  fclose($fd);
?>
```

13.3.4. Дозапись файлов

Приемы записи, продемонстрированные в *разд. 13.3.2*, перезаписывают содержимое файлов. Однако для ведения журналов, файловых баз данных и т. п. гораздо удобнее оставлять ранее полученные записи. Для этого вовсе не обязательно предварительно считывать содержимое файла, достаточно открыть файл при помощи функции `fopen()` в одном из режимов `a` или `a+`. В результате файловый указатель будет помещен в конец файла и записанная ранее информация не будет уничтожена. В листинге 13.30 демонстрируется скрипт, вызов которого дописывает в конец файла text.txt строку `"Hello, world!"`. Сколько раз будет вызван скрипт, столько строчек появится в файле.

Замечание

В конце "Hello, world!" добавлена последовательность перевода строки \r\n, чтобы содержимое файла было удобнее просматривать в текстовом редакторе.

Листинг 13.30. Дозапись в файл

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "a+b");
  // Записываем строку в файл
  fwrite($fd, "Hello, world!\r\n");
  // Закрываем файл
  fclose($fd);
?>
```

При помощи функции file_put_contents() можно осуществлять дозапись, не прибегая к получению с помощью функции fopen() дескриптора файла (листинг 13.31). Для этого достаточно передать в качестве третьего параметра константу FILE_APPEND (см. табл. 13.8).

Листинг 13.31. Дозапись в файл при помощи функции file_put_contents()

```
<?php
  file_put_contents("text.txt",
                    "Hello, world!\r\n",
                    FILE_APPEND | FILE_BINARY);
?>
```

13.3.5. Уменьшение размера файла

Для уменьшения размера открытого файла можно воспользоваться специальной функцией ftruncate(), которая имеет следующий синтаксис:

```
ftruncate($handle, $size)
```

Функция уменьшает размер файла с дескриптором $handle до $size байт. Возвращает TRUE в случае успеха и FALSE в противном случае. Пусть имеется файл text.txt, в котором содержится строчка

```
Hello, world!
```

В листинге 13.32 приводится скрипт, который при помощи функции ftruncate() удаляет из файла все, что находится за словом "Hello".

Листинг 13.32. Использование функции ftruncate()

```
<?php
  // Открываем файл для чтения
  $fd = fopen("text.txt", "r+b");
```

```
  if (!$fd) exit("Невозможно открыть файл");
  // Урезаем длину файла
  ftruncate($fd, 6);
  // Закрываем файл
  fclose($fd);
?>
```

Функция `ftruncate()` имеет ограничение — она удаляет лишь конечные символы файла. Для того чтобы удалить строки из начала файла, следует использовать комбинацию из строковых и файловых функций. В листинге 13.33 демонстрируется скрипт, который удаляет из файла text.txt первые 7 символов, оставляя лишь слово "world!".

Листинг 13.33. Удаление фрагмента из начала файла

```
<?php
  // Получаем конечные байты файла
  $content = substr(file_get_contents("text.txt"), 7);
  // Записываем новое содержимое файла
  file_put_contents("text.txt", $content);
?>
```

13.3.6. Блокировка файлов

Главной особенностью Web-программирования является большое количество одновременных обращений к приложению. Это может стать серьезной проблемой при записи информации в файл. Одновременная попытка дописать или переписать содержимое файла несколькими процессами может исказить информацию в нем или обнулить. Для того чтобы предотвратить искажение файла, прибегают к блокировке при помощи функции `flock()`, которая имеет следующий синтаксис:

```
flock($handle, $operation [, $wouldblock])
```

Функция устанавливает для открытого файла с дескриптором `$handle` режим блокировки `$operation`. Возможные режимы блокировки представлены в табл. 13.9. Обычно функция ожидает, пока будет получена блокировка, и лишь потом передает управление скрипту. Однако при использовании режима `LOCK_NB` управление передается незамедлительно в независимости от того, была получена блокировка или нет. Узнать результат выполнения функции в этом случае можно через необязательный параметр `$wouldblock`, который принимает значение `TRUE` в случае успешного получения блокировки и `FALSE` в противном случае.

> ***Замечание***
>
> Механизм блокировки работает лишь в том случае, если его поддерживают все процессы, стремящиеся получить доступ к файлу. Вызов функции `flock()` означает, что скрипт согласен добровольно подождать, когда файл освободится. Если при этом какой-то из процессов не вызывает функции `flock()`, никто не препятствует ему в доступе к файлу (который он может перезаписать или прочитать в обход всех блокировок).

Таблица 13.9. *Режимы блокировки функции* `flock()`

Режим	Описание
LOCK_EX	Исключительная блокировка, которую может удерживать лишь один процесс. Это режим, используемый, как правило, при записи в файл, гарантирует, что только текущий процесс имеет доступ к файлу
LOCK_SH	Разделяемая блокировка, которую может удерживать несколько процессов. Этот режим, используемый, как правило, при чтении, гарантирует, что в момент доступа к файлу читающих процессов в него не производится никаких записей
LOCK_UN	Снятие исключительной или разделяемой блокировки
LOCK_NB	Позволяет передать управление скрипту, не дожидаясь получения блокировки

Функцию `flock()` следует вызывать сразу после открытия файла для установки блокировки и непосредственно перед закрытием файла для снятия блокировки. В листинге 13.34 приводится пример получения исключительной блокировки для файла text.txt.

> ***Замечание***
>
> Исключительную блокировку следует применять лишь с дескрипторами `r+`, `a` или `a+`, т. к. режимы `w` и `w+` очищают файл и этот процесс происходит до момента получения блокировки, т. е. он может вмешаться в процесс записи в файл другого потока. При этом следует помнить, что в режимах `r+`, `a` и `a+` старые записи не удаляются, а в `r+` файловый дескриптор расположен в начале и перезапись идет по уже существующему тексту. Поэтому зачастую приходится прибегать к функциям, позволяющим напрямую манипулировать файловым дескриптором *(см. разд. 13.3.9)*.

Листинг 13.34. Получение исключительной блокировки

```
<?php
  // Имя файла
  $filename = "text.txt";

  // Открываем файл для записи
  $fd = fopen($filename, "r+b");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Получаем исключительную блокировку
  flock($fd, LOCK_EX);

  // Перезаписываем содержимое файла
  fwrite($fd, "Текст для записи в файл $filename");
  // Сбрасываем содержимого буфера на диск
  fflush($fd);
```

```
  // Снимаем блокировку с файла
  flock($fd, LOCK_UN);
  // Закрываем файл
  fclose($fd);
?>
```

Перед снятием исключительной блокировки необходимо в обязательном порядке вызвать функцию `fflush()`, сбрасывающую содержимое буфера записи на жесткий диск. В противном случае существует вероятность, что данные в файл будут записываться уже после снятия блокировки в момент закрытия файла функцией `fclose()`.

Функция `fclose()` также снимает блокировку, поэтому скрипт из листинга 13.34 можно переписать, исключив вызов функции `flock()`, снимающей блокировку, и функции `fflush()`, сбрасывающей данные из буфера на жесткий диск (листинг 13.35).

Листинг 13.35. Альтернативный порядок получения исключительной блокировки

```
<?php
  // Имя файла
  $filename = "text.txt";

  // Открываем файл для записи
  $fd = fopen($filename, "r+b");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Получаем исключительную блокировку
  flock($fd, LOCK_EX);

  // Перезаписываем содержимое файла
  fwrite($fd, "Текст для записи в файл $filename");

  // Закрываем файл
  fclose($fd);
?>
```

Исключительная блокировка гарантирует, что в момент записи в файл никакой другой процесс (также пытающийся получить исключительную блокировку) не сможет производить запись. Однако остается другая проблема: процессы, осуществляющие чтение из файла, могут прочитать файл в момент записи и получить его в искаженном виде. Для предотвращения такой ситуации предназначена разделяемая блокировка `LOCK_SH`. Такой вид блокировки гарантирует, что в момент чтения данных из файла данные в него записываться не будут. В листинге 13.36 приводится пример скрипта, блокирующего файл перед чтением.

Замечание

В отличие от исключительной блокировки, которую удерживает лишь один процесс, разделяемую блокировку могут удерживать несколько читающих процессов. Однако это не означает, что на посещаемом ресурсе не будет возможности реализовать исключительную блокировку из-за того, что файл постоянно удерживается разделяемой блокировкой — PHP заботится о предоставлении доступа для обоих видов блокировок.

Листинг 13.36. Получение разделяемой блокировки

```
<?php
  // Имя файла
  $filename = "text.txt";

  // Открываем файл для чтения
  $fd = fopen($filename, "r+b");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Получаем разделяемую блокировку
  flock($fd, LOCK_SH);

  // Читаем содержимое файла
  echo fread($fd, filesize($filename));

  // Снимаем блокировку с файла
  flock($fd, LOCK_UN);

  // Закрываем файл
  fclose($fd);
?>
```

В реальных Web-приложениях зачастую требуются более сложные схемы блокировки файлов. Если блокировку файла невозможно получить в настоящий момент, функция `flock()` будет ожидать такой возможности, не передавая управление скрипту. Для посетителя это будет выглядеть как "зависание" Web-приложения. Чтобы предотвратить это, прибегают к режиму `LOCK_NB` (см. табл. 13.9), который предписывает функции `flock()` немедленно возвращать управление скрипту, независимо от успешности или не успешности получения блокировки. Ориентироваться об исходе операции можно по третьему параметру функции, который принимает значение `TRUE`, если блокировка получена, и `FALSE` в противном случае. В листинге 13.37 приводится пример скрипта, который пытается получить блокировку в течение 10 секунд, причем если на третьей секунде блокировка так и не получена — выводится предупреждающая надпись о необходимости подождать.

Замечание

Режим `LOCK_NB` не поддерживается операционной системой Windows.

Листинг 13.37. Использование режима `LOCK_NB`

```
<?php
  // Имя файла
  $filename = "text.txt";

  // Открываем файл
  $fd = fopen($filename, "r+b");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Принимает значение TRUE, если получена блокировка
  $wouldblock = FALSE;

  // В течение 10 секунд пытаемся получить исключительный доступ к файлу
  for ($i = 0; $i < 10; $i++)
  {
    // Как только доступ получен, выходим из цикла
    flock($fd, LOCK_EX | LOCK_NB, $wouldblock);
    if ($wouldblock) break;
    // Задерживаем время выполнения программы на 1 секунду,
    // если доступ получить не удалось
    else
    {
      sleep(1);
      if ($i == 2) echo "Не удается получить блокировку...";
    }
  }

  if ($wouldblock)
  {
    // Записываем строку в файл
    fwrite($fd, "Текст для записи в файл $filename");
    // Очищаем буфер записи
    fflush($fd);
    // Снимаем блокировку с файла
    flock($fd, LOCK_UN);

    // Закрываем файл
    fclose($fd);
  }
?>
```

13.3.7. Файлы CSV

Файлы CSV (Comma Separated Value) — это текстовые файлы для представления электронных таблиц. Каждая строка файла соответствует одной строке таблицы, при этом значения полей таблицы отделяются друг от друга разделителями, например, точкой с запятой. CSV-файлы приобрели большую популярность у Web-разработчиков, т. к. файлы этого формата могут модифицироваться редактором электронных таблиц Excel и использоваться для формирования прайс-листов. В табл. 13.10 представлены функции, предназначенные для чтения и формирования CSV-файлов.

Таблица 13.10. *Функции для работы с CSV-файлами*

Функция	Описание
`fgetcsv($handle [, $length [, $delimiter [, $enclosure [, $escape]]]])`	Разбирает строку CSV-файла длиной `$length` (можно указать 0, чтобы функция самостоятельно оценила размер строки). Необязательный параметр `$delimiter` определяет разделитель поля (по умолчанию запятая). Параметр `$enclosure` определяет ограничитель поля (по умолчанию двойная кавычка). Параметр `$escape` определяет символ экранирования (по умолчанию обратный слеш \)
`fputcsv($handle, $fields [, $delimiter [, $enclosure]])`	Формирует строку CSV-файла из массива `$fields` и записывает ее в открытый файл `$handle`. Необязательный параметр `$delimiter` определяет разделитель поля (по умолчанию запятая). Параметр `$enclosure` определяет ограничитель поля (по умолчанию двойная кавычка)

В листинге 13.38 приводится пример скрипта, формирующего CSV-файл с таблицей умножения.

Листинг 13.38. Использование функции `fputcsv()`

```
<?php
  // Имя файла
  $filename = "text.csv";

  // Открываем файл для записи
  $fd = fopen($filename, "w+b");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Формируем электронную таблицу
  for($i = 1; $i <= 10; $i++)
  {
    $fields = array();
```

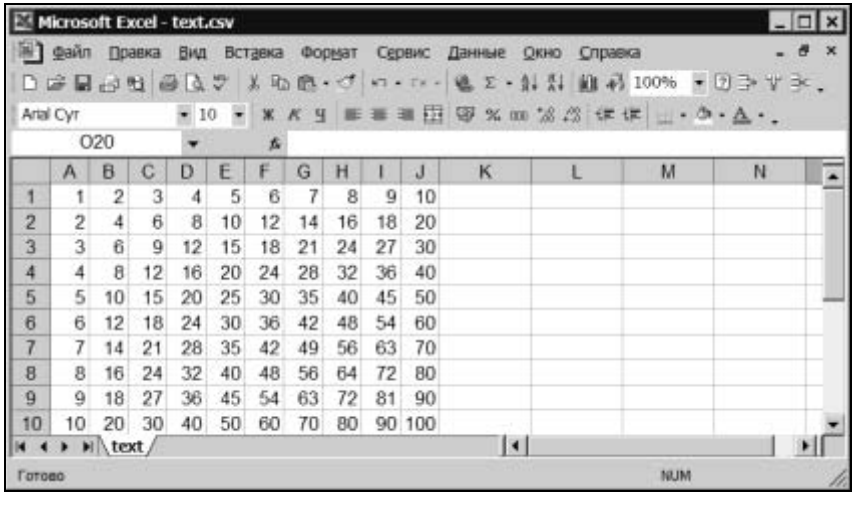
```
    for($j = 1; $j <= 10; $j++) $fields[] = $i * $j;
    fputcsv ($fd, $fields, ";");
  }

  // Закрываем файл
  fclose($fd);
?>
```

В результате в файл text.csv будет записаны следующие строки:

```
1;2;3;4;5;6;7;8;9;10
2;4;6;8;10;12;14;16;18;20
3;6;9;12;15;18;21;24;27;30
4;8;12;16;20;24;28;32;36;40
5;10;15;20;25;30;35;40;45;50
6;12;18;24;30;36;42;48;54;60
7;14;21;28;35;42;49;56;63;70
8;16;24;32;40;48;56;64;72;80
9;18;27;36;45;54;63;72;81;90
10;20;30;40;50;60;70;80;90;100
```

Если теперь попытаться открыть CSV-файл в Excel, результат может выглядеть так, как это представлено на рис. 13.3.



	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Рис. 13.3. Сгенерированный CSV-файл открыт в Excel

При помощи функции `fgetcsv()` можно разбирать содержимое CSV-файла. В листинге 13.39 представлен скрипт, формирующий из ранее полученного файла text.csv двумерный массив.

Листинг 13.39. Чтение и разбор CSV-файла

```
<?php
  // Имя файла
  $filename = "text.csv";

  // Открываем файл для чтения
  $fd = fopen($filename, "rb");
  if (!$fd) exit("Невозможно открыть файл $filename");

  // Читаем содержимое файла
  $result = array();
  while($arr = fgetcsv($fd, 0, ";")) $result[] = $arr;

  // Закрываем файл
  fclose($fd);

  // Выводим дамп результирующего массива
  echo "<pre>";
  print_r($result);
  echo "</pre>";
?>
```

13.3.8. Чтение и разбор ini-файла

Издавна для управления программами использовались параметры командной строки (это позволяло не перекомпилировать программы всякий раз, когда требовалось изменить режим их работы). Однако современные графические интерфейсы, где программы запускаются двойным щелчком мыши, не оставляют места для ввода параметров. В связи с этим повсеместно получили распространение конфигурационные ini-файлы, позволяющие настроить работу программы. Такие файлы содержат, как правило, пары "ключ=значение", секции, заключенные в квадратные скобки, и комментарии, начинающиеся с символов диеза (`#`) или точки с запятой (`;`). Для разбора таких файлов в PHP предусмотрена отдельная функция `parse_ini_file()`, которая имеет следующий синтаксис:

```
parse_ini_file($filename [, $process_sections])
```

Функция возвращает содержимое файла `$filename` в виде массива. Причем в качестве ключей выступают имена директив. Если необязательный параметр `$process_sections` принимает значение `FALSE`, результирующий массив является одномерным, если `TRUE` — двумерным (еще одно измерение отводится под секции).

> ***Замечание***
>
> Следует отметить, что PHP тоже имеет конфигурационный файл php.ini, позволяющий настроить разнообразные параметры ядра PHP и его расширений.

В качестве примера рассмотрим конфигурационный файл СУБД MySQL my.ini (листинг 13.40), который обычно содержит как минимум две секции:

- `[client]` — определяет директивы для клиентских утилит, таких как mysql, mysqldump, PHP-скрипты;
- `[mysqld]` — определяет директивы для MySQL-сервера.

> ***ЗАМЕЧАНИЕ***
>
> СУБД MySQL широко используется в Web-программировании. Более подробно порядок работы с ней освещается в *главе 27*.

Листинг 13.40. Типичный конфигурационный файл my.ini

```
# CLIENT SECTION
[client]

port=3306

# SERVER SECTION
[mysqld]

port=3306
basedir="C:/mysql5/"
datadir="C:/mysql5/Data/"
default-character-set=cp1251
default-storage-engine=MYISAM
max_connections=100
query_cache_size=0
table_cache=256
tmp_table_size=5M
thread_cache_size=8
```

В листинге 13.41 демонстрируется использование функции `parse_ini_file()`.

Листинг 13.41. Использование функции `parse_ini_file()`

```
<?php
  // Разбираем INI-файл
  $ini = parse_ini_file("my.ini");

  // Выводим дамп массива
  echo "<pre>";
  print_r($ini);
  echo "</pre>";
?>
```

В качестве результата скрипт из листинга 13.41 выводит следующий дамп массива `$ini`:

```
Array
(
    [port] => 3306
    [default-character-set] => cp1251
    [basedir] => C:/mysql5/
    [datadir] => C:/mysql5/Data/
    [default-storage-engine] => MYISAM
    [max_connections] => 100
    [query_cache_size] => 0
    [table_cache] => 256
    [tmp_table_size] => 5M
    [thread_cache_size] => 8
)
```

Следует отметить, что директивы `port=3306` из секций `[client]` и `[mysqld]` были объединены. Такое поведение не всегда является желательным, поэтому, передав в качестве второго параметра функции значение `TRUE`, можно получить двумерный массив, данные в котором разбиты по секциям:

```
Array
(
    [client] => Array
        (
            [port] => 3306
        )

    [mysqld] => Array
        (
            [port] => 3306
            [basedir] => C:/mysql5/
            [datadir] => C:/mysql5/Data/
            [default-character-set] => cp1251
            [default-storage-engine] => MYISAM
            [max_connections] => 100
            [query_cache_size] => 0
            [table_cache] => 256
            [tmp_table_size] => 5M
            [thread_cache_size] => 8
        )
)
```

13.3.9. Прямое манипулирование файловым указателем

Запись или чтение внутри файла начинается с точки, определяемой файловым указателем. При открытии файла он может быть установлен в начало (если требуется чтение файла или его перезапись) или в конец (если требуется дозапись файла). При чтении или записи блока информации, файловый указатель автоматически смещается в конец блока. То есть все рассмотренные выше функции оперировали файловым указателем неявно. Это не всегда удобно, поэтому PHP предоставляет разработчикам функции для явного управления положением файлового указателя (табл. 13.11).

Таблица 13.11. *Функции прямого манипулирования файловым указателем*

Функция	Описание
`ftell($handle)`	Возвращает текущее положение файлового указателя в открытом файле `$handle`
`fseek($handle, $offset [, $whence])`	Для открытого файла `$handle` устанавливает файловый указатель в новое положение, смещая его на `$offset` байт относительно позиции `$whence`. В случае успеха возвращает 0, в случае неудачи — –1
`rewind($handle)`	Устанавливает файловый указатель в начало файла `$handle`

Пусть для удобства существует файл text.txt, в котором записана лишь одна строчка:

```
Hello world!
```

Открыв такой файл при помощи функции `fopen()` в режиме чтения/записи `r+`, при помощи функции `ftell()` можно убедиться (листинг 13.42), что файловый указатель находится в начале файла (функция возвращает 0).

Листинг 13.42. Использование функции `ftell()`

```
<?php
  // Открываем файл
  $fd = fopen("text.txt", "r+b");
  if (!$fd) exit("Невозможно открыть файл");

  // Получаем текущую позицию файлового указателя
  echo ftell($fd)."<br>";
  // Записываем в файл строку
  fwrite($fd, "Bye  ");
  // Получаем текущую позицию файлового указателя
  echo ftell($fd)."<br>";

  // Закрываем файл
  fclose($fd);
?>
```

В качестве результата скрипт из листинга 13.42 выведет два положение файлового указателя: начальное и конечное.

```
0
5
```

При этом файл будет содержать подстроку "Bye world!". Таким образом, при открытии файла в режиме r+b, файловый указатель устанавливается в начале, поэтому функция fwrite() перезаписывается строкой "Bye " первые пять символов файла, устанавливая файловый указатель на пятый байт строки. Если бы за первым вызовом функции fwrite() последовал бы еще один, новая строка переписала бы слово "world!".

Часто при чтении или записи файловый указатель оказывается в конце файла. Для того чтобы осуществить повторное чтение, требуется переместить его в начало. Этого эффекта можно добиться, закрыв файл и открыв его по новой. Однако проще всего возвратить файловый указатель в начало файла при помощи функции rewind(). Функция принимает в качестве единственного параметра дескриптор открытого файла. В листинге 13.43 приводится пример скрипта, который выводит из текстового файла три последние строки. Неизвестно, сколько байтов содержится в одной строке, поэтому необходимо их предварительно посчитать (в результате файловый указатель оказывается в конце), а потом передвинуть файловый указатель в начало файла, чтобы повторно отобрать только последние три строки.

Листинг 13.43. Использование функции rewind()

```
<?php
  // Открываем файл
  $fd = fopen("text.txt", "r+b");
  if (!$fd) exit("Невозможно открыть файл");

  // Подсчитываем количество строк
  $count = 0;
  while (!feof($fd))
  {
    fgets($fd, 10000);
    $count++;
  }

  // Перемещаем файловый указатель в начало файла
  rewind($fd);

  // Повторно читаем файл
  for ($i = 0; $i < $count; $i++)
  {
    $str = fgets($fd, 10000);
```

```
    if ($i > $count - 4) echo $str."<br>";
  }

  // Закрываем файл
  fclose($fd);
?>
```

Файловый указатель можно установить не только в начало файла, но и в произвольную его точку. Для этого можно использовать функцию `fseek()`, которая имеет следующий синтаксис:

```
fseek($handle, $offset [, $whence])
```

Функция устанавливает файловый указатель в новое положение, смещая его на `$offset` байт относительно позиции `$whence`. Параметр `$whence` может принимать значения из табл. 13.12. В случае успеха возвращает 0, в случае неудачи — –1.

***Таблица 13.12.** Возможные значения параметра `$whence` функции `fseek()`*

Значение	Описание
SEEK_SET	Смещение отсчитывается от начала файла
SEEK_CUR	Смещение отсчитывается от текущей позиции файлового указателя
SEEK_END	Смещение отсчитывается от конца файла

В листинге 13.44 приводится пример использования функции `fseek()` — на только что записанную в файл строку накладывается дополнительная строка.

Листинг 13.44. Использование функции `fseek()`

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
  if (!$fd) exit("Невозможно открыть файл");

  // Помещаем строку в файл
  fwrite($fd, "Hello world!");

  // Помещаем файловый указатель на 4 позиции от начала файла
  fseek($fd, 4, SEEK_SET);

  // Записываем новую строку
  fwrite($fd, "4444"); // Hell4444rld!

  // Закрываем файл
  fclose($fd);
?>
```

Замечание

Функция `fseek()` может установить файловый указатель на любую точку файла, в том числе и за конец файла (при этом признак конца файла стирается), однако попытка установить файловый указатель на позицию перед началом файла приводит к ошибке.

В листинге 13.45 демонстрируется скрипт, позволяющий создать файл из 10 000 символов за счет перемещения файлового указателя за конец файла.

Листинг 13.45. Создание файла размером в 10 000 байт

```
<?php
  // Открываем файл для записи
  $fd = fopen("text.txt", "wb");
  if (!$fd) exit("Невозможно открыть файл");

  // Помещаем файловый указатель на 10 000 – 1 байт от начала файла
  fseek($fd, 9999, SEEK_END);

  // Записываем пробел
  fwrite($fd, " ");

  // Закрываем файл
  fclose($fd);
?>
```

13.4. Создание и работа со ссылками

В начале главы упоминалось, что файловые системы поддерживают жесткие ссылки (дополнительное имя для файла) и символические ссылки (перенаправляющий файл). В табл. 13.13 приводятся функции для работы со ссылками средствами PHP.

Замечание

Функции данной группы не реализованы для операционной системы Windows.

Таблица 13.13. *Функции для работы со ссылками*

Функция	Описание
`link($target, $link)`	Создает жесткую ссылку с именем `$link` для файла `$target`
`symlink($target, $link)`	Создает символическую ссылку с именем `$link` для файла `$target`
`readlink($link)`	Возвращает путь к файлу, на который указывает символическая ссылка `$link`
`linkinfo($link)`	Возвращает `TRUE`, если ссылка `$link` существует, и `FALSE` в противном случае

В листинге 13.46 приводится пример скрипта, создающего для файла text.txt жесткую ссылку another.txt.

Листинг 13.46. Использование функции `link()`

```
<?php
  // Создаем жесткую ссылку another.txt для файла text.txt
  if (!link("text.txt", "another.txt")) exit("Невозможно создать ссылку");
?>
```

Выполнив скрипт из листинга 13.46, можно убедиться, что помимо файла text.txt в каталоге создается файл another.txt. Причем изменения, внесенные в один файл, тут же отображаются в другом файле — на самом деле это один и тот же файл, только с двумя точками входа. Файл будет существовать до тех пор, пока существует хотя бы одна точка входа.

Функция `symlink()` позволяет создавать символические ссылки (листинг 13.47).

> ***Замечание***
>
> Путь, на который указывает символическая ссылка, может быть как относительным, так и абсолютным. Ссылки можно создавать только на локальные файлы, т. е. использовать сетевой путь не получится.

Листинг 13.47. Использование функции `symlink()`

```
<?php
  // Создаем символическую ссылку link.txt для файла text.txt
  if (!symlink("text.txt", "link.txt")) exit("Невозможно создать ссылку");
?>
```

Внешне символическая ссылка ничем не отличается от жесткой, однако, если будет удален исходный файл text.txt, открыть файл при помощи символической ссылки будет уже невозможно. То есть символическая ссылка — это файл, указывающий на другой файл, в то время как жесткая ссылка является прямым входом в файл (рис. 13.4).

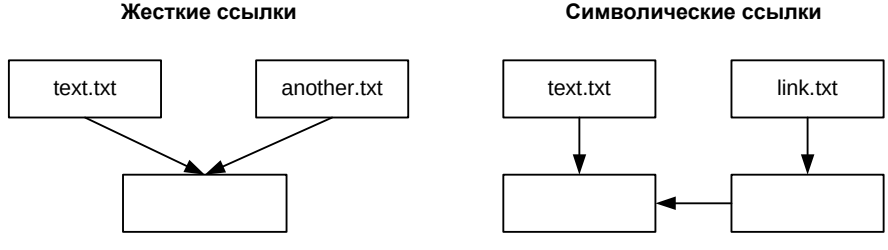


Рис. 13.4. Различие между жесткими и символическими ссылками

Разница между символической и жесткой ссылками очень хорошо проявляется при выполнении системной команды `ls -l`, выводящей список файлов в каталоге. Если жесткая ссылка another.txt выглядит как обычный файл, то для символической ссылки, помимо ее названия, выводится путь, на который она указывает: `link.txt -> text.txt`. Для символической ссылки получить этот путь можно при помощи функции `readlink()`, пример использования которой демонстрируется в листинге 13.48.

Листинг 13.48. Использование функции `readlink()`

```
<?php
  echo readlink("link.txt"); // text.txt
?>
```

13.5. Атрибуты файла

Функции данного раздела позволяют получить метаинформацию, такую как время последней модификации файла, его тип или размер. Список функций, позволяющих получать разнообразные атрибуты файла, представлен в табл. 13.14.

Таблица 13.14. *Функции извлечения атрибутов файлов*

Функция	Описание
`fileatime($filename)`	Возвращает время последнего обращения (чтение или запись) к файлу `$filename`
`filectime($filename)`	Возвращает время последнего изменения атрибутов файла `$filename`
`filemtime($filename)`	Возвращает время последнего изменения файла `$filename` или `FALSE`, если файл отсутствует
`touch($filename [, $time [, $atime]])`	Устанавливает для файла `$filename` время последней модификации `$time` и время последнего доступа `$atime`. Если файл не существует — он создается
`fileinode($filename)`	Возвращает номер блока, соответствующего файлу `$filename` в файловой системе
`filesize($filename)`	Возвращает размер файла `$filename` в байтах
`filetype($filename)`	Возвращает тип файла `$filename`
`file_exists($filename)`	Возвращает `TRUE`, если файл `$filename` существует, и `FALSE` в противном случае
`linkinfo($link)`	Возвращает `TRUE`, если ссылка `$link` существует, и `FALSE` в противном случае
`is_dir($filename)`	Возвращает `TRUE`, если `$filename` является каталогом, и `FALSE` в противном случае
`is_file($filename)`	Возвращает `TRUE`, если `$filename` является обычным файлом, и `FALSE` в противном случае

Таблица 13.14 *(окончание)*

Функция	Описание
`is_link($filename)`	Возвращает `TRUE`, если `$filename` является символической ссылкой, и `FALSE` в противном случае. Для жесткой ссылки функция возвращает `FALSE`, т. к. она рассматривается как обычный файл
`stat($filename)`	Возвращает массив с метаинформацией о файле `$filename`
`fstat($handle)`	Функция `fstat()` аналогична функции `stat()`, только в качестве параметра вместо имени файла принимает файловый дескриптор `$handle`, ранее полученный при помощи функции `fopen()`
`lstat($filename)`	Функция `lstat()` аналогична функции `stat()`. За исключением того, что если `$filename` является символической ссылкой, возвращается информация о файле-ссылке, а не о файле, на который эта ссылка указывает
`clearstatcache()`	Информационные функции из *разд. 13.5* и *13.6* кэшируют результаты. Это означает, что изменения не сразу находят отражение. Функция `clearstatcache()` позволяет очистить кэш и дает уверенность в том, что информационные функции оперируют самыми свежими данными

Для того чтобы узнать время последнего изменения файла, удобно воспользоваться функцией `filemtime()` (листинг 13.49). Время возвращается в формате UNIXSTAMP, т. е. количестве секунд, прошедших с полуночи 1 января 1970 года, поэтому перед выводом его следует отформатировать при помощи функции `date()` *(см. главу 11)*.

Листинг 13.49. Использование функции `filemtime()`

```
<?php
  $time = filemtime("text.txt");
  echo "Время последнего изменения - ".date("d.m.Y H:i", $time);
?>
```

Помимо функции `filemtime()`, PHP предоставляет разработчикам функцию `fileatime()`, которая возвращает время последнего доступа к файлу, будь то чтение или запись (листинг 13.50).

Листинг 13.50. Использование функции `fileatime()`

```
<?php
  $time = fileatime("text.txt");
  echo "Время последнего доступа - ".date("d.m.Y H:i", $time);
?>
```

Можно выставить произвольное время последнего изменения и доступа при помощи функции `touch()`, которая имеет следующий синтаксис:

```
touch($filename [, $time [, $atime]])
```

Функция устанавливает для файла `$filename` время последней модификации `$time` и время последнего доступа `$atime`.

> ***Замечание***
>
> В *разд. 13.1* для создания файла использовался побочный эффект функции `touch()`: если файл `$filename` не существует — он создается.

В листинге 13.51 для файла text.txt выставляются годичной давности времена последнего изменения и доступа.

Листинг 13.51. Использование функции `touch()`

```
<?php
  // Формируем время годичной давности
  $lastyear = mktime(date('H'),
                     date('m'),
                     date('i'),
                     date('m'),
                     date('d'),
                     date('Y') - 1);
  // Выставляем файлу времена последнего изменения и последнего доступа
  touch("text.txt", $lastyear, $lastyear);
?>
```

Ранее уже упоминалось, что файловые системы назначают файлам уникальные номера, с которыми потом сопоставляется имя файла. Получить номер, присвоенный файлу, можно при помощи функции `fileinode()` (листинг 13.52).

> ***Замечание***
>
> Функция поддерживается не всеми файловыми системами.

Листинг 13.52. Использование функции `fileinode()`

```
<?php
  echo fileinode("text.txt");
?>
```

К сожалению, функции, возвращающей имя файла по его уникальному номеру, не существует. В UNIX-подобных операционных системах для поиска файла по его уникальному номеру можно воспользоваться следующей командой:

```
find / -inum 171895 -print
```

Где 171895 — уникальный номер файла, возвращенный ранее функцией `fileinode()`. Результат поиска может вернуть несколько позиций, если для файла создано несколько жестких ссылок.

Определить размер файла можно при помощи функции `filesize()`, которая возвращает результат в байтах (листинг 13.53).

> ***Замечание***
>
> Функция `filesize()` часто используется совместно с функцией `fwrite()`, когда требуется прочитать файл целиком *(см. разд. 13.3.1)*.

Листинг 13.53. Использование функции `filesize()`

```
<?php
  echo filesize("text.txt");
?>
```

В повседневной практике не всегда удобно оперировать байтами, поэтому для функции `filesize()` часто создают функцию-обертку, автоматически преобразующую байты в килобайты, мегабайты и гигабайты в зависимости от величины файла (листинг 13.54).

Листинг 13.54. Форматирование размера файла

```
function getfilesize($filename)
{
  // Проверяем, существует ли файл
  if (!file_exists($filename)) return "файл не существует";
  // Определяем размер файла
  $filesize = filesize($filename);
  // Если размер файл превышает 1024 байта, пересчитываем размер в килобайты
  if ($filesize > 1024)
  {
    $filesize = (float)($filesize/1024);
    // Если размер файл превышает 1024 Кбайт, пересчитываем размер в мегабайты
    if ($filesize > 1024)
    {
      $filesize = (float)($filesize/1024);
      // Округляем дробную часть до первого знака после запятой
      $filesize = round($filesize, 1);
      return $filesize." Мб";
    }
    else
    {
      // Округляем дробную часть до первого знака после запятой
```

```
      $filesize = round($filesize, 1);
      return $filesize." Кб";
    }
  }
  else
  {
    return $filesize." байт";
  }
}
?>
```

Как видно из листинга 13.54, перед тем как перейти к вычислениям, при помощи функции `file_exists()` была осуществлена проверка файла. Функция возвращает `TRUE`, если файл или каталог существует, и `FALSE` в противном случае.

Иногда бывает важно определить, является ли файл обычным, ссылкой или каталогом, для этого используются соответственно функции `is_file()`, `is_link()` и `is_dir()` (листинг 13.55).

Листинг 13.55. Использование функций `is_file()`, `is_link()` и `is_dir()`

```
<?php
  // Путь к файлу
  $filename = "/path/to/file";
  // Проверяем файл на существование
  if (!file_exists($filename))
  {
    return "файл $filename не существует";
  }
  // Проверяем тип файла
  if (is_file($filename))
  {
    echo "$filename - файл";
  }
  else if (is_dir($filename))
  {
    echo "$filename - каталог";
  }
  else if (is_link($filename))
  {
    echo "$filename - символическая ссылка";
  }
  else
```

```
  {
    echo "$filename - что-то непонятное";
  }
?>
```

Использовать отдельные функции для определения типа файла не всегда удобно, иногда удобнее воспользоваться функцией `filetype()`, которая возвращает одно из значений, приведенных в табл. 13.15.

***Таблица 13.15.** Значения, возвращаемые функцией `filetype()`*

Функция	Описание
`file`	Обычный файл
`dir`	Каталог
`link`	Символическая ссылка
`char`	Специальный символьный файл
`block`	Специальный блочный файл
`fifo`	FIFO-канал
`unknown`	Файл неизвестного формата

В листинге 13.56 демонстрируется пример использования функции `filetype()`.

Листинг 13.56. Использование функции `filetype()`

```
<?php
  // Определяем тип файла
  $type = filetype("text.txt");
  // Выводим результат
  if ($type == "file") echo "text.txt - обычный файл";
  else if ($type == "dir") echo "text.txt - каталог";
  else echo "text.txt - специальный файл";
?>
```

В заключение рассмотрим функцию `stat()`, которая позволяет получить все рассмотренные атрибуты в виде массива (листинг 13.57).

Листинг 13.57. Использование функции `stat()`

```
<?php
  $arr = stat("text.txt");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Функция `stat()` возвращает смешанный массив, состоящий из ассоциативных и дублирующих их индексных значений, описание которых приводится в табл. 13.16.

Таблица 13.16. *Элементы результирующего массива функции* `stat()`

Индекс	Ключ	Описание
0	`dev`	Номер устройства (если файл ассоциирован с ним)
1	`ino`	Уникальный номер файла в файловой системе
2	`mode`	Режим защиты файла
3	`nlink`	Количество жестких ссылок на файл
4	`uid`	Уникальный номер владельца файла
5	`gid`	Уникальный номер группы-владельца файла
6	`rdev`	Тип устройства (если файл ассоциирован с ним)
7	`size`	Размер файла в байтах
8	`atime`	Время последнего доступа к файлу (чтение или запись)
9	`mtime`	Время последнего изменения содержимого файла
10	`ctime`	Время последнего изменения атрибутов файла
11	`blksiz`	Размер блока
12	`bloks`	Количество блоков, занятых файлом

13.6. Права доступа

Большинство серверов в Интернете работает под управлением UNIX-подобных операционных систем, которые имеют устоявшуюся систему прав доступа. Права доступа проектировались в расчете на многопользовательскую операционную систему.

Права доступа на каталоги и файлы могут быть заданы для трех категорий пользователей:

- владельца файла;
- группы владельца (все пользователи, входящие в эту группу);
- всех остальных.

Права доступа могут быть на чтение (4), на запись (2) и на исполнение (1) файла. Для разрешения чтения и записи в файл используется цифра 6 (2 + 4 = 6), для предоставления полного доступа служит цифра 7 (1 + 2 + 4 = 7). В результате права доступа представляют собой трехзначное восьмеричное число, первое значение в котором относится к владельцу файла, второе — к группе, третье — ко всем остальным.

Для файлов наиболее приемлемые права доступа: чтение и запись для владельца и чтение для всех остальных, т. е. 644. Для того чтобы иметь возможность "заходить" в каталог, для последних необходимо указывать права доступа и на исполнение, поэтому для них следует выставлять 755.

ЗАМЕЧАНИЕ

Права на выполнение выставляются для файлов, которые являются скриптами и которые следует запускать на выполнение. Однако это относится только к запуску скрипта из оболочек операционной системы и через демон cron. PHP-скрипты, которые запускаются Web-сервером Apache, должны иметь права доступа только на чтение.

В табл. 13.17 приводится список функций, предназначенных для обслуживания прав доступа в UNIX-подобных операционных системах.

ЗАМЕЧАНИЕ

Операционная система Windows не поддерживает UNIX-права доступа, поэтому использование функций из табл. 13.17 в этой операционной системе не имеет смысла.

Таблица 13.17. *Функции, обслуживающие UNIX-права доступа*

Функция	**Описание**
`chmod($filename, $mode)`	Устанавливает права доступа `$mode` для файла `$filename`. Возвращает `TRUE` в случае успеха и `FALSE` в случае неудачи
`chgrp($filename, $group)`	Устанавливает группу владельца `$group` (имя группы или уникальный идентификатор GID) для файла `$filename`. Возвращает `TRUE` в случае успеха и `FALSE` в случае неудачи
`chown($filename, $user)`	Устанавливает владельца `$user` (имя владельца или уникальный идентификатор UID) для файла `$filename`. Возвращает `TRUE` в случае успеха и `FALSE` в случае неудачи
`lchgrp($link, $group)`	Осуществляет попытку установить группу владельца `$group` (имя группы или уникальный идентификатор GID) для символической ссылки `$link`. Возвращает `TRUE` в случае успеха и `FALSE` в случае неудачи
`lchown($link, $user)`	Осуществляет попытку установить владельца `$user` (имя владельца или уникальный идентификатор GID) для символической ссылки `$link`. Возвращает `TRUE` в случае успеха и `FALSE` в случае неудачи
`fileperms($filename)`	Возвращает права доступа файла `$filename`. В случае неудачи возвращает `FALSE`
`filegroup($filename)`	Возвращает уникальный идентификатор группы владельца GID. В случае неудачи возвращает `FALSE`
`fileowner($filename)`	Возвращает уникальный идентификатор владельца UID файла `$filename`. В случае неудачи возвращает `FALSE`
`umask([$mask])`	Устанавливает права доступа по умолчанию для вновь создаваемых файлов и каталогов
`is_readable($filename)`	Возвращает `TRUE`, если файл `$filename` доступен для чтения, в противном случае возвращает `FALSE`

Таблица 13.17 *(окончание)*

Функция	Описание
`is_writable($filename)`	Возвращает `TRUE`, если файл `$filename` доступен для записи, в противном случае возвращает `FALSE`
`is_writeable()`	Синоним функции `is_writable()`
`is_executable($filename)`	Возвращает `TRUE`, если файл `$filename` доступен для выполнения, в противном случае возвращает `FALSE`

Установить права доступа можно при помощи специальной функции `chmod()`. В листинге 13.58 демонстрируется скрипт, выставляющей права доступа 644 для файла index.php.

> ***Замечание***
>
> Если Web-сервер Apache выполняется с правами пользователя `nobody`, `apache` или любой другой учетной записи с минимальными правами, непосредственно из PHP-скрипта может не получиться воспользоваться функцией `chmod()`. В этом случае права доступа следует выставлять вручную, используя средства FTP-клиента.

Листинг 13.58. Использование функции `chmod()`

```
<?php
  chmod("index.php", 0644);
?>
```

Как видно из листинга 13.58, функция `chmod()` в качестве второго аргумента принимает восьмеричное число, поэтому его необходимо предварять нулем.

Только что выставленные права доступа можно проконтролировать при помощи функции `fileperms()`, возвращающей числовой код, в котором зашифрованы информация о типе файла (первые 7 бит числа) и его права доступа (последние 9 бит числа). Для того чтобы лучше понять результат, возвращаемый функцией `fileperms()`, удобно преобразовать результат в восьмеричную и двоичную системы счисления (листинг 13.59).

Листинг 13.59. Использование функции `fileperms()`

```
<?php
  // Получаем права доступа и тип файла
  $perms = fileperms("text.txt");
  // Преобразуем результат в восьмеричную систему счисления
  echo decoct($perms); // 100664
  // Преобразуем результат в двоичную систему счисления
  echo decbin($perms); // 1000000110100100
?>
```

Как видно из результатов выполнения скрипта, в восьмеричной системе счисления файлу text.txt назначены права доступа 664 (110100100) — чтение и запись для владельца и группы владельца и чтение для всех остальных. Последовательность 1000000, предшествующая непосредственно правам доступа, сообщает, что перед нами обычный файл. В табл. 13.18 приводится соответствие масок различным типам файлов.

Таблица 3.18. *Битовые маски, соответствующие различным типам файлов*

Маска	Описание
1000000	Обычный файл
1100000	Сокет
1010000	Символическая ссылка
0110000	Специальный блочный файл
0100000	Каталог
0010000	Специальный символьный файл
0001000	FIFO-канал

Для работы с отдельными битами удобно воспользоваться поразрядным пересечением `&` (листинг 13.60).

Замечание

Поразрядные операторы подробно рассматриваются в *разд. 5.4*.

Листинг 13.60. Определение типа файла

```
<?php
  // Получаем права доступа и тип файла
  $perms = fileperms("text.txt");

  // Определяем тип файла
  if (($perms & 0xC000) == 0xC000)
    echo "Сокет";
  elseif (($perms & 0xA000) == 0xA000)
    echo "Символическая ссылка";
  elseif (($perms & 0x8000) == 0x8000)
    echo "Обычный файл";
  elseif (($perms & 0x6000) == 0x6000)
    echo "Специальный блочный файл";
  elseif (($perms & 0x4000) == 0x4000)
    echo "Каталог";
  elseif (($perms & 0x2000) == 0x2000)
```

```
    echo "Специальный символьный файл";
  elseif (($perms & 0x1000) == 0x1000)
    echo "FIFO-канал";
  else
    echo "Неизвестный файл";
?>
```

Работа с отдельными разрядами числа может быть достаточно утомительна и выглядеть загадочно, поэтому PHP предоставляет разработчикам функции `is_readable()`, `is_writable()` и `is_executable()`, которые позволяют определить, доступен ли файл для чтения, записи или исполнения (листинг 13.61).

Листинг 13.61. Использование функций `is_readable()`, `is_writable()` и `is_executable()`

```
<?php
  if (is_readable("text.txt")) echo "Файл доступен для чтения<br>";
  echo "Файл не доступен для чтения<br>";

  if (is_writable("text.txt")) echo "Файл доступен для записи<br>";
  echo "Файл не доступен для записи<br>";

  if (is_executable("text.txt")) echo "Файл доступен для исполнения<br>";
  echo "Файл не доступен для исполнения<br>";
?>
```

13.7. Состояние файловой системы

PHP предоставляет разработчикам функции, позволяющие оценить количество свободного места на жестких дисках. В табл. 3.19 представлены функции данной группы.

***Таблица 3.19.** Функции определения состояния файловой системы*

Функция	Описание
`disk_free_space($part)`	Возвращает размер свободного пространства в байтах для раздела диска `$part`
`diskfreespace()`	Синоним для функции `disk_free_space()`
`disk_total_space($part)`	Возвращает размер раздела диска `$part` в байтах

В листинге 13.62 демонстрируется скрипт, который выводит в окно браузера суммарный объем диска D:, а также количество занятного и свободного пространства.

Листинг 13.62. Использование функций `disk_free_space()` и `disk_total_space()`

```
<?php
  // Общий объем раздела
  $total = disk_total_space("D:");
  // Свободное пространство
  $free = disk_free_space("D:");
  // Занятое пространство
  $busy = $total - $free;

  echo "Объем диска D: составляет $total байт, ".
       "из которых свободно $free байт, а занято $busy байт";
?>
```

13.8. Каталоги

Каталогами называют специальный тип файлов, которые могут содержать ссылки на другие файлы и каталоги. Однако править содержимое таких файлов непосредственно невозможно — это задача ядра операционной системы. Разработчикам предоставляется набор функций, позволяющих создавать, удалять каталоги, а также читать их содержимое (табл. 13.20).

***Таблица 13.20.** Функции для работы с каталогами*

Функция	Описание
`mkdir($pathname [, $mode [, $recursive [, $context]]])`	Создает каталог `$pathname` с правами доступа `$mode`. Если необязательный параметр `$recursive` принимает значение `TRUE`, все несуществующие каталоги в пути `$pathname` создаются. Параметр `$context` позволяет задать заголовки, отправляемые сервером при обращении к сетевому файлу. При успешном создании каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`rmdir($dirname [, $context])`	Удаляет пустой каталог с именем `$dirname`. При успешном удалении каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`getcwd()`	Возвращает полный путь к текущему каталогу
`chdir($directory)`	Изменяет текущий каталог на новый, указанный в параметре `$directory`. При успешной смене текущего каталога функция возвращает `TRUE`, в противном случае — `FALSE`
`chroot($directory)`	Изменяет корневой каталог текущего процесса на `$directory`. При успешной смене корневого каталога возвращает `TRUE`, в противном случае — `FALSE`

Таблица 13.20 *(окончание)*

Функция	Описание
`opendir($path [, $context])`	"Открывает" каталог — функция возвращает дескриптор, позволяющий читать содержимое каталога `$path`
`readdir($dir_handle)`	Читает одну позицию из каталога и передвигает дескриптор на одну позицию вперед. Последовательный вызов функции позволяет прочитать содержимое каталога файл за файлом. Функция принимает в качестве единственного параметра `$dir_handle` дескриптор, полученный ранее при помощи функции `opendir()`. Если достигнут конец каталога, функция возвращает `FALSE`
`closedir($dir_handle)`	Закрывает дескриптор `$dir_handle`, открытый ранее при помощи функции `closedir()`
`rewinddir($dir_handle)`	Помещает дескриптор `$dir_handle` в начало каталога
`scandir($directory [, $sorting_order [, $context]])`	Возвращает массив с содержимым каталога `$directory` или `FALSE`, если `$directory` не является каталогом. Содержимое каталога сортируется в алфавитном порядке, если необязательный параметр `$sorting_order` принимает значение `TRUE`, элементы результирующего массива сортируются в обратном алфавитном порядке
`glob($pattern [, $flags])`	Возвращает массив с именами файлов и подкаталогов, путь к которым удовлетворяет шаблону `$pattern`. Необязательный параметр `$flags` позволяет задать режим интерпретации шаблона

В начале выполнения скрипта ему назначается текущий каталог, именно от него отсчитывается относительный путь, в нем создаются файлы и каталоги. При работе интерпретатора PHP в составе Web-сервера Apache в качестве такого каталога зачастую выступает каталог, в котором расположен скрипт. В любом случае всегда можно уточнить текущий полный путь при помощи функции `getcwd()` (листинг 13.63).

Листинг 13.63. Использование функции `getcwd()`

```
<?php
  // Получаем абсолютный путь к текущему каталогу
  echo getcwd();
?>
```

При необходимости можно назначить новый текущий каталог при помощи функции `chdir()` (листинг 13.64). Такое переназначение особенно актуально при запуске скриптов по cron-заданию или из командной строки, когда текущий каталог по умолчанию может не совпадать с каталогом, где расположен скрипт.

Листинг 13.64. Использование функции `chdir()`

```
<?php
  // Устанавливаем новый каталог по умолчанию
  chdir("/var/www/html/test/");
?>
```

В листинге 13.65 при помощи функции `mkdir()` в текущем каталоге создается новый каталог new. Особенностью функции `mkdir()` является то, что второй параметр позволяет сразу же выставить права доступа на вновь создаваемый каталог.

ЗАМЕЧАНИЕ

Права доступа более подробно обсуждаются в *разд. 13.6*.

Листинг 13.65. Использование функции `mkdir()`

```
<?php
  if (mkdir("new", 0700)) echo "Каталог успешно создан";
  else echo "Ошибка создания каталога";
?>
```

Если требуется удалить каталог, можно воспользоваться функцией `rmdir()` (листинг 13.66).

Листинг 13.66. Использование функции `rmdir()`

```
<?php
  if (rmdir("c:/temp/test")) echo("Каталог успешно удален");
  else echo("Ошибка удаления каталога");
?>
```

Однако, если удаляемый каталог содержит хотя бы один файл или вложенный подкаталог, функция не удалит каталог и вернет `FALSE`. Для того чтобы удалить непустой каталог, необходимо рекурсивно спуститься по всем его подкаталогам и удалить все файлы. Для этого каждый из подкаталогов открывается при помощи функции `opendir()`, после чего его содержимое в цикле читается функцией `readdir()`. Пример такой функции демонстрируется в листинге 13.67.

Листинг 13.67. Рекурсивная функция для удаления каталога

```
<?php
  // Рекурсивная функция удаления каталога с произвольной степенью вложенности
  function full_del_dir($directory)
  {
```

```
    // Открываем каталог
    $dir = opendir($directory);
    // В цикле выводим его содержимое
    while(($file = readdir($dir)) !== FALSE)
    {
      // Если функция readdir() вернула файл — удаляем его
      if (is_file("$directory/$file")) unlink("$directory/$file");
      // Если функция readdir() вернула каталог, и он не равен текущему или
      // родительскому — осуществляем рекурсивный вызов full_del_dir()
      // для этого каталога
      else if (is_dir("$directory/$file") &&
               $file != "." &&
               $file != "..")
      {
        full_del_dir("$directory/$file");
      }
    }
    closedir($dir);
    rmdir($directory);
    echo("Каталог успешно удален");
  }

  // Удаляем каталог temp
  full_del_dir("temp");
?>
```

При обходе дерева каталогов важно следить, чтобы в рекурсивный спуск не попадали каталоги `.` и `..`, обозначающие текущий и родительский каталоги — иначе произойдет зацикливание.

Большинство операций с вложенными каталогами требует их рекурсивного обхода. В листинге 13.68 приводится скрипт, осуществляющий копирование вложенного каталога.

Листинг 13.68. Рекурсивное копирование каталога

```
<?php
  // Копируем содержимое каталога home в home2
  lowering("home","home2");
  ////////////////////////////////////////////////////////
  // Рекурсивная функция спуска
  ////////////////////////////////////////////////////////
  function lowering($dirname,$dirdestination)
  {
```

```
    // Открываем каталог
    $dir = opendir($dirname);
    // В цикле выводим его содержимое
    while (($file = readdir($dir)) !== FALSE)
    {
      echo $file."<br>";
      if (is_file("$dirname/$file"))
      {
        copy("$dirname/$file.", "$dirdestination/$file");
      }
      // Если это каталог - создаем его
      if (is_dir("$dirname/$file") &&
         $file != "." &&
         $file != "..")
      {
        // Создаем каталог
        if (!mkdir($dirdestination."/".$file))
        {
          echo "Невозможно создать каталог $dirdestination/$file";
        }
        // Вызываем рекурсивно функцию lowering
        lowering("$dirname/$file", "$dirdestination/$file");
      }
    }
    // Закрываем каталог
    closedir($dir);
  }
?>
```

В листинге 13.69 демонстрируется скрипт, позволяющий вывести содержимое текущего каталога.

Листинг 13.69. Вывод списка файлов каталога

```
<?php
  // Открываем каталог
  $dir = opendir(".");
  // В цикле выводим его содержимое
  while (($file = readdir($dir)) !== FALSE) echo "$file<br>";
  // Закрываем каталог
  closedir($dir);
?>
```

При просмотре каталога необходимо явно сравнивать результат операции `$file = readdir($dir)` с `FALSE`, т. к. в случае если файл или подкаталог называется "0" или любым другим именем, начинающимся с нуля, то возвращаемое функцией `readdir()` имя может быть приведено к `FALSE`, и цикл обхода может закончиться раньше времени.

Вместо использования комбинации функций `opendir()` и `readdir()` имена файлов и подкаталогов можно также получить при помощи функции `scandir()`, которая имеет следующий синтаксис:

```
scandir($directory [, $sorting_order [, $context]])
```

Функция возвращает массив с содержимым каталога *`$directory`* или `FALSE`, если *`$directory`* не является каталогом. Содержимое каталога сортируется в алфавитном порядке, если необязательный параметр *`$sorting_order`* принимает значение `TRUE`, элементы результирующего массива сортируются в обратном алфавитном порядке.

В листинге 13.70 демонстрируется простой пример, возвращающий массив с именами файлов и каталогов в заданном каталоге.

Листинг 13.70. Использование функции `scandir()`

```
<?php
  $dir = 'dir';
  // сортировка по возрастанию
  $files = scandir($dir);
  // сортировка по убыванию
  $sort_files = scandir($dir, 1);
  // Выводим содержимое массивов
  echo "<pre>";
  print_r($files);
  print_r(sort_files);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 13.70 может выглядеть следующим образом:

```
Array
(
    [0] => array.php
    [1] => file.txt
    [2] => somedir
)
Array
(
    [0] => somedir
    [1] => file.txt
    [2] => array.php
)
```

Еще более гибкие возможности для получения содержимого массива предоставляет функция `glob()`, которая имеет следующий синтаксис:

```
glob($pattern [, $flags])
```

Функция возвращает массив с именами файлов и подкаталогов, путь к которым удовлетворяет шаблону `$pattern`. Необязательный параметр `$flags` принимает одно или несколько значений из табл. 13.21, позволяя задать режим интерпретации шаблона.

***Таблица 13.21.** Возможные значения параметра `$flags` функции `glob()`*

Значение	Описание
`GLOB_MARK`	Добавляет прямой слеш `/` в UNIX-подобных операционных системах и обратный слеш `\` в Windows к тем элементам результирующего массива, которые являются каталогами
`GLOB_NOSORT`	По умолчанию результирующий массив сортируется по алфавиту, использование этого флага отменяет такую сортировку
`GLOB_NOCHECK`	Возвращает шаблон поиска, если не было найдено ни одно соответствие
`GLOB_NOESCAPE`	Имена файлов могут содержать специальные символы, такие как звездочка `*`, знак вопроса `?` или знак доллара `$`. По умолчанию, все эти символы экранируются при помощи обратного слеша. Использование флага `GLOB_NOCHECK` позволяет отменить экранирование
`GLOB_BRACE`	Позволяет задать несколько вариантов в фигурных скобках через запятую, например `{*.html,*.htm,*.php}` вернет все файлы, которые имеют расширения html, htm или php
`GLOB_ONLYDIR`	В результирующий массив попадают только подкаталоги — файлы игнорируются
`GLOB_ERR`	По умолчанию ошибки доступа игнорируются. Флаг `GLOB_ERR` предписывает остановить сканирование каталога и вывести сообщение об ошибке

В листинге 13.71 демонстрируется пример использования функции `glob()`.

Листинг 13.71. Использование функции `glob()`

```
<?php
  // Получаем все HTML- и PHP-файлы текущего каталога
  $arr = glob("{*.html,*.htm,*.php}", GLOB_BRACE | GLOB_MARK);

  // Выводим список файлов
  if (is_array($arr))
  {
    foreach ($arr as $filename)
      echo "$filename (".filesize($filename)." байт)<br>";
  }
?>
```

13.9. Архивные файлы

Сами скрипты и также многочисленные текстовые файлы могут эффективно сжиматься, позволяя экономить дисковое пространство. В PHP для этого используется GZIP-алгоритм сжатия, традиционный для UNIX-подобных операционных систем. В табл. 13.22 приводится список функций, обслуживающих GZIP-файлы (*.gz).

> ***ЗАМЕЧАНИЕ***
>
> Для Windows-пользователей более привычными являются ZIP- и RAR-форматы, однако соответствующие PHP-расширения либо платны, либо малодоступны на серверах хост-провайдеров. Поэтому в данном разделе мы остановимся на GZIP-формате, тем более, что с ним прекрасно работают Windows-утилиты WinRAR и WinZIP.

Таблица 13.22. *Функции для работы с архивными файлами*

Функция	Описание
`gzopen($filename, $mode [, $use_include_path])`	Открывает gz-файл `$filename`. Параметр `$mode` определяет режим. Помимо традиционных режимов `r`, `w`, `b` и `t` может содержать степень сжатия: от 0 (без сжатия) до 9 (максимальная степень). При минимальной степени сжатия все операции осуществляются быстрее, хотя архив получается самым объемным
`gzclose($zp)`	Закрывает ранее открытый gz-файл. Функция принимает единственный параметр `$zp` — дескриптор, полученный при помощи функции `gzopen()`
`gzgetc($zp)`	Читает из gz-файла с дескриптором `$zp` один символ
`gzgets($zp, $length)`	Читает из gz-файла с дескриптором `$zp` строку длиной `$length` символов. Чтение прекращается при достижении максимальной длины, конца строки или конца файла
`gzgetss($zp, $length [, $allowable_tags])`	Аналогична функции `gzgets()`, однако из прочитанной строки удаляются все HTML-теги. Необязательный параметр `$allowable_tags` может содержать допустимые HTML-теги, которые не должны отбрасываться
`gzeof($zp)`	Возвращает `TRUE`, если достигнут конец файла, в противном случае возвращается `FALSE`
`gzread($zp, $length)`	Читает из gz-файла с дескриптором `$zp` строку длиной `$length` символов. Чтение прекращается при достижении максимальной длины или конца файла
`gzpassthru($zp)`	Читает содержимое открытого файла `$zp` от текущей позиции до конца файла и выводит в окно браузера
`gzfile($filename [, $use_include_path])`	Читает gz-файл с именем `$filename` и возвращает его распакованное содержимое в виде массива, каждый элемент которого соответствует отдельной строке. Необязательный параметр `$use_include_path` позволяет задать каталог, где следует искать файл

Таблица 13.22 *(окончание)*

Функция	Описание
`readgzfile($filename [, $use_include_path])`	Читает gz-файл с именем `$filename` и возвращает его распакованное содержимое в окно браузера. Необязательный параметр `$use_include_path` позволяет задать каталог, где следует искать файл
`gzwrite($zp, $string [, $length])`	Записывает в открытый gz-файл `$zp` содержимое строки `$str`. Необязательный параметр `$length` позволяет задать количество байтов, предназначенных для записи (по достижению этого количества запись останавливается)
`gzputs()`	Синоним функции `gzwrite()`
`gztell($zp)`	Возвращает текущее положение файлового указателя в открытом gz-файле `$zp`
`gzseek($zp, $offset)`	Для открытого gz-файла `$zp` устанавливает файловый указатель в новое положение, смещая его на `$offset` байтов относительно начала файла. Если файл открыт для чтения, эта функция симулируется, но будет работать очень медленно. Если файл открыт для записи, то разрешен только поиск вперед. В случае успеха возвращает 0, в случае неудачи –1
`gzrewind($zp)`	Устанавливает файловый указатель в начало gz-файла `$zp`
`gzcompress($data [, $level])`	Сжимает строку `$data` со степенью сжатия `$level`: от 0 (без сжатия) до 9 (максимальная степень). Используется алгоритм сжатия ZLIB (см. RFC 1950)
`gzuncompress($data [, $length])`	Распаковывает строку `$data`, сжатую ранее при помощи функции `gzcompress()`. Необязательный параметр `$level` задает степень сжатия: от 0 (без сжатия) до 9 (максимальная степень)
`gzdeflate($data [, $level])`	Сжимает строку `$data` со степенью сжатия `$level`: от 0 (без сжатия) до 9 (максимальная степень). Используется алгоритм сжатия DEFLATE (см. RFC 1951)
`gzinflate($data [, $length])`	Распаковывает строку `$data`, сжатую ранее при помощи функции `gzdeflate()`. Необязательный параметр `$level` задает степень сжатия: от 0 (без сжатия) до 9 (максимальная степень)
`gzencode($data [, $level [, $encoding_mode]])`	Сжимает строку `$data` со степенью сжатия `$level`: от 0 (без сжатия) до 9 (максимальная степень). Необязательный параметр `$encoding_mode` задает алгоритм сжатия и может принимать либо `FORCE_GZIP` (алгоритм ZLIB), либо `FORCE_DEFLATE` (алгоритм сжатия DEFLATE)
`zlib_get_coding_type()`	Возвращает алгоритм сжатия, используемый по умолчанию, функция может возвращать лишь три значения: `gzip`, `deflate` или `FALSE`

Как видно из табл. 3.22, многие функции для работы с gz-файлами дублируют обычные файловые функции. В листинге 13.72 приводится пример создания архивного gz-файла.

Листинг 13.72. Создание gz-файла

```
<?php
  // Формируем содержимое файла
  $content = "Hello world!\r\nHello PHP!\r\nEnd.";

  // Создаем архивный файл
  $zp = gzopen("archive.gz", "wb9");

  // Проверяем успешность создания файла
  if (!$zp) exit("Невозможно открыть архивный файл");

  // Записываем файл, сжимая строку $content
  gzwrite($zp, $content);

  // Закрываем файл
  gzclose($zp);
?>
```

Если необходимо прочитать содержимое архивного файла archive.gz, можно воспользоваться скриптом из листинга 13.73.

Листинг 13.73. Чтение GZ-файла

```
<?php
  // Открываем архивный файл
  $zp = gzopen("archive.gz", "rb9");

  // Проверяем успешность открытия файла
  if (!$zp) exit("Невозможно открыть архивный файл");

  // Распаковываем и читаем содержимое файла
  while(!gzeof($zp))
  {
    echo gzgets($zp, 1000);
  }

  // Закрываем файл
  gzclose($zp);
?>
```

Извлечь содержимое архивного файла можно и в один прием при помощи функции `readgzfile()` (листинг 13.74).

Листинг 13.74. Распаковка архивного файла

```
<?php
  readgzfile("archive.gz");
?>
```



Глава 14

HTTP-заголовки

Протокол — это набор синтаксических и семантических правил, использующихся при обмене данных между двумя компьютерами. Протокол определяет команды, их синтаксис и порядок отправки, а также очередность отправки команд. Хотя протоколов взаимодействия очень много, Web-разработчики в основном имеют дело с прикладными протоколами и в первую очередь с HTTP-протоколом, который служит для обмена информацией между Web-сервером и браузером (клиентом).

> ***Замечание***
>
> Все стандарты в Интернете оформляются в виде RFC-документов; так, протокол HTTP описывается главным образом в RFC 2616. Ссылки на него можно получить при помощи любой поисковой системы.

Работа протокола HTTP не сводится только к передаче сервером HTML-документа по запросу клиента. Помимо HTML-документа, файла или изображения браузер и сервер обмениваются HTTP-заголовками, через которые клиент может сообщить о своих предпочтениях, типе и версии браузера и операционной системы. Сервер, в свою очередь, может сообщить клиенту об ошибочном запросе, "попросить" его установить cookies и т. п. Язык разметки HTML и серверный язык PHP разрабатывались таким образом, чтобы разработчик мог работать без знания HTTP — это позволят отделить уровни протокола и Web-приложения друг от друга. Тем не менее, совершенно абстрагироваться от протокола HTTP не удается, т. к. его использование зачастую позволяет более гибко управлять работой Web-приложения. В связи с этим в PHP введено несколько функций управления протоколом HTTP, которые будут обсуждаться далее в этой главе.

В ответ на запрос серверу клиент получает HTTP-документ, который состоит из HTTP-заголовков и тела документа, содержащего, как правило, HTML-страницу или изображение (рис. 14.1).

HTTP-заголовки, как правило, формируются сервером автоматически и в большинстве случаев Web-разработчику нет надобности отправлять их вручную. Впрочем, последнее справедливо только для PHP, тогда как, например, при создании CGI-программы при помощи Perl или C разработчик вынужден самостоятельно реализовывать эту часть HTTP-протокола.

> ***Замечание***
>
> HTTP-документ может не содержать тела документа, однако всегда содержит HTTP-заголовки.

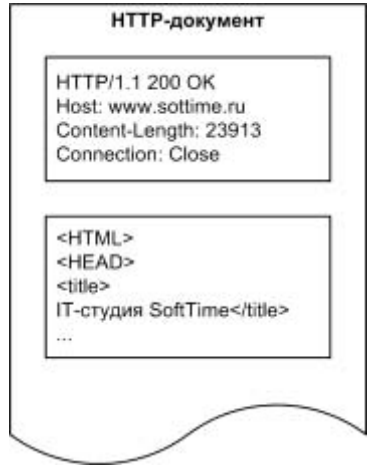


Рис. 14.1. HTTP-документ состоит из HTTP-заголовков и тела документа

Браузер, получая HTTP-заголовки, автоматически выполняет предписания, даже не показывая посетителю их содержимое. Таким образом, HTTP-заголовки служат своеобразной метаинформацией, которой сервер и клиент обмениваются скрыто. Впрочем, иногда необходимо вмешиваться в эту скрытую часть работы клиента и сервера, поскольку она управляет многими важными процессами, такими как переадресация, кэширование, аутентификация и т. п. Кроме того, скрипты сами могут выступать в роли клиентов, обращаясь к страницам других сайтов: в этом случае они вынуждены брать на себя всю работу по реализации и обработке HTTP-протокола.

14.1. Функции для управления HTTP-заголовками

Для управления HTTP-заголовками в PHP предназначены функции, представленные в табл. 14.1.

***Таблица 14.1.** Функции для управления HTTP-заголовками*

Функция	Описание
`header($header [, $replace [, $http_response_code]])`	Отправляет HTTP-заголовок `$header`
`headers_list()`	Возвращает список отправленных или готовых к отправке HTTP-заголовков
`headers_sent([$file [, $line]])`	Проверяет, отправлены ли HTTP-заголовки
`get_headers($url [, $format])`	Позволяет получить массив HTTP-заголовков, которые отправляются удаленным сервером с адресом `$url`. Если необязательный параметр `$format` принимает значение `TRUE`, функция возвращает ассоциативный массив, ключами в котором выступают названия HTTP-заголовков

Функция `header()`, позволяющая отправить клиенту произвольный HTTP-заголовок, имеет следующий синтаксис:

```
header($header [, $replace [, $http_response_code]])
```

Данная функция отправляет HTTP-заголовок `$header`. Второй параметр `$replace` определяет поведение интерпретатора PHP, если тот встречает два одинаковых заголовка: если параметр принимает значение `TRUE`, отправляется последний заголовок, в противном случае отправляется первый заголовок. Третий параметр `$http_response_code` позволяет задать код возврата HTTP.

Простейшей процедурой, которую можно осуществить при помощи функции `header()`, является переадресация, выполняемая при помощи HTTP-заголовка `Location`. В листинге 14.1 представлена переадресация на главную страницу сайта **http://www.softtime.ru**.

> ***Замечание***
>
> Вместо полного сетевого адреса (начинающегося с префикса http://) можно использовать относительные адреса: в этом случае браузер сам подставит адрес сайта.

Листинг 14.1. Переадресация при помощи HTTP-заголовка `Location`

```
<?php
  header("Location: http://www.softtime.ru");
?>
```

Вообще говоря, механизм HTTP-заголовков дублирован в языке разметки HTTP, и добиться схожего поведения можно при помощи передачи HTTP-заголовка через `META`-тег (листинг 14.2).

Листинг 14.2. Переадресация средствами HTML

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV='Refresh' CONTENT='0; URL=http://www.softtime.ru'>
  </HEAD>
</HTML>
```

Если в качестве второго параметра передать функции `header()` значение `TRUE`, то все предыдущие HTTP-заголовки с таким же именем будут заменяться последующими. В листинге 14.3 переадресация на сайт **http://www.softtime.ru** игнорируется, и посетитель будет направлен на сайт **http://www.softtime.biz**.

Листинг 14.3. Переадресация на сайт http://www.softtime.biz

```
<?php
  header("Location: http://www.softtime.ru", TRUE);
  header("Location: http://www.softtime.biz", TRUE);
?>
```

14.2. Кодировка страницы

Обычно по умолчанию Web-сервер настроен таким образом, чтобы автоматически отправлять данные клиенту в определенной кодировке. В листинге 14.4 приводится фрагмент конфигурационного файла httpd.conf Web-сервера Apache, задающего в качестве кодировки по умолчанию Windows-1251.

Листинг 14.4. Фрагмент файла httpd.conf

```
...
AddDefaultCharset WINDOWS-1251
...
```

В настоящий момент используются достаточно мощные серверы, и один сервер может обслуживать сотни сайтов. В связи с этим вероятность того, что все сайты будут использоваться, незначительна. Если сайт отображается в кодировке UTF-8, а сервер отправляет клиенту HTTP-заголовок `Content-Type`, в котором сообщает, что передаваемые данные находятся в кодировке Windows-1251, клиенту придется каждый раз переключать кодировку в браузере. Обойти ситуацию можно несколькими путями: попросить службу технической поддержки сервера явно указать кодировку в секции виртуального хоста, обслуживающего сайт, попытаться прописать кодировку в конфигурационном файле .htaccess (если это разрешено на данном сервере) или явно отправить HTTP-заголовок `Content-Type` из скрипта при помощи функции `header()` (листинг 14.5).

Листинг 14.5. Явное задание кодировки через функцию `header()`

```
<?php
  header("Content-Type: text/html;charset=windows-1251");
  ...
?>
```

14.3. HTTP-коды состояния

На каждый запрос клиента сервер может возвращать HTTP-код состояния, отражающий вид переадресации при окончательном или временном перемещении документа. Если документ найден и успешно отправлен клиенту, в HTTP-заголовки помещается код состояния 200; если документ не найден — 404; в случае переадресации, представленной в листинге 14.3, клиенту отправляется код состояния 302 — "ресурс временно перемещен", иногда бывает полезно изменить код состояния на 301 — "ресурс перемещен постоянно" (листинг 14.6).

Листинг 14.6. Изменение кода состояния

```
<?php
  header("Location: http://www.softtime.biz/");
  header("HTTP/1.1 301 Moved Remanently", FALSE);
?>
```

Коды состояния разделяются на классы, каждый из которых начинается с новой сотни:

- 100—199 — информационные коды состояния, сообщающие клиенту, что сервер пребывает в процессе обработки запроса. Реакция клиента на данные коды не требуется;
- 200—299 — коды успешного выполнения запроса. Получение данного кода ответа на запрос означает, что запрос успешно выполнен и в теле присланного документа находится запрашиваемый документ, который можно передать пользователю;
- 300—399 — коды переадресации, предназначенные для уведомления клиента о том, что для завершения запроса необходимо выполнить дальнейшие действия (как правило, перейти по новому адресу);
- 400—499 — коды ошибочного запроса, отправляемые клиенту, если сервер не может обработать его запрос;
- 500—599 — коды ошибок сервера, возникающих по вине сервера (как правило, из-за синтаксической ошибки в файле .htaccess).

Не все коды состояния имеют смысл; часть из них зарезервирована для дальнейших расширений. В табл. 14.2—14.6 описываются коды состояния для протокола HTTP 1.1, используемого в настоящий момент для распространения HTTP-страниц в Интернете.

Таблица 14.2. *Информационные HTTP- коды состояния*

HTTP-код	Описание
100 Continue	Сервер готов получить оставшуюся часть запроса
101 Switching Protocols	Сервер готов переключить протокол приложения на протокол, указанный в заголовке запроса `Upgrade`, предоставленного клиентом. Переключение должно выполняться, только если указанный протокол имеет преимущество над старым, например, клиент может отправить запрос, чтобы сервер использовал вместо текущего более новый протокол HTTP

Таблица 14.3. *HTTP-коды успешного выполнения запроса*

HTTP-код	Описание
200 OK	Сервер успешно обработал запрос, и клиент может получить запрашиваемый документ в теле ответа
201 Created Location	Сервер успешно создал новый URL, заданный в HTTP-заголовке `Location`
202 Accepted	Запрос принят сервером для обработки, но она еще не завершена
203 Non-Authoritative Information	Метаинформация в заголовке запроса не связана с данным сервером и скопирована с другого сервера
204 No Content	Выполнение запроса завершено, но никакой информации отправлять обратно не требуется. Клиент может продолжить просматривать текущий документ

Таблица 14.3 *(окончание)*

HTTP-код	Описание
205 Reset Content	Клиент должен сбросить текущий документ. Этот HTTP-заголовок можно использовать для сброса и удаления всех значений полей ввода в HTML-форме
206 Partial Content	Сервер выполнил неполный запрос ресурса методом GET. Этот HTTP-код используется для ответа на запросы, содержащие HTTP-заголовок `Range`. Сервер отправляет HTTP-заголовок `Content-Range`, чтобы указать, какой сегмент данных приложен

Таблица 14.4. *HTTP-коды переадресации*

HTTP-код	Описание
300 Multiple Choices	Запрашиваемый ресурс соответствует набору документов. Сервер может отправить информацию о каждом документе с его собственным местоположением и информацией по согласованию содержимого, оставляя выбор на усмотрение клиента
301 Moved Permanently	Запрашиваемый ресурс на сервере отсутствует. Для переадресации клиента на новый URL отправляется HTTP-заголовок `Location`. Все последующие запросы клиент должен отправлять на новый URL
302 Moved Temporarily	Запрашиваемый ресурс временно перемещен. Для переадресации клиента на новый URL отправляется HTTP-заголовок `Location`. В последующих запросах клиент может продолжать использовать старый URL
303 See Other	Запрашиваемый ресурс найден в другом месте, его местоположение уточняется сервером при помощи HTTP-заголовка `Location`
304 Not Modified	Сервер использует этот HTTP-код в соответствии с HTTP-заголовком `If-Modified-Since`. Это означает, что запрашиваемый документ не модифицировался с даты, определенной в HTTP-заголовке `If-Modified-Since`
305 Use Proxy	Для получения запрашиваемого ресурса клиент должен использовать прокси-сервер, адрес которого предается сервером в HTTP-заголовке `Location`
307 Temporary Redirect	Запрашиваемый ресурс временно перемещен в другое место. Для переадресации клиента на новый URL отправляется HTTP-заголовок `Location`

Таблица 14.5. *HTTP-коды ошибочного запроса*

HTTP-код	Описание
400 Bad Request	В запросе клиента обнаружена синтаксическая ошибка
401 Unauthorized	Запрос требует аутентификации клиента. Для уточнения типа аутентификации и области запрашиваемого ресурса сервер отправляет HTTP-заголовок `WWW-Authentificate`
402 Payment Required	Данный HTTP-код зарезервирован для будущего использования в электронной коммерции
403 Forbidden	Доступ к запрашиваемому ресурсу запрещен. Клиент не должен повторять этот запрос
404 Not Found	Запрашиваемый документ отсутствует на сервере
405 Method Not Allowed	Метод запроса, используемый клиентом, неприемлем. Сервер отправляет HTTP-заголовок `Allow`, в котором уточняются допустимые методы для получения доступа к запрашиваемому ресурсу
406 Not Acceptable	Запрашиваемый ресурс недоступен в том формате, который может принимать клиент (клиент обычно уточняет эти форматы в специальном HTTP-заголовке `Accept`). Если запрос не является запросом HEAD на получение лишь HTTP-заголовков с сервера без тела документа, то сервер может отправить заголовки `Content-Language`, `Content-Encoding` и `Content-Type`, чтобы определить, какие форматы являются доступными
407 Proxy Authentification Required	Несанкционированный запрос доступа к прокси-серверу: клиент должен аутентифицировать себя на прокси-сервере. Сервер отправляет HTTP-заголовок `Proxy-Authentificate` со схемой аутентификации и областью запрашиваемого ресурса
408 Request Time-Out	Клиент не завершил свой запрос за время ожидания запроса, заданное серверу, однако может повторить запрос
409 Conflict	Возник конфликт запроса клиента с другим запросом. Вместе с кодом состояния сервер может переслать информацию о типе конфликта
410 Gone	Запрашиваемый ресурс удален с сервера
411 Length Required	Клиент должен прислать в запросе HTTP-заголовок `Content-Length`
412 Precondition Filed	Если запрос клиента содержит один или более HTTP-заголовков `If...`, сервер использует этот HTTP-код для оповещения, что одно или более условий, заданных в этих заголовках, не выполняются
413 Request Entity Too Large	Сервер отказывается выполнять запрос: слишком длинное тело сообщений

Таблица 14.5 *(окончание)*

HTTP-код	Описание
414 Request-URI Too Long	Сервер отказывается выполнять запрос: слишком длинный URI (URL)
415 Unsupported Media Type	Сервер отказывается выполнять запрос: отсутствует поддержка формата тела сообщения
417 Expectation Failed	Сервер не смог выполнить требования HTTP-заголовка `Expect Request`

Таблица 14.6. *HTTP-коды ошибочного запроса*

HTTP-код	Описание
500 Internal Server Error	Ошибка конфигурации сервера или внешней программы
501 Not Implemented	Сервер не поддерживает функции, требуемые для выполнения запроса
502 Bad Gateway	Неверный ответ вышестоящего сервера или прокси-сервера
503 Service Unavailable	Служба временно недоступна. С целью оповещения о времени открытия доступа к службе сервер может отправить заголовок `Retry-After`
504 Gateway Time-Out	Шлюз или прокси-сервер временно заблокирован
505 HTTP Version Not Supported	Не поддерживается используемая клиентом версия протокола HTTP

14.4. Список HTTP-заголовков

При работе с HTTP-заголовками следует помнить, что они всегда предваряют содержимое страницы. Вывод любой информации (при помощи конструкции `echo`, функции `print()` или непосредственно вне тегов `<?php` и `?>`) в окно браузера приводит к тому, что начинается отправка HTTP-документа. Согласно протоколу, HTTP-заголовки отправляются перед телом документа, и последующие попытки отправить дополнительные заголовки заканчиваются неудачей. В листинге 14.7 перед тегом `<?php` расположен перевод строки, что также считается началом документа.

Листинг 14.7. Даже перевод строки перед функцией `header()` блокирует ее работу

```
<?php
  header("Location: http://www.softtime.biz/"); // Ошибка
?>
```

В результате переадресация не осуществляется, а функция `header()` выводит предупреждение "Warning: Cannot modify header information - headers already sent by", сооб-

щающее, что отправка HTTP-заголовка невозможна, поскольку уже начата передача тела HTTP-документа. Иногда пробелы и переводы строк могут быть расположены внутри файлов, включаемых при помощи конструкций `include()` и `require()`.

В ряде случаев нельзя быть заранее уверенным, отправлены HTTP-заголовки или все еще имеется возможность использовать функцию `header()`. Чтобы выяснить это, можно воспользоваться функцией `headers_sent()`, которая имеет следующий синтаксис:

```
headers_sent([$file [, $line]])
```

Функция возвращает `FALSE`, если HTTP-заголовки не были отправлены клиенту, и `TRUE` — в противном случае. После отправки HTTP-заголовков функция при помощи необязательных аргументов `$file` и `$line` может сообщить, в каком файле и строке была начата передача тела HTTP-документа. В листинге 14.8 демонстрируется пример работы функции.

Листинг 14.8. Использование функции `headers_sent()`

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

  if (!headers_sent($filename, $linenum))
  {
    header('Location: http://www.softtime.biz/');
    exit;
  }
  else
  {
    echo "Передача HTTP-документа начата в файле $filename в строке
          $linenum. Поэтому перенаправление на другой ресурс невозможно.";
    exit;
  }
?>
```

Иногда бывает полезно проконтролировать, какие HTTP-заголовки были отправлены клиенту. Для этого удобно воспользоваться функцией `headers_list()`, которая имеет следующий синтаксис:

```
headers_list()
```

Функция возвращает массив, содержащий отправленные клиенту HTTP-заголовки. В листинге 14.9 приводится пример работы функции `headers_list()`. Получаемый в результате ее работы массив для удобства восприятия выводится при помощи функции `print_r()` в тегах `<pre>` и `</pre>`, сохраняющих отступы и переводы строк.

> ***Замечание***
>
> В листинге 14.9 клиенту отправляется произвольный HTTP-заголовок `X-my-header` со значением `"Hello world!"`. Нестандартные HTTP-заголовки, как правило, всегда предваряют префиксом `X-`.

Листинг 14.9. Использование функции `headers_list()`

```
<?php
  header("X-my-header: Hello world!");

  $arr = headers_list();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 14.9 будет следующий дамп массива HTTP-заголовков:

```
Array
(
    [0] => X-Powered-By: PHP/5.1.2
    [1] => X-my-header: Hello world!
)
```

Интересно отметить, что это не полный список HTTP-заголовков, которые получает клиент: это лишь те заголовки, которые отправляет PHP-скрипт. Помимо них клиент получает HTTP-заголовки, отправляемые Web-сервером. Получить полный список HTTP-заголовков к странице можно, обратившись по сетевому адресу при помощи функции `get_headers()`, которая имеет следующий синтаксис:

```
get_headers($url [, $format])
```

Функция возвращает массив HTTP-заголовков, которые отправляются удаленным сервером с адресом `$url`. Если необязательный параметр `$format` принимает значение `TRUE`, в качестве результата возвращается ассоциативный массив, ключами в котором выступают названия HTTP-заголовков.

В листинге 14.10. на примере обращения к серверу **http://www.softtime.ru** демонстрируется пример использования функции `get_headers()`.

Листинг 14.10. Использование функции `get_headers()`

```
<?php
  $arr = get_headers("http://www.softtime.ru");
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 14.10 может быть следующий дамп массива `$arr`:

```
Array
(
```

```
    [0] => HTTP/1.1 200 OK
    [1] => Date: Tue, 08 Jul 2008 13:12:13 GMT
    [2] => Content-Type: text/html; charset=windows-1251
    [3] => Connection: close
    [4] => Server: Apache
    [5] => Cache-Control: max-age=0
    [6] => Expires: Tue, 08 Jul 2008 13:12:13 GMT
)
```

Если функции `get_header()` в качестве второго параметра передать значение `TRUE`, то результирующий массив будет ассоциативным:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [Date] => Tue, 08 Jul 2008 13:15:49 GMT
    [Content-Type] => text/html; charset=windows-1251
    [Connection] => close
    [Server] => Apache
    [Cache-Control] => max-age=0
    [Expires] => Tue, 08 Jul 2008 13:15:49 GMT
)
```

14.5. Подавление кэширования

Механизм кэширования применяется с целью оптимизации пересылки данных между клиентом и сервером. Запрошенный пользователем по HTTP-протоколу документ может быть сохранен в кэше промежуточного сервера или браузера и при повторном его запросе будет выдаваться без обращения к источнику.

Кэши принято подразделять на два вида: локальные и глобальные. Локальный кэш создается браузером клиента, тогда как глобальный располагается на прокси-сервере провайдера (или организации, в которой имеется свой внутренний прокси-сервер).

> ***Замечание***
>
> *Прокси-сервером*, в отличие от обычного сервера, предоставляющего доступ к какому-либо ресурсу, называют сервер-посредник, расположенный между клиентом и обычным сервером. В отличие от шлюз-сервера, осуществляющего обычное транслирование запросов клиента и ответов сервера, в задачи прокси-сервера входит предоставление дополнительных услуг, таких как преобразование медиатипа, анонимная фильтрация, сжатие протокола, кэширование и др.

В большинстве случаев кэширование позволяет ускорить работу с Интернетом и значительно снизить трафик, но иногда кэширование может мешать работе Web-приложений. Если посетители часто загружают страницы Web-сайта с редко изменяющейся информацией, такой, как расписание пригородного транспорта, кэширование полезно, поскольку экономит трафик и сокращает время выполнения запроса. Если же посетитель загружает динамические страницы, например, активного форума, кэширо-

вание, без сомнения, вредно, поскольку содержимое таких страниц меняется слишком часто, и посетитель вынужден будет постоянно вручную обновлять содержимое страницы.

Для подавления кэширования можно использовать HTTP-заголовки, представленные в листинге 14.11.

> ***ЗАМЕЧАНИЕ***
>
> К сожалению, в последнее время кэширующие прокси-серверы настраиваются крайне агрессивно. Дело в том, что динамические страницы, созданные, например, при помощи PHP, вообще не должны подвергаться кэшированию. Однако сплошь и рядом наблюдается обратное. Более того, ряд кэширующих серверов игнорирует HTTP-заголовки, подавляющие кэширование. В этом случае разработчику ничего не остается делать, как добавлять к URL GET-параметр со случайно сгенерированным значением.

Листинг 14.11. Подавление кэширования

```
<?php
  // Любая дата в прошлом
  header("Expires: Mon, 23 May 1995 02:00:00 GMT");
  header("Last-Modified: ".gmdate("D, d M Y H:i:s")." GMT");
  header("Cache-Control: no-cache, must-revalidate");
  header("Pragma: no-cache");
  ...
?>
```

HTTP-заголовок `Expires` задает дату, по достижению которой документ считается устаревшим, поэтому задание для этого заголовка уже прошедшей даты предотвращает кэширование данной страницы.

HTTP-заголовок `Last-Modified` определяет дату последнего изменения Web-страницы. Если с момента последнего обращения к ней прокси-сервера значение параметра этого заголовка изменилось, происходит повторная загрузка страницы, поэтому присвоение этой директиве текущего времени при каждом обращении предотвращает кэширование.

HTTP-заголовок `Cache-Control` предназначен для управления кэшированием, и указание его значения равным `no-cache` также приводит к запрету кэширования. В устаревшем стандарте HTTP 1.0 для запрета кэширования нужно присвоить значение `no-cache` HTTP-заголовку `Pragma`.

Для того чтобы сообщить промежуточному прокси-серверу о том, что данный документ можно кэшировать, HTTP-заголовку `Cache-control` следует передать значение `public`. Если информация не предназначена для публичных кэш-серверов и может быть сохранена только в локальном кэше браузера, HTTP-заголовку `Cache-control` следует передать значение `private` (листинг 14.12).

Листинг 14.12. Кэширование документов

```
<?php
  header("Cache-control: public");
  header("Cache-control: private");
  ...
?>
```

Если содержимое страницы обновляется с определенной регулярностью, задать период обновления можно путем передачи HTTP-заголовку `Cache-control` параметра `max-age`. Данный параметр определяет количество секунд, определяющее время жизни копии страницы в кэше. В листинге 14.13 скрипт сообщает, что страница может храниться на промежуточном прокси-сервере 1 час.

Листинг 14.13. Управление временем кэширования документа

```
<?php
  header("Cache-control: public");
  header("Cache-control: max-age=3600");
  ...
?>
```

Для управления кэшем в PHP предусмотрены две специальные функции: `session_cache_limiter()` и `session_cache_expire()`.

> ***Замечание***
>
> Вообще в PHP достаточно много функций, которые берут на себя формирование и отправку HTTP-заголовков (`session_cache_limiter()`, `session_cache_expire()`, `setcookie()`, `session_start()` и т. п.). При отсутствии буферизации использование таких функций после вывода какой-либо информации в окно браузера приводит к появлению предупреждения: "Warning: Cannot modify header information - headers already sent by".

Функция `session_cache_limiter()` возвращает и/или устанавливает ограничение кэширования и имеет следующий синтаксис:

```
session_cache_limiter([$cache_limiter])
```

Данная функция возвращает одну из следующих строк, определяющих ограничение кэширования:

- `none` — отсутствие каких-либо ограничений, HTTP-заголовок `Cache-control` не отправляется;
- `nocache` — подавление кэширования;
- `private` — HTTP-заголовок `Cache-control` получает значение `"private"`, соответствующее кэшированию на уровне браузера посетителя;

- `private_no_expire` — режим, аналогичный `private`, за исключением того, что клиенту не отсылается HTTP-заголовок `Expires`, который может вызывать неоднозначность в браузерах, построенных на движке Mozilla;
- `public` — HTTP-заголовок `Cache-control` получает значение `"public"`, соответствующее кэшированию на уровне прокси-серверов.

Если определен необязательный параметр `$cache_limiter`, то текущее значение ограничителя заменяется новым. В листинге 14.14 приводится пример использования функции `session_cache_limiter()`.

Листинг 14.14. Использование функции `session_cache_limiter()`

```
<?php
  // Получаем текущее значение ограничения
  $cache_limiter_fst = session_cache_limiter();
  // Устанавливаем ограничение кэша в 'public'
  session_cache_limiter('public');
  // Получаем текущее значение ограничителя
  $cache_limiter_snd = session_cache_limiter();

  // Выводим отчет
  echo "Ограничение на кэш изменено ".
       "с $cache_limiter_fst на $cache_limiter_snd";
?>
```

Результатом работы скрипта из листинга 14.14 будет следующая строка:

```
Ограничение на кэш изменено с nocache на public
```

Функция `session_cache_expire()` возвращает и/или устанавливает текущее время жизни HTTP-документа в кэше и имеет следующий синтаксис:

```
session_cache_expire([$new_cache_expire])
```

Необязательный параметр *`$new_cache_expire`* определяет время жизни документа в кэше в минутах (по умолчанию этот параметр равен 180). Если вызов функции `session_cache_expire()` осуществляется с параметром *`$new_cache_expire`*, то текущее время жизни (180 минут) заменяется значением, переданным в этом параметре (листинг 14.15).

Листинг 14.15. Использование функции `session_cache_expire()`

```
<?php
  // Устанавливаем время жизни кэша равным 30 минутам
  $cache_expire =  session_cache_expire(30*60);
  echo $cache_expire
?>
```



Глава 15

Cookie

HTTP-протокол, лежащий в основе Интернета, не сохраняет информации о состоянии сеанса. Это означает, что любое обращение клиента сервер воспринимает как обращение нового клиента, и даже если клиент формирует запрос для загрузки картинок с текущей страницы, сервером он воспринимается как запрос нового клиента, никак не связанного с тем, который только что загрузил страницу. Данная схема достаточно хорошо работала для статических страниц, но стала совершенно неприемлемой для динамических. В связи с этим в протокол HTTP был введен механизм cookie, который в настоящий момент поддерживают все участники Интернета: клиенты, прокси-серверы и конечные серверы.

> ***Замечание***
>
> При помощи механизма cookie не составляет труда построить механизм сессий, который бы запоминал состояния для каждого клиента. Можно самостоятельно создавать такой механизм, можно воспользоваться встроенным в язык программирования PHP *(см. главу 16)*.

Cookies — это небольшие файлы, сохраняемые просматриваемыми серверами на машине посетителя и содержащие текстовую информацию о настройках пользователя, доступную для считывания создавшему их серверу. Дословно cookie переводится как "кекс" или сладкий бонус, выдаваемый клиентам ресторана, чтобы они запомнили его и посетили вторично. Из-за достаточно сумбурного английского названия, для cookie так и не было подобранного адекватного русского перевода, поэтому в этой и последующих главах будет использоваться английское название.

Для создания cookie предназначена функция `setcookie()`, которая имеет следующий синтаксис:

```
setcookie($name [, $value [, $expire [, $path [, $domain [, $secure]]]]])
```

Функция принимает следующие аргументы:

- `$name` — имя cookie;
- `$value` — значение, хранящееся в cookie с именем `name`;
- `$expire` — время в секундах, прошедших с 0 часов 00 минут 1 января 1970 года. По истечении этого времени cookie удаляется с машины клиента;
- `$path` — путь, по которому доступен cookie;
- `$domain` — домен, из которого доступен cookie;

- `$secure` — директива, определяющая, доступен ли файл cookie по защищенному протоколу HTPPS. По умолчанию эта директива имеет значение 0, что означает возможность доступа к cookie по стандартному незащищенному протоколу HTTP.

Функция `setcookie()` возвращает `TRUE` при успешной установке cookie на машине клиента и `FALSE` — в противном случае. После того как cookie установлен, его значение можно получить на всех страницах Web-приложения, обращаясь к суперглобальному массиву `$_COOKIE` и используя в качестве ключа имя cookie.

Так как cookie передается в заголовке HTTP-запроса, то вызов функции `setcookie()` необходимо размещать до начала вывода информации в окно браузера функциями `echo()`, `print()` и т. п., а также до включения в файл HTML-тегов. Работа с cookie без установки времени жизни продемонстрирована в листинге 15.1.

> ***ЗАМЕЧАНИЕ***
>
> Файл на жестком диске для cookie создается только в том случае, если выставляется время жизни cookie; в противном случае cookie действует только до конца сеанса, т. е. до того момента, пока пользователь не закроет окно браузера. В этом случае говорят о *сессионном cookie*.

Листинг 15.1. Подсчет количества обращений к странице

```
<?php
  // Выставляем уровень обработки ошибок
  // (http://www.softtime.ru/info/articlephp.php?id_article=23)
  error_reporting(E_ALL & ~E_NOTICE);

  // Увеличиваем значение cookie
  $_COOKIE['counter']++;
  // Устанавливаем cookie
  setcookie("counter", $_COOKIE['counter']);
  // Выводим значение cookie
  echo "Вы посетили эту страницу $_COOKIE[counter] раз";
?>
```

Результат выполнения скрипта, приведенного в листинге 15.1, показан на рис. 15.1.

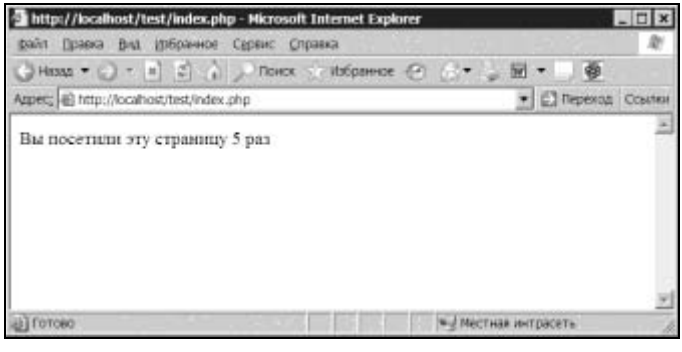


Рис. 15.1. Подсчет количества посещений при помощи cookies

В листинге 15.1 реализована установка cookie с именем `counter`, значение которого увеличивается при каждой перезагрузке страницы.

Время жизни cookie задается при помощи функций `time()` или `mktime()` (листинг 15.2).

Листинг 15.2. Установка cookie с определенным временем жизни

```
<?php
  // cookie действителен в течение 10 мин после создания
  setcookie("name", "value", time() + 600);
  // действие этого cookie прекращается в полночь 25 января 2010 года
  setcookie("name", "value", mktime(0,0,0,1,25,2010));
  // действие этого cookie прекращается в 18.00 25 января 2010 года
  setcookie("name", "value", mktime(18,0,0,1,25,2010));
?>
```

ЗАМЕЧАНИЕ

В большинстве современных систем, где время представляется 32-битным целым числом, допустимыми являются значения года между 1901 и 2038. Типичной ошибкой при работе с cookie является установка года, выходящего за пределы этого интервала, например, 2100. В этом случае cookie будут устанавливаться только как сессионные.

Ограничить доступ к cookie со всех страниц, кроме расположенных в определенном каталоге (например, /web), можно при помощи четвертого параметра (листинг 15.3).

Листинг 15.3. Ограничение доступа к cookie

```
<?php
  setcookie("name", "value", time() + 600, "/web");
?>
```

Однако и в этом случае каталоги /web/index.php, /web1/page.html и т. д. будут удовлетворять этому ограничению. Если это нежелательно, можно ограничить область видимости cookie до конкретной страницы, как это продемонстрировано в листинге 15.4.

Листинг 15.4. Сужение области видимости cookie

```
<?php
  setcookie("name", "value", time() + 600, "/web/index.php");
?>
```

Однако и такой способ в полной мере не решает проблему, т. к. в этом случае доступ к информации, содержащейся в cookie, может получить, к примеру, скрипт /web/index.php-script/anti_cookie.php. Во избежание доступа с посторонних хостов можно

еще более сузить область видимости, задав хост, страницы которого имеют доступ к cookie, как это продемонстрировано, например, в листинге 15.5.

Листинг 15.5. Устанавливаем хост, с которого виден cookie

```
<?php
  setcookie("name", "value", time() + 600, "/", "www.softtime.ru");
?>
```

Уничтожить cookie можно, установив нулевое время жизни (листинг 15.6).

Листинг 15.6. Удаление cookie

```
<?php
  setcookie("name", "value");
?>
```

Замечание

Следует помнить, что для уничтожения cookie с ограничениями по каталогу и самому каталогу необходимо выставлять точно такие же ограничения для его удаления, в противном случае он не будет уничтожен.

Cookie можно установить в обход функции `setcookie()`, используя HTTP-заголовок `Set-Cookie` и функцию `header()` (листинг 15.7).

Листинг 15.7. Ручная отправка cookie клиенту

```
<?php
  $cookie = "Set-Cookie: name=value; ".
            "expires=Thu, 30-Aug-2008 13:00:04 GMT; ".
            "path=/; ".
            "domain=www.softtime.ru";
  header($cookie);
?>
```

Нередко посетители отключают cookies в настройках своих браузеров. Для корректной работы в Web-приложение, использующее cookie, необходимо помещать код, проверяющий, включены ли cookies у посетителя. Такая проверка позволит использовать другой механизм аутентификации, например, сессии, или просто разрешит вывести сообщение о необходимости включить cookies. Пример такой проверки приведен в листинге 15.8.

Листинг 15.8. Проверка, включены ли cookies

```
<?php
  // Выставляем уровень обработки ошибок
  // (http://www.softtime.ru/info/articlephp.php?id_article=23)
```

```
  error_reporting(E_ALL & ~E_NOTICE);

  if(!isset($_GET['probe']))
  {
    // Устанавливаем cookie с именем "test"
    if (setcookie("test","set"));
    // Отправляем заголовок переадресации на страницу,
    // с которой будет предпринята попытка установить cookie
    header("Location: $_SERVER[PHP_SELF]?probe=set");
  }
  else
  {
    if(!isset($_COOKIE['test']))
    {
      echo "Для корректной работы приложения необходимо включить cookies";
    }
    else
    {
      echo "Cookies включены";
    }
  }
?>
```

ЗАМЕЧАНИЕ

Суперглобальный массив `$_SERVER` содержит переменные окружения, которые Web-сервер передает скрипту. Среди переменных окружения наибольшей популярностью пользуется `$_SERVER['PHP_SELF']`, сообщающая имя текущего файла. Текущие значения переменных окружения можно посмотреть в отчете, формируемом функцией `phpinfo()`.

В листинге 15.8 для проверки корректности работы с cookies при помощи функции `setcookie()` устанавливается пробное значение cookie. Функция `setcookie()` в данном случае принимает два параметра, первый из которых представляет собой имя, а второй — значение `cookie`. Далее осуществляется перегрузка текущей страницы с передачей через GET-параметр `probe` значения `set`, тем самым сообщая скрипту, что проверка выполнена, и необходимо проверить, содержит ли что-нибудь элемент `$_COOKIE['test']`. Если данный элемент не установлен — cookies отключены, если он содержит значение — включены.



Глава 16

Сессии

Сессия во многом походит на cookie и представляет собой текстовый файл, хранящий пары "ключ/значение", но уже не на машине клиента, а на сервере. Во многих случаях сессии являются более предпочтительным вариантом, чем cookies. Для каждого из посетителей сохраняется собственный набор данных в отдельном файле. Изменение данных одного посетителя никак не отражается на данных другого посетителя.

Так как на сервере скапливается большое количество файлов, принадлежащих сессиям разных клиентов, то для их идентификации каждому новому клиенту назначается уникальный номер — идентификатор сессии (SID), который передается либо через строку запроса, либо через cookies, если они доступны. К недостаткам сессий относится невозможность контроля времени их жизни из PHP-скриптов, т. к. этот параметр задается в конфигурационном файле php.ini директивой `session.cookie_lifetime` и может быть запрещен к изменению из скрипта.

Оба механизма, сессии и cookies, взаимно дополняют друг друга. Cookies хранятся на компьютере посетителя, и продолжительность их жизни определяет разработчик. Обычно они применяются для долгосрочных задач (от нескольких часов) и хранения информации, которая относится исключительно к конкретному посетителю (личные настройки, логины, пароли и т. п.). В свою очередь, сессии хранятся на сервере, и продолжительность их существования (обычно небольшую) определяет администратор сервера. Они предназначены для краткосрочных задач (до нескольких часов) и хранения и обработки информации обо всех посетителях в целом (количество посетителей on-line и т. п.). Поэтому использовать тот или иной механизм следует в зависимости от преследуемых целей.

Директива `session.save_path` в конфигурационном файле php.ini позволяет задать путь к каталогу, в котором сохраняются файлы сессий; это может быть удобным для отладки Web-приложений на локальном сервере. Если данная директива не задана, сессии хранятся в оперативной памяти сервера.

> ***Замечание***
>
> Более подробно настройки сессии при помощи конфигурационного файла php.ini обсуждаются в *разд. 20.1.12*.

16.1. Функции обработки сессии

Для работы с сессиями предназначен специальный набор функций, представленных в табл. 16.1.

Таблица 16.1. *Функции обработки сессии*

Функция	Описание
`session_start()`	Инициирует сессию, любая страница, использующая сессию, должна иметь в начале вызов этой функции
`session_register($key [, $value])`	Устаревшая функция для помещения ключа `$key` и значения `$value` в сессию, в настоящий момент вместо вызова этой функции используется прямое помещение данных в суперглобальный массив `$_SESSION`. Например, `$_SESSION[$key] = $value`
`session_is_registered($key)`	Устаревшая функция, возвращающая `TRUE`, если в сессию помещено значение с ключом `$key`, и `FALSE` в противном случае. В настоящий момент существование элемента в `$_SESSION[$key]` принято проверять при помощи конструкции `isset()`
`session_unregister($key)`	Устаревшая функция для удаления элемента сессии с ключом `$key`. В настоящее время для удаления элемента `$_SESSION[$key]` используется конструкция `unset()`
`session_id([$id])`	Каждой сессии назначается уникальный идентификатор SID, по которому сессия одного посетителя отличается от сессии другого. Функция `session_id()` возвращает уникальный идентификатор сессии для текущего посетителя. Необязательный параметр `$id` позволяет назначить текущей сессии новый идентификатор
`session_regenerate_id ([$delete])`	Вызов функции приводит к обновлению уникального идентификатора сессии SID (генерируется новое значение). Если необязательный флаг `$delete` принимает значение `TRUE`, файл с данными сессии уничтожается (по умолчанию параметр принимает значение `FALSE`). Функция возвращает `TRUE` в случае успешного обновления сессии и `FALSE` в противном случае
`session_save_path([$path])`	Возвращает путь к временному каталогу, где хранятся файлы сессии. При помощи необязательного параметра `$path` можно задать новый каталог для файлов сессии. Установить каталог для временных файлов сессии можно при помощи директивы `session.save_path` конфигурационного файла php.ini
`session_name($name])`	Позволяет создать пространство имен `$name`, данные из разных пространств имен обрабатываются независимо. Таким образом, можно разделить приложения, использующие одинаковые ключи в двух разных сессиях, использующихся одновременно

Таблица 16.1 *(окончание)*

Функция	Описание
`session_get_cookie_params()`	Данные сессии хранятся на сервере, однако уникальный идентификатор сессии SID передается клиенту через cookies. Функция `session_get_cookie_params()` возвращает массив с параметрами cookies, используемой для обмена SID между клиентом и сервером
`session_destroy()`	Уничтожает все данные, помещенные в сессию. Возвращает `TRUE` в случае успеха и `FALSE` в противном случае

Перед тем как начать работать с сессией, клиенту должен быть установлен cookie с уникальным идентификатором SID, а на сервере должен быть создан файл с данными сессии. Все эти начальные действия выполняет функция `session_start()`. Эта функция должна вызываться на каждой странице, где происходит обращение к переменным сессии.

Точно так же, как и функция `setcookie()`, функция `session_start()` должна вызываться до начала вывода информации в окно браузера, т. к. уникальный идентификатор сессии SID передается через cookie, т. е. посредством HTTP-заголовка `Set-Cookie`.

> ***Замечание***
>
> Установив директиве `session.auto_start` конфигурационного файла php.ini значение 1, можно добиться автоматического старта сессии, без вызова функции `session_start()`.

После инициализации сессии появляется возможность сохранять информацию в суперглобальном массиве `$_SESSION`. В листинге 16.1 на странице index.php в массив `$_SESSION` сохраняется переменная и массив.

Листинг 16.1. Помещаем данные в суперглобальный массив `$_SESSION`

```
<?php
  // Инициируем сессию
  session_start();

  // Помещаем значение в сессию
  $_SESSION['name'] = "value";

  // Помещаем массив в сессию
  $arr = array("first", "second", "third");
  $_SESSION['arr'] = $arr;

  // Выводим ссылку на другую страницу
  echo "<a href='other.php'>другая страница</a>"
?>
```

На страницах, где происходит вызов функции session_start(), значения данных переменных можно извлечь из суперглобального массива $_SESSION. В листинге 16.2 приводится содержимое страницы other.php, где извлекаются данные, ранее помещенные на странице index.php.

> ***Замечание***
>
> Массив $_SESSION перед сохранением в файл подвергается сериализации, поэтому хранить сериализованные данные в сессии нельзя, т. к. массивы, два раза подряд подвергающиеся действию функции serialize(), восстановлению не подлежат.

Листинг 16.2. Извлечение данных из суперглобального массива $_SESSION

```
<?php
  // Инициируем сессию
  session_start();
  // Выводим содержимое суперглобального массива $_SESSION
  echo "<pre>";
  print_r($_SESSION);
  echo "</pre>";
?>
```

Результат работы скрипта выглядит следующим образом:

```
Array
(
    [name] => value
    [arr] => Array
        (
            [0] => first
            [1] => second
            [2] => third
        )

)
```

Завершить работу сессии можно, вызвав функцию session_destroy(), которая имеет следующий синтаксис:

```
session_destroy()
```

Функция возвращает TRUE при успешном уничтожении сессии и FALSE в противном случае. Если достаточно не уничтожать текущую сессию, а лишь обнулить все значения, хранящиеся в сессии, следует вызвать функцию unset(), которая уничтожит все элементы в суперглобальном массиве $_SESSION текущей сессии. Данная функция точно так же, как и функция session_destroy(), не принимает ни одного параметра и возвращает TRUE в случае успеха и FALSE в случае неудачи.

Точно также уничтожается отдельный элемент суперглобального массива `$_SESSION`:

```
unset($_SESSION['name'])
```

Еще одной полезной функцией является функция `session_id()`, позволяющая узнать текущий идентификатор сессии или задать собственный идентификатор и имеющая следующий синтаксис:

```
session_id([$id])
```

Функция возвращает текущий идентификатор сессии. Необязательный параметр `$id` позволяет задать собственный идентификатор сессии, который должен состоять из строчных или прописных букв латинского алфавита и цифр. При задании собственного идентификатора функция `session_id()` должна вызываться до функции `session_start()` (листинг 16.3).

Листинг 16.3. Узнаем текущий идентификатор сессии

```
<?php
  // Инициируем сессию
  session_start();

  // Узнаем текущий идентификатор сессии
  echo session_id();
?>
```

Возможным результатом работы скрипта в листинге 16.3 может быть строка:

```
a27e8c4724f07cae913c50e55cea1b38
```

16.2. Перегрузка механизма сессии

Собственный механизм сессий можно реализовать при помощи функции `session_set_save_handler()`, которая имеет следующий синтаксис:

```
session_set_save_handler($open, $close, $read, $write, $destroy, $gc)
```

Функция принимает шесть аргументов-строк — названий функций-обработчиков:

- `$open` — метод, который выполняется в момент инициализации сессии в клиентском коде посредством функции `session_start()`, здесь мы разместим код установки соединения с базой данных;
- `$close` — метод, осуществляющий завершающие действия, здесь мы разместим код закрытия соединения с базой данных;
- `$read` — метод, извлекающий данные из таблицы базы данных для текущей сессии;
- `$write` — метод, добавляющий новую запись в таблицу базы данных, если это первое обращение к сессии, и обновляющий запись, соответствующую строке, если это последующее обращение;

- *$destroy* — метод, вызываемый для уничтожения сессии, здесь будет размещено уничтожение записи, соответствующей текущей сессии;
- *$gc* — метод, осуществляющий "сборку мусора", удаляющий старые записи, потерявшие актуальность.

Скрипт session.php, реализующий собственный механизм сессий, представлен в листинге 16.4.

Листинг 16.4. Файл session.php

```
<?php
  // Инициализация сессии
  function open($save_path, $session_name)
  {
    global $sess_save_path;

    $sess_save_path = $save_path;
    return TRUE;
  }

  // Закрытие сессии
  function close()
  {
    return TRUE;
  }

  // Извлечение данных сессии
  function read($id)
  {
    global $sess_save_path;

    $sess_file = "$sess_save_path/sess_$id";
    return @file_get_contents($sess_file);
  }

  // Помещение данных в сессию
  function write($id, $sess_data)
  {
    echo $id;
    global $sess_save_path;

    $sess_file = "$sess_save_path/sess_$id";
    if ($fp = @fopen($sess_file, "w")) {
```

```
      $return = fwrite($fp, $sess_data);
      fclose($fp);
      return $return;
    }
    else
    {
      return FALSE;
    }
  }

  // Уничтожение сессии
  function destroy($id)
  {
    global $sess_save_path;

    $sess_file = "$sess_save_path/sess_$id";
    return(@unlink($sess_file));
  }

  // Сборка мусора
  function gc($maxlifetime)
  {
    global $sess_save_path;

    foreach (glob("$sess_save_path/sess_*") as $filename)
    {
      if (filemtime($filename) + $maxlifetime < time())
      {
        @unlink($filename);
      }
    }
    return TRUE;
  }

  // Перегрузка механизма сессии
  session_set_save_handler("open", "close", "read",
                           "write", "destroy", "gc");

  session_start();
?>
```

Файл session.php необходимо подключить вместо вызова функции `session_start()` так, как это продемонстрировано в файле index.php (листинг 16.5).

Листинг 16.5. Использование собственного механизма сессий (index.php)

```
<?php
  include "session.php";
  $_SESSION['name'] = "cheops";
?>
<a href='index1.php'>Переход на другую страницу</a>
```

В целом же работа с данным механизмом ничем не отличается от обычного механизма сессий, реализованного по умолчанию. Это демонстрирует скрипт index1.php (листинг 16.6).

Листинг 16.6. Файл index1.php

```
<?php
  include "session.php";
  echo "Здравствуйте, $_SESSION[name]!<br>";
?>
<a href='index.php'>Переход на другую страницу</a>
```

Результат работы скрипта index1.php из листинга 16.6 представлен на рис. 16.1.

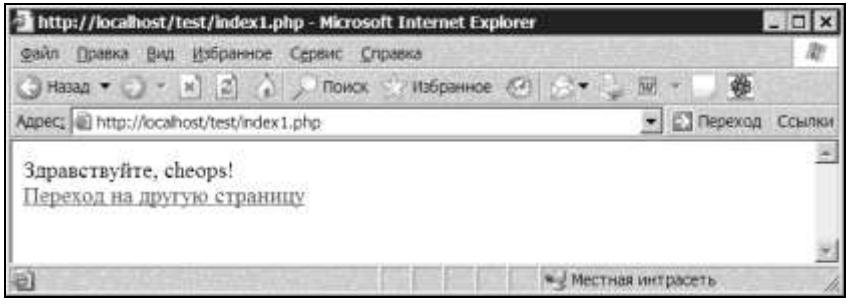


Рис. 16.1. Вывод приветствия при помощи механизма сессий

Механизм перегрузки сессий удобен в том случае, когда сессии необходимо хранить в базе данных, например, для того, чтобы отслеживать количество посетителей в каждый момент времени.



Глава 17

Управление выводом

В объемном проекте часто сложно проконтролировать все возможные ситуации, и рано или поздно отображение информации в окно браузера осуществляется перед выполнением функции `header()`, `session_start()` или `setcookie()`, что приводит к выводу предупреждения: "Warning: Cannot modify header information — headers already sent by". Такая ситуация может возникнуть, например, при наличии случайного пробела или какой-либо ошибки, которая ни разу не возникала в процессе тестирования. Не стоит и говорить, что вывод подобного предупреждения искажает дизайн сайта и выглядит непрофессионально. Конечно, можно подавить вывод сообщения, предварив функцию `header()` символом `@` или отключив вывод сообщений в окно браузера при помощи вызова функции `error_reporting(0)`, однако это лишь скроет предупреждение и не заставит работать функцию `header()`.

Выходом из ситуации является буферизация вывода, при которой весь объем данных, выводимых в окно браузера, помещается в буфер и отправляется клиенту только тогда, когда скрипт завершает работу. Это позволяет безболезненно выполнить все функции `header()` в теле скрипта, где бы они ни находились, а также осуществлять преобразование содержимого страницы перед отправкой ее клиенту.

17.1. Функции управления выводом

Самый простой способ установить буферизацию вывода — воспользоваться директивой `output_buffering` конфигурационного файла php.ini, установив ее значение равной `On` (листинг 17.1).

Листинг 17.1. Включение буферизации при помощи директивы `output_buffering`

```
...
output_buffering = On
...
```

Настройки конфигурационного файла php.ini могут отличаться на разных серверах, поэтому при переносе приложения не стоит рассчитывать на то, что директива `output_buffering` окажется включенной (тем более, что по умолчанию она принимает значение `Off` — отключена).

К счастью, PHP предоставляет альтернативный способ управления буферизаций вывода. Для этого предназначены функции группы управления выводом, список которых представлен в табл. 17.1.

***Таблица 17.1.** Функции управления выводом*

Функция	Описание
`ob_start([$output_callback [, $chunk_size [, $erase]]])`	Включает буферизацию вывода; в качестве параметра `$output_callback` может принимать название функции обратного вызова, которая задает обработчик. Необязательный параметр `$chunk_size` задает количество байтов буфера, через которые будет осуществляться повторный вызов функции `$output_callback`. Если необязательный параметр `$erase` принимает значение `FALSE`, то буфер не будет удаляться после завершения работы скрипта
`ob_clean()`	Очищает буфер вывода
`ob_end_clean()`	Очищает буфер вывода и отключает буферизацию вывода
`ob_end_flush()`	Отправляет буфер вывода клиенту и отключает буферизацию вывода
`ob_flush()`	Отправляет буфер вывода клиенту
`ob_get_clean()`	Возвращает текущее содержимое буфера и удаляет текущий буфер
`ob_get_contents()`	Возвращает содержимое буфера вывода
`ob_get_flush()`	Очищает буфер, возвращая его содержимое в виде строки, и отключает буферизацию
`ob_get_length()`	Возвращает размер буфера вывода
`ob_get_level()`	Возвращает уровень вложения буфера
`ob_get_status([$full_status])`	Возвращает статус буфера вывода в виде массива; если необязательный параметр `$full_status` принимает значение `TRUE`, возвращается более подробная информация
`ob_gzhandler($buffer, $mode)`	Функция обратного вызова, которая используется функцией `ob_start()` для сжатия содержимого буфера
`ob_iconv_handler($contents, $status)`	Функция обратного вызова, которая используется функцией `ob_start()` для перекодировки содержимого буфера
`ob_tidyhandler()`	Функция обратного вызова, которая используется функцией `ob_start()` для восстановления содержимого буфера

***Таблица 17.1** (окончание)*

Функция	Описание
`ob_implicit_flush([$flag])`	Включает (если значение параметра `$flag` устанавливается в 1) или выключает (если значение `$flag` устанавливается в 0) неявную очистку буфера
`ob_list_handlers()`	Возвращает список всех используемых обработчиков буферизации
`output_add_rewrite_var($name, $value)`	Добавляет к URL, обнаруженным в буфере вывода, дополнительный GET-параметр с именем `$name` и значением `$value`
`output_reset_rewrite_vars()`	Уничтожает дополнительные GET-параметры, добавленные функцией `output_add_rewrite_var()`

В листинге 17.2 приводится пример буферизации вывода при помощи функций из табл. 17.1.

Листинг 17.2. Буферизация вывода

```
<?php
  // Весь вывод направляем в буфер
  ob_start();

  // Выводим содержимое страницы
  echo "Hello world";
  // Отправляем HTTP-заголовок
  header("X-my-header: Hello world!");

  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

Несмотря на то, что функция `header()` используется после вывода текста в окно браузера, предупреждение "Warning: Cannot modify header information — headers already sent by" не возникает, и HTTP-заголовок успешно отправляется клиенту.

17.2. Размер и тип документа

При загрузке объемного файла сервер обычно отправляет размер документа в байтах при помощи HTTP-заголовка `Content-length`. Это позволяет оценить объем загружаемого документа (файла), а также время, необходимое для его загрузки. Следует помнить, что Web-сервер способен вычислить только размер статического файла,

например, изображения, HTML-страницы или zip-архива. Для динамических файлов, таких как PHP-скрипт, отправка размера документа при помощи HTTP-заголовка `Content-length` ложится на плечи разработчика. Однако часто определить размер документа можно только после того, как скрипт завершает свою работу и клиенту отправлена значительная часть HTTP-документа. В такой ситуации удобно воспользоваться функциями управления вывода, рассмотренными в предыдущем разделе.

В листинге 17.3 представлен скрипт, формирующий объемный текстовый файл, размер которого вычисляется при помощи функции `ob_get_length()` и отправляется клиенту при помощи HTTP-заголовка `Content-length`.

Листинг 17.3. Формирование HTTP-заголовка `Content-length`

```
<?php
  // Весь вывод направляем в буфер
  ob_start();

  // Выводим содержимое страницы
  for($i = 0; $i < 300000; $i++) echo "01\r\n";

  // Отправляем клиенту размер страницы в HTTP-заголовке Content-length
  header("Content-length: ".ob_get_length());

  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

При загрузке скрипта из листинга 17.3 Web-сервер и браузер интерпретируют его содержимое как HTML-документ, который выводится в окно браузера. Поскольку HTTP-заголовок `Content-type`, устанавливающий тип документа, в листинге 17.3 отсутствует, сервер назначает ему значение `text/html`, и браузер интерпретирует такой документ по умолчанию как HTML-страницу. Однако такая интерпретация не всегда удобна: для объемных файлов было бы разумнее предложить пользователю сохранить его на жесткий диск, а для этого необходимо явно изменить тип файла. В листинге 17.4 приводится модифицированный вариант скрипта из листинга 17.3.

Замечание

Подход, представленный в листинге 17.4, может использоваться для загрузки файлов любых типов: текстовых документов, изображений и т. п., для которых требуется сохранение файла на жесткий диск, а не интерпретация его браузером.

Листинг 17.4. Изменение типа файла

```
<?php
  // Весь вывод направляем в буфер
  ob_start();

  // Выводим содержимое страницы
  for($i = 0; $i < 300000; $i++) echo "01<br>";

  // Задаем имя, которое будет предложено клиенту для сохранения файла
  header("Content-Disposition: attachment; filename=text.txt");
  // В качестве типа файла задаем бинарный поток
  header("Content-type: application/octet-stream");
  // Отправляем клиенту размер страницы в HTTP-заголовке Content-length
  header("Content-length: ".ob_get_length());

  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

В качестве типа документа в HTTP-заголовке `Content-type` указывается значение `application/octet-stream`, соответствующее бинарному потоку. Заголовок `Content-Disposition` позволяет задать имя файла, в данном случае text.txt. В результате загрузки страницы со скриптом из листинга 17.4 на экран посетителя будет выведено окно, представленное на рис. 17.1.

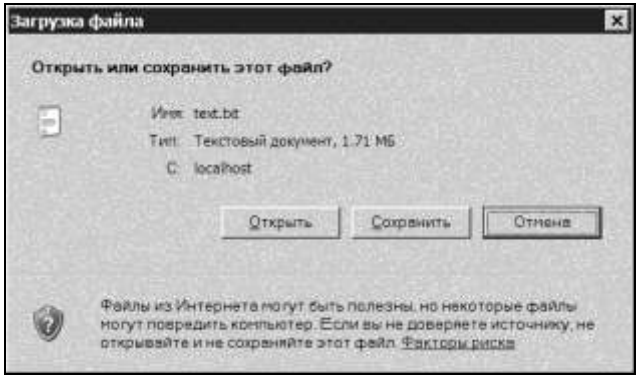


Рис. 17.1. Предложение сохранить файл вместо отображения его содержимого в окне браузера

При этом если в листинге 17.4 будет случайно или умышленно пропущен HTTP-заголовок `Content-type`, то размер загружаемого файла не отобразится (рис. 17.2).

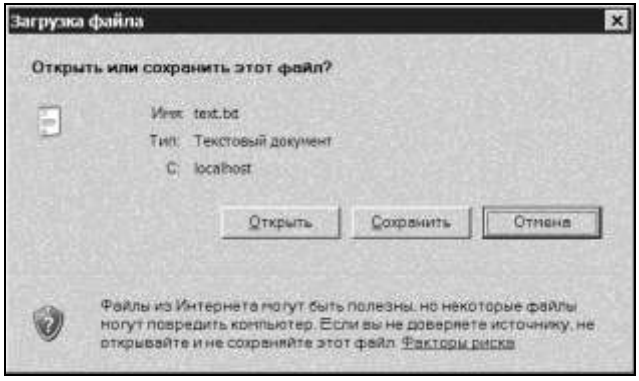


Рис. 17.2. Если HTTP-заголовок `Content-length` отсутствует, размер загружаемого файла не указывается

17.3. Подсветка ссылок

Пусть текстовая страница содержит множество адресов на другие страницы в виде обычного текста **http://www.softtime.ru**. При помощи буферизации содержимого и регулярных выражений можно заменить все ссылки на гиперссылки `<a href='http://www.softtime.ru'>http://www.softtime.ru</a>`. В листинге 17.5 приводится возможная реализация такого скрипта.

Листинг 17.5. Замена текстовых ссылок на гиперссылки

```
<?php
  // Весь вывод направляем в буфер
  ob_start();
?>
Форум, посвященный языку программирования PHP, можно
обнаружить по ссылке http://www.softtime.ru/forum/ .
На сайте http://www.softtime.ru можно найти множество
материалов, посвященных Web-разработке.
<?php
  // Получаем содержимое буфера, уничтожая при этом сам буфер
  $content = ob_get_clean();
  // Заменяем все текстовые ссылки гиперссылками
  $pattern = "|(http://[^\s]+)\s|is";
  $content = preg_replace($pattern, "<a href='\\1'>\\1</a> ", $content);
  // Выводим обновленное содержимое
  echo $content;
?>
```

Как видно из листинга 17.5, в начале скрипта вызывается функция `ob_start()`, переправляющая весь вывод в буфер. Функция `ob_get_clean()` помещает содержимое страницы в переменную `$content`, одновременно уничтожая буфер. Далее при помощи регулярного выражения все ссылки в переменной `$content` заменяются гиперссылками, после чего переменная переправляется в буфер обмена. Результат работы скрипта из листинга 17.5 представлен на рис. 17.3.

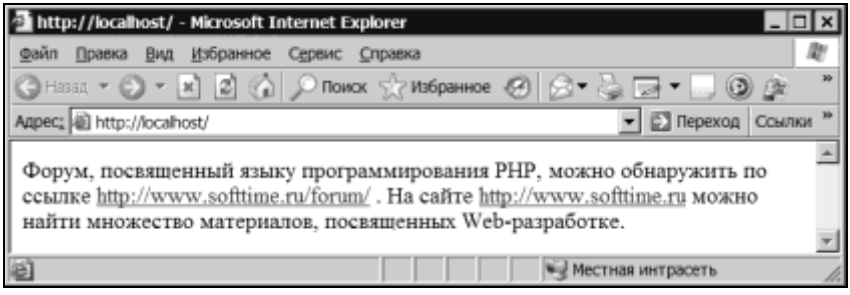


Рис. 17.3. Замена ссылок гиперссылками

17.4. Вложенная буферизация

Следует отметить, что функции управления выводом допускают создания вложенных буферов. При этом за уровнем вложения можно следить при помощи `ob_get_level()`, которая возвращает 0, если буферизация отсутствует, и значения больше 0, обозначающие уровень вложения буфера. В листинге 17.6 приводится скрипт, демонстрирующий создание вложенной системы из трех буферов.

Листинг 17.6. Вложенная буферизация

```
<?php
  // Первый буфер
  ob_start();
  echo "Буфер номер N ".ob_get_level()."<br>";

    // Второй буфер
    ob_start();
    echo "Буфер номер N ".ob_get_level()."<br>";

    // Отправляем содержимое буфера № 2 буферу № 1
    ob_end_flush();

  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

17.5. Обработчики буферов

До этого момента все преобразования буфера сводились к извлечению его содержимого и последующего вывода. Однако такие преобразования можно оформлять в виде обработчиков — функций обратного вызова, имена которых можно передавать в качестве первого аргумента функции `ob_start()`. Вернемся к скрипту из листинга 17.5, осуществляющего замену текстовых ссылок на гиперссылки. В листинге 17.7 приводится альтернативная реализация этого скрипта, использующая обработчик `ob_hyperlink_handler()`, который осуществляет преобразование содержимого буфера.

Листинг 17.7. Использование обработчика буфера

```
<?php
  // Обработчик
  function ob_hyperlink_handler($content)
  {
    // Заменяем все текстовые ссылки гиперссылками
    $pattern = "|(http://[^\s]+)\s|is";
    $content = preg_replace($pattern,"<a href='\\1'>\\1</a> ", $content);
    // Результат
    return $content;
  }

  // Весь вывод направляем в буфер с обработчиком ob_hyperlink_handler
  ob_start("ob_hyperlink_handler");
?>
Форум, посвященный языку программирования PHP, можно
обнаружить по ссылке http://www.softtime.ru/forum/.
На сайте http://www.softtime.ru можно найти множество
материалов, посвященных Web-разработке.
<?php
  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

Помимо пользовательских обработчиков, PHP предоставляет предопределенные обработчики:

- `ob_gzhandler()` — сжатие страниц перед отправкой клиенту *(см. разд. 17.7)*;
- `ob_iconv_handler()` — преобразование кодировки страницы *(см. разд. 17.8)*;
- `ob_tidyhandler()` — осуществляет попытку восстановления структуры HTML-документа, добавляя недостающие теги и закрывая незакрытые.

17.6. Список применяемых обработчиков

В реальных приложениях количество вызовов функции `ob_start()` с различными обработчиками может быть значительным. Это может приводить к запутанным ситуациям, когда заранее не известно, будет вызван тот или иной обработчик для этой страницы или нет. Определить список обработчиков можно при помощи функции `ob_list_handlers()` (листинг 17.8).

Листинг 17.8. Использование функции `ob_list_handlers()`

```
<?php
  // Подсвечиваем ссылки
  function ob_hyperlink_handler($content)
  {
    // Заменяем все текстовые ссылки гиперссылками
    $pattern = "|(http://[^\s]+)\s|is";
    $content = preg_replace($pattern,"<a href='\\1'>\\1</a> ", $content);
    // Результат
    return $content;
  }
  // Выделяем текст жирным
  function ob_bold_handler($content)
  {
    return "<b>$content</b>";
  }

  // Весь вывод направляем в буфер с обработчиком ob_hyperlink_handler
  ob_start("ob_hyperlink_handler");
  // Восстанавливаем HTML-структуру документа
  ob_start("ob_bold_handler");
?>
Форум, посвященный языку программирования PHP, можно
обнаружить по ссылке http://www.softtime.ru/forum/.
На сайте http://www.softtime.ru можно найти множество
материалов, посвященных Web-разработке.
<?php
  // Получаем список обработчиков
  $hadlers = ob_list_handlers();
  echo "<pre>";
  print_r($hadlers);
  echo "</pre>";
  // Отправляем содержимое буфера вывода клиенту
```

```
  ob_end_flush();
  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

Как видно из листинга 17.8, скрипт инициирует дважды вложенный буфер. Первому буферу назначается обработчик `ob_hyperlink_handler()`, заменяющий все ссылки гиперссылками, второму — обработчик `ob_bold_handler()`, выделяющий содержимое страницы жирным стилем. Функция `ob_list_handlers()` позволяет получить список всех обработчиков, которые еще не сработали (рис. 17.4).

Если расположить вызов функции после вызова `ob_end_flush()`, т. е. после закрытия вложенного буфера, в результирующем списке будет лишь один обработчик `ob_hyperlink_handler()`, причем дамп не будет выделенным жирным (рис. 17.5). Такое поведение связано с тем, что обработчик `ob_hyperlink_handler()` уже сработал и не фигурирует в списке обработчиков.

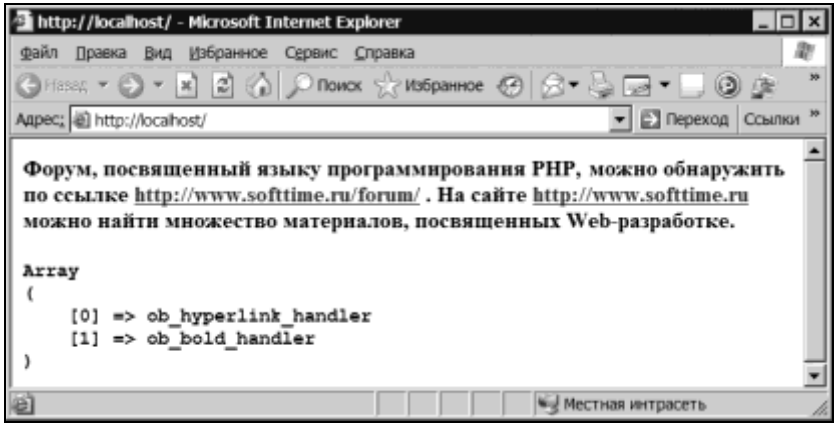


Рис. 17.4. Вывод обработчиков

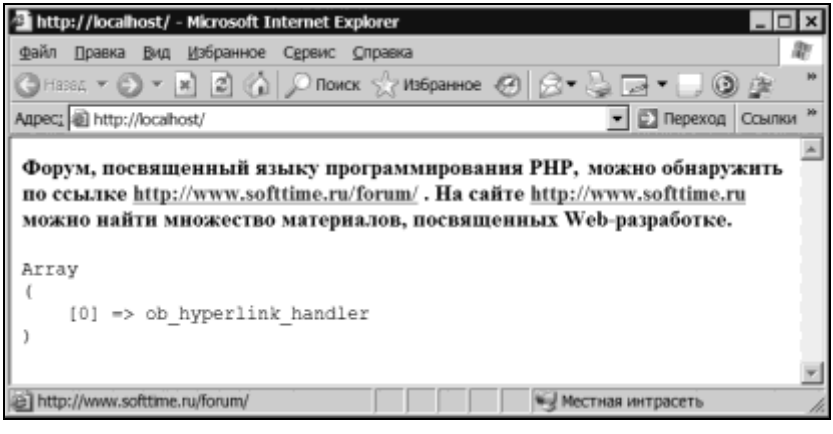


Рис. 17.5. Остался только один обработчик

17.7. Сжатие страницы

Текстовая информация достаточно удобна в обработке, однако может занимать значительный объем, по сравнению с бинарными данными. Это приводит к необходимости загружать значительные объемы "лишнего" трафика. К счастью, большинство современных браузеров поддерживает прием сжатых страниц. Для того чтобы сжать страницу перед отправкой, клиенту достаточно воспользоваться предопределенным GZ-обработчиком `ob_gzhandler()` (листинг 17.9).

Листинг 17.9. Сжатие страницы

```
<?php
  // Сжимаем трафик
  ob_start("ob_gzhandler");
?>
Форум, посвященный языку программирования PHP, можно
обнаружить по ссылке http://www.softtime.ru/forum/.
На сайте http://www.softtime.ru можно найти множество
материалов, посвященных Web-разработке.
<?php
  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

Браузер получает сжатую страницу и автоматически ее распаковывает. Такой подход несколько увеличивает нагрузку на процессор на стороне сервера и клиента, зато значительно сокращает сетевую нагрузку, которая традиционно является узким местом в работе Web-приложений.

17.8. Преобразование кодировки страницы

При создании многоязычных сайтов может потребоваться автоматическое преобразование кодировки страницы (например, текст в UTF-8 можно преобразовать в какую-либо другую кодировку, для снижения объема страницы). Такая же проблема может возникать при развертывании Web-приложений на множестве серверов с разными настройками Web-сервера Apache по умолчанию. В последнем случае потребуется подбирать кодировку и отправлять ее явно при помощи HTTP-заголовка `Content-Type` *(см. разд. 14.2)*. Еще одним решением может послужить преобразование кодировки в буфере с использованием обработчика `ob_iconv_handler()` из библиотеки iconv.

ЗАМЕЧАНИЕ

Библиотека iconv и ее функция `iconv_set_encoding()`, устанавливающая кодировки по умолчанию, более подробно рассматриваются в *разд. 9.9*.

В листинге 17.10 приводится скрипт, осуществляющий преобразование содержимого страницы из кодировки KOI8-R в cp1251.

Листинг 17.10. Преобразование кодировки страницы

```
<?php
  // Внутренняя кодировка библиотеки iconv
  iconv_set_encoding("internal_encoding", "UTF-8");
  // Кодировка входящего текста
  iconv_set_encoding("input_encoding", "KOI8-R");
  // Результирующая кодировка
  iconv_set_encoding("output_encoding", "UTF-8");

  // Создаем буфер с обработчиком, преобразующим
  // текст из кодировки KOI8-R в cp1251
  ob_start("ob_iconv_handler");
?>
Форум, посвященный языку программирования PHP, можно
обнаружить по ссылке http://www.softtime.ru/forum/.
На сайте http://www.softtime.ru можно найти множество
материалов, посвященных Web-разработке.
<?php
  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

17.9. Преобразование адресов ссылок

В заключение главы рассмотрим функции `output_add_rewrite_var()` и `output_reset_rewrite_vars()`, предназначенные соответственно для добавления и удаления GET-параметров каждой локальной гиперссылке на странице. В листинге 17.11 приводится скрипт, демонстрирующий использование функции `output_add_rewrite_var()` — каждой ссылке на странице добавляется GET-параметр `test` со значением `probe`.

> ***Замечание***
>
> GET-параметры добавляются только локальным ссылкам; абсолютным ссылкам, начинающимся с префикса http://, GET-параметры не добавляются.

Листинг 17.11. Использование функции `output_add_rewrite_var()`

```
<?php
  // Добавляем GET-параметр test со значением probe
  output_add_rewrite_var('test', 'probe');
```

```
?>
Форум, посвященный языку программирования PHP, можно обнаружить по ссылке
<a href='http://www.softtime.ru/forum/index.php'
>http://www.softtime.ru/forum/</a>.
На сайте <a href='/index.php'>http://www.softtime.ru</a> можно найти множество
материалов, посвященных Web-разработке.
```

Исходный код результирующей страницы будет выглядеть следующим образом:

```
Форум, посвященный языку программирования PHP, можно обнаружить по ссылке
<a href='http://www.softtime.ru/forum/index.php'
>http://www.softtime.ru/forum/</a>.
На сайте <a href='/index.php?test=probe'>http://www.softtime.ru</a> можно
найти множество материалов, посвященных Web-разработке.
```

Как видно из результата, дополнительный GET-параметр добавляется только к локальной ссылке и только в атрибуте `href` тега `<a>`.

Интересно отметить, что функция `ob_list_handlers()` реагирует на присутствие вызова `output_add_rewrite_var()`, сообщая, что используется обработчик `URL-Rewriter` (листинг 17.12).

Листинг 17.12. Извлекаем обработчик, соответствующий функции `output_add_rewrite_var()`

```
<?php
  // Добавляем GET-параметр test со значением probe
  output_add_rewrite_var('test', 'probe');

  // Извлекаем список обработчиков
  $arr = ob_list_handlers();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

В результате выполнения скрипта из листинга 17.12 в браузер будет выведен следующий дамп массива `$arr`:

```
Array
(
    [0] => URL-Rewriter
)
```



Глава 18

Средства шифрования

Различают два вида шифрования: необратимое шифрование (зашифрованные данные не поддаются расшифровке) и обратимое шифрование, для которого предусмотрена процедура восстановления данных.

> ***Замечание***
>
> В этой главе будут рассмотрены вопросы, связанные с шифрованием средствами PHP. Безусловно, проблема безопасности гораздо обширнее и ни в коем случае не сводится к одному только шифрованию. Однако объем книги не позволяет полностью рассмотреть все аспекты защиты Web-приложений, хотя в других главах этот вопрос неоднократно поднимается. Более подробно и систематически безопасность Web-приложений обсуждается в наших книгах "PHP. Практика создания Web-сайтов" и "Головоломки на PHP для хакера" (см. список литературы в конце книги).

18.1. Необратимое шифрование

При необратимом шифровании данные шифруются таким образом, что не подлежат обратной расшифровке. Функции, с помощью которых реализуется необратимое шифрование, называются *функциями хэширования*. При использовании таких функций создается уникальный "отпечаток" строки — причем одинаковые строки возвращают одинаковый хэш, в то время как разные строки возвращают разный хэш. На первый взгляд это может показаться странным, в действительности же такой метод шифрования используется очень часто, особенно для шифрования паролей. При аутентификации пароль, вводимый пользователем, также подвергается шифрованию, и сравниваются хэши оригинального и введенного паролей. Одинаковые пароли возвращают всегда один и тот же хэш, в то время как разные пароли возвращают всегда разные хэши. Таким образом, ни администраторы Web-приложения, ни потенциальные взломщики, получившие доступ к хэшам, не могут воспользоваться паролями напрямую. Можно попытаться подобрать пароль, однако подбор достаточно длинного пароля может потребовать значительных временных и вычислительных ресурсов.

В табл. 18.1 представлен список функций для необратимого шифрования, которые PHP предоставляет разработчикам.

Таблица 18.1. *Функции необратимого шифрования*

Функция	Описание
`md5($str [, $raw_output])`	Подвергает необратимому шифрованию строку `$str` по алгоритму MD5, в качестве результата функция возвращает 128-битный хэш-код. Если необязательный параметр `$raw_output` принимает значение `TRUE`, возвращается бинарная строка из 16 символов
`md5_file($filename [, $raw_output])`	Функция `md5_file()` аналогична `md5()`, только вместо строки в качестве первого аргумента указывается имя файла `$filename`, для которого необходимо вернуть хэш-код
`sha1($str [, $raw_output])`	Подвергает необратимому шифрованию строку `$str` по алгоритму SHA1 (Secure Hash Algorithm 1), в качестве результата функция возвращает 40-разрядное шестнадцатеричное число. Если необязательный параметр `$raw_output` принимает значение `TRUE`, возвращается бинарная строка из 20 символов
`sha1_file($filename [, $raw_output])`	Функция `sha1_file()` аналогична `sha1()`, только вместо строки в качестве первого аргумента указывается имя файла `$filename`, для которого необходимо вернуть хэш-код
`crc32($str)`	Возвращает 32-битный хэш-код для строки `$str`. Обычно этой функцией пользуются для проверки совпадения данных. С помощью этой функции удобно проверять, был ли изменен файл со времени последнего просмотра, совпадают ли исходные файл и копия, полученная через сеть и т. д.
`crypt($str [, $salt])`	Подвергает необратимому шифрованию строку `$str` в соответствии с алгоритмом DEC или альтернативным алгоритмом, если DEC не доступен. Необязательный параметр `$salt` позволяет задать ключ, на основе которого шифруется значение, если он не задается, то подбирается случайно
`str_rot13($str)`	Подвергает строку `$str` преобразованию ROT13: каждая латинская буква сдвигается на 13 позиций в алфавите, остальные символы не изменяются
`soundex($str)`	Функция возвращает хэш для строки `$str`, для слов, звучащих схоже, возвращается один и тот же хэш. Работает лишь для английского языка
`metaphone($str)`	Более точный вариант функции `soundex()`, к сожалению, также поддерживает лишь английский язык

В листинге 18.1 демонстрируется скрипт, использующий функцию `md5()` для шифрования паролей пользователя.

Листинг 18.1. Использование функции `md5()`

```
<?php
  $maks_password = "gfkjxrb99";         // сохраненный пароль пользователя
  $maks_cipher = md5($maks_password);   // сохраненный хэш-код пароля
```

```
  // Пароль пользователя, вводимый при посещении странички
  $user_password = "gfkjxrb99";
  // Хэш-код пароля пользователя, вводимый при посещении странички
  $user_cipher = md5($maks_password);
  // Если хэш-коды совпадают, то пароль верный
  if ($maks_cipher == $user_cipher)
  {
    echo "Hello, Maks";
  }
  else
  {
    echo "Введен неправильный пароль";
  }
?>
```

Таким образом, возможен следующий алгоритм авторизации пользователей. При первой регистрации пользователя сохраняется хэш-код его пароля (к примеру, в базе данных). При аутентификации хэш-код вводимого пользователем пароля сличается с сохраненным ранее хэш-кодом. Если эти отпечатки совпадают — пароль введен правильно, если нет — неправильно.

Еще одной интересной функцией является `crypt()`, которая также подвергает строку шифрованию по алгоритму DEC:

```
crypt ($str [, $salt])
```

Аргумент `$str` представляет собой строку, которую надо зашифровать. К примеру, если передать этой функции строку с паролем `"gfkjxrb99"` и аргумент `$salt`, равный `"ttt"`, то будет возвращена строка `"ttHcYCTyReZ76"`, которая не может быть дешифрована. Однако, поскольку результат работы функции строго определен в том смысле, что при вызове с одинаковыми параметрами `$str` и `$salt` функция возвращает один и тот же результат, ее можно использовать для проверки паролей (листинг 18.2).

> ***ЗАМЕЧАНИЕ***
>
> Следует отметить, что результат функции `crypt()` зависит от операционной системы. Вызов функции без аргумента `$salt` в UNIX-подобной операционной системе приводит к формату хэша, используемому в Web-сервере Apache. Однако под управлением операционной системы Windows такого формата добиться невозможно.

Листинг 18.2. Использование функции `crypt()`

```
<?php
  $maks_password = "gfkjxrb99";
  $maks_crypt = crypt($maks_password,'ttt');
  echo $maks_crypt;
  $user_password = "gfkjxrb99";
```

```
  $user_crypt = crypt($maks_password,'ttt');
  if ($maks_crypt == $maks_crypt)
  {
    echo "Пароли совпадают";
  }
  else
  {
    echo "Введен неправильный пароль";
  }
?>
```

18.2. Система регистрации и аутентификации

Создадим систему регистрации посетителей, сохраняющих информацию о них в файле text.txt следующего формата:

```
имя_пользователя::пароль::e-mail::url
```

Имя пользователя, его пароль, адрес электронной почты (e-mail) и адрес домашней страницы разделяются последовательностью `::`. Причем пароль будем подвергать MD5-шифрованию. В листинге 18.3 представлена возможная реализация системы регистрации.

Листинг 18.3. Регистрация посетителя, файл register.php

```
<table>
<form method='post'>
<tr><td>Имя:</td><td><input type='text' name='name'></td></tr>
<tr><td>Пароль:</td><td><input type='password' name='pass'></td></tr>
<tr><td>Пароль:</td><td><input type='password' name='pass_again'></td></tr>
<tr><td>e-mail:</td><td><input type='text' name='email'></td></tr>
<tr><td>URL:</td><td><input type='text' name='url'></td></tr>
<tr><td></td><td><input type='submit' value='Зарегистрировать'></td></tr>
</form>
</table>
<?php
  // Обработчик HTML-формы

  ////////////////////////////////////////////////
  // 1. Блок проверки правильности ввода данных
  ////////////////////////////////////////////////
  // Удаляем лишние пробелы
  $_POST['name'] = trim($_POST['name']);
  $_POST['pass'] = trim($_POST['pass']);
```

```
$_POST['pass_again'] = trim($_POST['pass_again']);
// Проверяем, не пустой ли суперглобальный массив $_POST
if (empty($_POST['name'])) exit();
// Проверяем, правильно ли заполнены обязательные поля
if (empty($_POST['name'])) exit('Поле "Имя" не заполнено');
if (empty($_POST['pass'])) exit('Одно из полей "Пароль" не заполнено');
if (empty($_POST['pass_again']))
                          exit('Одно из полей "Пароль" не заполнено');
if ($_POST['pass'] != $_POST['pass_again']) exit('Пароли не совпадают');
// Если введен e-mail, проверяем его на корректность
if (!empty($_POST['email']))
{
  if (!preg_match("|^[0-9a-z_]+@[0-9a-z_^\.]+\.[a-z]{2,6}$|i",
                $_POST['email']))
  {
    exit('Поле "E-mail" должно соответствовать формату
         somebody@somewhere.ru');
  }
}

////////////////////////////////////////////////
// 2. Блок проверки имени на уникальность
////////////////////////////////////////////////
// Имя файла данных
$filename = "text.txt";
// Проверяем, не было ли переданное имя зарегистрировано ранее
$arr = file($filename);
$temp = array();
foreach($arr as $line)
{
  // Разбиваем строку по разделителю ::
  $data = explode("::",$line);
  // В массив $temp помещаем имена уже зарегистрированных
  // посетителей
  $temp[] = $data[0];
}
// Проверяем, не содержится ли текущее имя в массиве имен $temp
if (!empty($temp) && in_array($_POST['name'], $temp))
{
  exit("Данное имя уже зарегистрировано, пожалуйста, выберите другое");
}
```

```
  ////////////////////////////////////////////////
  // 3. Блок регистрации пользователя
  ////////////////////////////////////////////////
  // Помещаем данные в текстовый файл
  $fd = fopen($filename, "ab");
  if (!$fd) exit("Ошибка при открытии файла данных");
  $str = $_POST['name']."::".
         md5($_POST['pass'])."::".
         $_POST['email']."::".
         $_POST['url']."\r\n";
  fwrite($fd,$str);
  fclose($fd);
  // Осуществляем перезагрузку страницы, чтобы сбросить POST-данные
  echo "<HTML><HEAD>
         <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
        </HEAD></HTML>";
?>
```

Как видно из листинга 18.3, HTML-форма для регистрации содержит два поля типа `password` для редактирования пароля, два поля типа `text` для редактирования адреса электронной почты (e-mail) и адреса домашней страницы (URL). Кроме этого, форма имеет три скрытых поля (`hidden`) для передачи имени пользователя, его пароля и вспомогательной переменной `edit`, сообщающей, что пользователь осуществляет редактирование данных при помощи второй HTML-формы.

В листинге 18.4 приводится реализация системы аутентификации посетителей, проверяющей, зарегистрирован ли посетитель с таким именем и соответствует ли имени введенный пароль.

Листинг 18.4. Система аутентификации

```
<?php
  // Имя файла данных
  $filename = "text.txt";
  // Проверяем, не пусто ли содержимое массива $_POST.
  // Если это так, то выводим форму для авторизации
  if (empty($_POST))
  {
    ?>
    <table>
      <form method='post'>
      <tr>
        <td>Имя:</td>
        <td><input type='text' name='name'></td>
```

```
    </tr>
    <tr>
      <td>Пароль:</td>
      <td><input type='password' name='pass'></td>
    </tr>
    <tr>
      <td> </td>
      <td><input type='submit' value='Войти'></td>
    </tr>
    </form>
 </table>
 <?php
}
// В противном случае, если POST-данные переданы - обрабатываем их
else
{
  // Проверяем корректность введенного имени и пароля
  $arr = file($filename);
  $i = 0;
  $temp = array();
  foreach($arr as $line)
  {
    // Разбиваем строку по разделителю ::
    $data = explode("::",$line);
    // В массив $temp помещаем имена и пароли
    // зарегистрированных посетителей
    $temp['name'][$i]     = $data[0];
    $temp['password'][$i] = $data[1];
    $temp['email'][$i]    = $data[2];
    $temp['url'][$i]      = trim($data[3]);
    // Увеличиваем счетчик
    $i++;
  }
  // Если в массиве $temp['name'] нет введенного
  // логина - останавливаем работу скрипта
  if (!in_array($_POST['name'], $temp['name']))
  {
    exit("Пользователь с таким именем не зарегистрирован");
  }
  // Если пользователь с именем $_POST['name'] обнаружен,
  // проверяем правильность введенного пароля
  $index = array_search($_POST['name'], $temp['name']);
```

```
    if (md5($_POST['pass']) != $temp['password'][$index])
    {
      exit("Пароль не соответствует логину");
    }
    // Доступ к закрытым данным
    ...
  }
?>
```

18.3. Обратимое шифрование

При симметричном шифровании строки шифруются с помощью ключа, который известен как отправителю, так и получателю. Алгоритмы симметричного шифрования в PHP реализованы в библиотеке mcrypt, и ознакомиться с полным набором функций и алгоритмов этой библиотеки можно на сайте **http://www.php.net**. Функции mcrypt работают только, если подключена библиотека mcrypt.

> ***ЗАМЕЧАНИЕ***
>
> Как и любое расширение, библиотека mcrypt отключена по умолчанию в PHP, начиная с версии 5. Для того чтобы ее подключить, необходимо убрать комментарий напротив строки `extension=php_mcrypt.dll` в конфигурационном файле php.ini. Кроме этого в системный каталог C:\Windows\sysyem32 необходимо скопировать дополнительную библиотеку libmcrypt.dll, которая не входит в состав дистрибутива PHP, и ее следует загрузить из сети, например, по ссылке **http://www.softtime.ru/files/libmcrypt.dll**.

В листинге 18.5 приведен пример, в котором строка зашифровывается и расшифровывается при помощи алгоритма 3DES (Triple-DES).

Листинг 18.5. Использование симметричного шифрования

```
<?php
  // Зашифровываем пароль
  $user_password = "gfkjxrb99";
  $key = "Это секретный ключ";
  // Шифруем пароль с использованием ключа $key
  $user_crypt = mcrypt_ecb(MCRYPT_3DES, $key, $maks_password, MCRYPT_ENCRYPT);
  echo "Зашифрованный пароль - ".$user_crypt;
  // Расшифровываем пароль
  $user_crypt= mcrypt_ecb(MCRYPT_3DES, $key, $user_crypt, MCRYPT_DECRYPT);
  echo "Расшифрованный пароль - ".$user_crypt;
?>
```

В листинге 18.5 пароль, размещенный в переменной `$user_password`, сначала шифруется функцией `mcrypt_ecb()` с применением ключа `$key`, а затем проводится его обратная дешифрация.



Глава 19

Сетевое взаимодействие. Сокеты и CURL

Иногда PHP-скрипты выполняются не только на сервере, но и на стороне клиента (которым может выступать другой сервер), обращаясь к удаленным страницам. PHP предоставляет широкие возможности для работы скриптов в таком качестве.

Сокетом обычно называют сетевое соединение, которое ведет себя подобно файлу: его можно открывать и закрывать, писать и читать из него. Большинство разработчиков хорошо понимает принципы работы с файлами, именно поэтому сокеты получили широкое распространение.

Сетевое программирование — гораздо более сложная задача, чем обращение к обычному файлу, поэтому взаимодействие с сокетами несколько сложнее. Впрочем, PHP предоставляет множество функций, позволяющих работать с удаленными файлами как с локальными.

В данной главе будут рассмотрены два основных расширения: сокеты и библиотека CURL. В конце главы рассматриваются средства для работы с доменными именами и IP-адресами.

19.1. Обращение к удаленным страницам

Самый простой способ реализации такого обращения — воспользоваться файловыми функциями `file_get_contents()` и `file_put_contents()`, как это демонстрируется в листинге 19.1.

> ***Замечание***
>
> Содержимое листинга 19.1 можно сократить до одной строки, если воспользоваться функцией `copy()`, которая также поддерживает обращение к удаленному источнику. Более подробно файловые функции рассматриваются в *главе 13*.

Листинг 19.1. Загрузка страницы с удаленного хоста (вариант 1)

```
<?php
  // Загружаем страницу с удаленного хоста
  $content = file_get_contents("http://www.softtime.ru/");
  // Сохраняем содержимое страницы в файле
  file_put_contents("url.txt", $content);
?>
```

В результате работы скрипта из листинга 19.1 содержимое индексной страницы сайта **http://www.softtime.ru** будет сохранено в файле url.txt. Функция `file_get_contents()` является достаточно удобной и возвращает содержимое локального или удаленного файла в виде строки. Функция появилась в PHP, только начиная с версии 4.3.0. Для более ранних версий необходимо использовать связку функций `fopen()` и `fgets()` (листинг 19.2).

Листинг 19.2. Загрузка страницы с удаленного хоста (вариант 2)

```
<?php
  // Получаем дескриптор удаленной страницы
  $fd = fopen("http://www.softtime.ru/", "r");
  if (!$fd) exit("Запрашиваемая страница не найдена");
  $content = "";
  // Чтение содержимого файла в переменную text
  while (!feof ($fd))
  {
    $content .= fgets($fd, 1024);
  }
  // Закрыть открытый указатель файла
  fclose ($fd);

  // Сохраняем содержимое страницы в файле
  $fd = fopen("url.txt", "w");
  fwrite($fd, $content);
  fclose($fd);
?>
```

В данном случае загрузка страницы осуществляется при помощи функции `fgets()`, которая построчно извлекает из удаленного файла данные. Второй параметр указывает максимальное количество данных, считываемых за раз. Как правило, строка не превышает 1024 символа, поэтому данный параметр можно оставить без изменений. Разумеется, загрузка файла во втором случае протекает медленнее, т. к. скрипту необходимо выполнять PHP-цикл, выполняемый интерпретатором, а не бинарный код функции `file_get_contents()`, созданной при помощи языка C. Однако второй подход имеет преимущества при загрузке объемных файлов: дело в том, что для PHP-скрипта отводится строго ограниченный объем памяти, который задается в конфигурационном файле php.ini директивой `memory_limit`. Данная директива обычно принимает значение 8 или 16 Мбайт. Этот объем отводится как под сам PHP-скрипт, так и под все используемые в нем переменные и массивы. Таким образом, если разработчик имеет дело с файлом размером 10 Мбайт и содержимое этого файла будет полностью считано в переменную, над такой переменной невозможно будет выполнить даже операции замены, т. к. для этого потребуется 20 Мбайт. Подход, используемый в листинге 19.2, предоставляет возможность обрабатывать файл построчно, не загружая полностью его содержимое в область памяти, отводимой для скрипта.

Файловые функции достаточно эффективны, однако удаленный сервер может определить, что к нему обращается не посетитель посредством браузера, а скрипт, потребовав установить cookies, проверить переменную окружения `USER_AGENT`, через которую клиент передает тип и версию браузера, операционной системы и т. п. Разумеется, стандартные файловые системы не предоставляют такой информации — для полной эмуляции работы браузера необходимо воспользоваться либо сокетами, либо специальной библиотекой CURL.

19.2. Сокеты

Файловые функции не всегда удобны, т. к. удаленный хост может быть оптимизирован для работы с браузером, который должен поддерживать все особенности HTTP-протокола. В этом случае прибегают к сокетам. Подключение к удаленному серверу можно осуществить с помощью функции `fsockopen()`, которая имеет следующий синтаксис:

```
fsockopen($target, $port [,$errno [,$errstr [,$timeout]]])
```

Функция принимает пять параметров, первый из которых содержит адрес соединения, а второй — номер порта. В случае удачи функция возвращает дескриптор соединения, в противном случае — значение `FALSE`. Если при этом функции переданы два необязательных параметра `$errno` и `$errstr`, в них размещается код и текстовое описание ошибки соответственно. Пятый, также необязательный параметр — значение тайм-аута, определяющего максимальное время ожидания ответа сервера в секундах. При успешной установке соединения функция возвращает дескриптор соединения, при неудаче — `FALSE`.

> ***ЗАМЕЧАНИЕ***
>
> Для того чтобы функция `fsockopen()` была доступна из PHP-скрипта, в конфигурационном файле php.ini необходимо подключить расширение php_sockets.dll, сняв комментарий (точка с запятой `;`) напротив соответствующей директивы `extension`.

Пример работы с функцией `fsockopen()` приведен в листинге 19.3, где происходит загрузка главной страницы портала **http://www.php.net**.

Листинг 19.3. Пример использования функции `fsockopen()`

```
<?php
  function get_content($hostname, $path)
  {
    $line = "";
    // Устанавливаем соединение, имя которого передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос для передачи его серверу
```

```
      $headers = "GET $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      // Получаем ответ
      while (!feof($fp))
      {
        $line .= fgets($fp, 1024);
      }
      fclose($fp);
    }
    return $line;
  }
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока вся страница не загружена, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  echo get_content($hostname, $path);
?>
```

При работе с сокетами скрипт вынужден брать на себя всю черновую работу по отправке серверу HTTP, получению и обработке ответов на них. В приведенном выше скрипте обращение к `fsockopen()` оформлено в виде функции `get_content()`, которая получает два параметра: `$hostname` — имя сервера, с которым устанавливается соединение, и `$path` — путь к странице относительно имени сервера.

> ***ЗАМЕЧАНИЕ***
>
> Следует помнить, что при работе с функцией `fsockopen()` имя сервера не должно содержать префикса http://, указывающего на тип протокола и стандартный порт 80. При работе с сокетами реализация протокола ложится на плечи программиста, а порт указывается во втором параметре функции `fsockopen()`, поэтому такая детализация не требуется.

После установки соединения с сервером в переменной `$headers` формируется HTTP-запрос серверу по методу GET. При подстановке значений всех переменных серверу будут отправлены соответствующие заголовки (листинг 19.4).

Листинг 19.4. Заголовки, отправляемые серверу *www.php.net*

```
GET / HTTP/1.1\r\n
Host: www.php.net\r\n
Connection: Close\r\n\r\n
```

После отправки запроса функцией `fgets()` производится чтение ответа из сокета блоками по 1024 байта до тех пор, пока не встретится символ конца файла, определяемый функцией `feof()`. Затем результат возвращается и выводится в окно браузера.

> ***ЗАМЕЧАНИЕ***
>
> Из-за ошибки в реализации обращение через сокеты по протоколу HTTP/1.1 в операционной системе Windows выполняется достаточно медленно, поэтому в ней зачастую удобнее использовать протокол HTTP/1.0.

В первой строке листинга 19.4 у сервера запрашивается индексный файл корневого каталога сервера (`/`) методом GET по протоколу HTTP/1.1. HTTP-заголовок `Host` является обязательным, он необходим HTTP-серверу для определения сайта, запрашиваемого клиентом (на тот случай, если к одному IP-адресу привязано несколько доменных имен). Последний HTTP-заголовок `Connection` требует от сервера закрыть соединение после передачи данных.

> ***ЗАМЕЧАНИЕ***
>
> Вместо / можно передать адрес страницы на сервере, например, /downloads.php, в результате чего будет загружена другая страница сайта — именно так и действуют браузеры. При переходе пользователей по ссылке они извлекают часть адреса после доменного имени и передают HTTP-запрос, подставив эту часть после ключевого слова GET.

Эти три строки эквивалентны запросу в окне браузера **http://www.php.net/**. Результат работы скрипта из листинга 19.3 приведен на рис. 19.1.

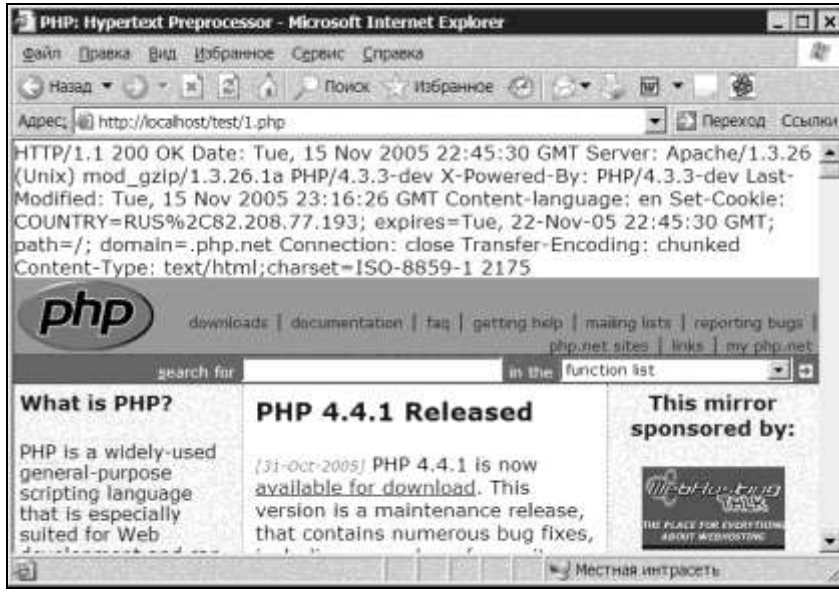


Рис. 19.1. Просмотр портала **www.php.net** с локального хоста

В начале страницы идут HTTP-заголовки, которые браузер обычно скрывает от пользователя, после чего начинается тело страницы, которое интерпретируется браузером.

Избавиться от HTTP-заголовков можно при помощи функции `strstr()`, которая возвращает часть строки (первый параметр), начиная с первого вхождения подстроки (второй параметр) (листинг 19.5).

Листинг 19.5. Удаление HTTP-заголовков

```
<?php
  ...
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока вся страница не будет загружена, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  $content = get_content($hostname, $path);
  echo strstr($content, '<');
?>
```

Результат работы скрипта из листинга 19.5 представлен на рис. 19.2. Несмотря на то, что скрипт загружает страницу с портала **http://www.php.net**, адресная строка указывает на скрипт, расположенный на локальном хосте. Таким образом, в качестве браузера выступает PHP-скрипт.



Рис. 19.2. Удаление "мешающих" HTTP-заголовков

19.3. Библиотека CURL

Помимо сокетов, обеспечивающих низкоуровневое обращение к серверу, PHP располагает специальным расширением CURL (Client URL Library). В листинге 19.6 представлено альтернативное решение задачи с использованием расширения CURL, полностью эквивалентное решению из листинга 19.5.

Замечание

Для того чтобы функции библиотеки CURL были доступны из PHP-скрипта, в конфигурационном файле php.ini необходимо подключить расширение php_curl.dll, сняв комментарий (точка с запятой `;`) напротив соответствующей директивы `extension`. Помимо этого необходимо скопировать библиотеки ssleay32.dll и libeay32.dll из каталога, где расположен PHP, в папку, прописанную в переменной окружения `PATH`, например, в C:\Windows\system32.

В случае расширения CURL не требуется удаление HTTP-заголовков, возвращаемых сервером, т. к. библиотека их удаляет по умолчанию. Однако CURL можно настроить на выдачу HTTP-заголовков, передаваемых сервером, если установить при помощи функции `curl_setopt()` ненулевое значение параметра `CURLOPT_HEADER` (см. табл. 3.1).

Листинг 19.6. Загрузка страницы с использованием расширения CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://www.php.net");
  // Устанавливаем параметры соединения
  curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  // Получаем содержимое страницы
  $content = curl_exec($curl);
  // Закрываем CURL соединение
  curl_close($curl);

  echo $content;
?>
```

При помощи функции `curl_init()` задается адрес удаленного сервера и путь к файлу на нем. В отличие от функции `fsockopen()`, необходимо задавать адрес полностью, включая префикс http://, т. к. расширение CURL позволяет работать с несколькими видами протоколов (HTTP, HTTPS, FTP). Если соединение с указанным сервером происходит успешно, функция `curl_init()` возвращает дескриптор соединения, который используется в качестве параметра во всех остальных функциях библиотеки.

Функция `curl_setopt()` позволяет задать параметры текущего соединения и имеет следующий синтаксис:

`curl_setopt($`*`curl`*`, $`*`option`*`, $`*`value`*`)`

Функция устанавливает для соединения с дескриптором `$`*`curl`* параметр `$`*`option`* со значением `$value`. В качестве параметра `$`*`option`* используются константы, определенные в табл. 19.1—19.5. Если параметр успешно установлен, функция возвращает `TRUE`, в противном случае возвращается значение `FALSE`.

Запрос выполняет функция `curl_exec()`. В листинге 19.6 содержимое запрашиваемой страницы возвращается в виде строки `$content` (такое поведение определяется константой `CURLOPT_RETURNTRANSFER`, установленной ранее при помощи функции `curl_setopt()`).

В завершение функция `curl_exec()` закрывает установленное ранее CURL-соединение.

Таблица 19.1. *Логические параметры CURL-соединения*

Параметр	Описание
`CURLOPT_AUTOREFERER`	При установке этого параметра в `TRUE`, если осуществляется следование HTTP-заголовку `Location`, HTTP-заголовок `Referer` устанавливается автоматически
`CURLOPT_COOKIESESSION`	При установке этого параметра в `TRUE` CURL игнорирует все сессионные cookie (хранящиеся в оперативной памяти и лишь до момента закрытия браузера)
`CURLOPT_CRLF`	При установке этого параметра в `TRUE` UNIX-переводы строк `\n` автоматически преобразуются к виду `\r\n`
`CURLOPT_DNS_USE_GLOBAL_CACHE`	При установке этого параметра в `TRUE` используется глобальный DNS-кэш
`CURLOPT_FAILONERROR`	При установке этого параметра в `TRUE` получение HTTP-кода более 300 считается ошибкой
`CURLOPT_FILETIME`	При установке этого параметра в `TRUE` CURL пытается получить дату последней модификации загружаемого документа
`CURLOPT_FOLLOWLOCATION`	При установке этого параметра в `TRUE` при получении HTTP заголовка `Location` будет происходить перенаправление на указанный этим заголовком URL (это действие выполняется рекурсивно, для каждого полученного заголовка `Location`)
`CURLOPT_FORBID_REUSE`	При установке этого параметра в `TRUE` CURL-соединение завершается сразу после выполнения функции `curl_exec()`; повторное использование дескриптора соединения не допускается
`CURLOPT_FRESH_CONNECT`	При установке этого параметра в `TRUE` для повторной операции `curl_exec()` CURL требует создания нового соединения, не используя кэшированное
`CURLOPT_FTPAPPEND`	При установке этого параметра в `TRUE` данные будут добавляться к файлу на FTP сервере, в противном случае файл будет перезаписан
`CURLOPT_FTPASCII`	При установке этого параметра в `TRUE` данные с FTP-сервера будут передаваться не в бинарном, а в текстовом (ASCII) виде
`CURLOPT_FTPLISTONLY`	При установке этого параметра в `TRUE` будет получен список файлов в текущем каталоге FTP-сервера
`CURLOPT_HEADER`	При установке этого параметра в `TRUE` результат будет включать полученные HTTP-заголовки

Таблица 19.1 *(окончание)*

Параметр	Описание
CURLOPT_HTTPPROXYTUNNEL	При установке этого параметра в TRUE данные будут передаваться через прокси-сервер
CURLOPT_MUTE	При установке этого параметра в true все сообщения CURL будут подавляться
CURLOPT_NETRC	При установке этого параметра в TRUE будет осуществлена попытка найти имя пользователя и пароль к удаленному серверу в файле ~/netrc (где ~ — домашний каталог)
CURLOPT_NOBODY	При установке этого параметра в TRUE результат не будет включать документ. Часто используется для того, чтобы получить только HTTP-заголовки
CURLOPT_NOPROGRESS	При установке этого параметра в TRUE не будет выводиться индикатор прогресса операции (по умолчанию установлен)
CURLOPT_POST	При установке этого параметра в TRUE отправляется POST-запрос типа application/x-www-form-urlencoded
CURLOPT_PUT	При установке этого параметра в TRUE будет производиться закачка файла методом PUT протокола HTTP. Файл задается параметрами CURLOPT_INFILE и CURLOPT_INFILESIZE. Впрочем, метод PUT на большинстве серверов запрещен к использованию
CURLOPT_RETURNTRANSFER	При установке этого параметра в TRUE CURL будет возвращать результат, а не выводить его
CURLOPT_SSL_VERIFYPEER	При установке параметра в FALSE запрещается проверка сертификата удаленного сервера. Дополнительные сертификаты можно задать с помощью параметра CURLOPT_CAINFO. Можно также указать путь к файлам сертификатов в параметре CURLOPT_CAPATH. Если CURLOPT_SSL_VERIFYPEER установлен в 0, возможно, потребуется установить в 1 или 0 также CURLOPT_SSL_VERIFYHOST (по умолчанию 2)
CURLOPT_UNRESTRICTED_AUTH	При установке параметра в TRUE имя пользователя и пароль сохраняются и передаются серверу при следовании HTTP-заголовку Location, даже если осуществляется переход на другой сайт
CURLOPT_UPLOAD	При установке этого параметра в TRUE производится закачка файла на удаленный сервер
CURLOPT_VERBOSE	При установке этого параметра в TRUE выводятся подробные сообщения обо всех производимых действиях

Таблица 19.2. *Целочисленные параметры CURL-соединения*

Параметр	Описание
CURLOPT_BUFFERSIZE	Размер буфера, используемого для чтения документа с удаленного сервера
CURLOPT_CONNECTTIMEOUT	Количество секунд перед попыткой соединения с сервером. По умолчанию имеет значение 0
CURLOPT_DNS_CACHE_TIMEOUT	Количество секунд, в течение которых хранится запись в DNS-кэше. По умолчанию имеет значение 120 (2 минуты)
CURLOPT_HTTP_VERSION	Версия HTTP-протокола; допустимы три значения: CURL_HTTP_VERSION_NONE (версия выбирается автоматически), CURL_HTTP_VERSION_1_0 (используется HTTP 1.0), CURL_HTTP_VERSION_1_1 (используется HTTP 1.1)
CURLOPT_HTTPAUTH	Метод (методы) HTTP-аутентификации; допустимые значения: CURLAUTH_BASIC, CURLAUTH_DIGEST, CURLAUTH_GSSNEGOTIATE, CURLAUTH_NTLM, CURLAUTH_ANY и CURLAUTH_ANYSAFE
CURLOPT_INFILESIZE	Размер файла при его загрузке на удаленный сервер
CURLOPT_LOW_SPEED_LIMIT	Минимальная скорость передачи в байтах в секунду. Если в течение периода времени, заданного параметром CURLOPT_LOW_SPEED_TIME, скорость передачи станет меньше этого значения, операция будет прервана
CURLOPT_LOW_SPEED_TIME	Время в секундах, в течение которого скорость передачи должна быть не ниже, чем CURLOPT_LOW_SPEED_LIMIT, чтобы операция была признана слишком медленной и прервана
CURLOPT_MAXCONNECTS	Максимальное количество постоянных соединений
CURLOPT_MAXREDIRS	Максимальное количество переходов по заголовку Location. Параметр используется совместно с CURLOPT_FOLLOWLOCATION
CURLOPT_PORT	Альтернативный номер порта для соединения
CURLOPT_PROXYAUTH	Метод (методы) аутентификации на прокси-сервере; принимает те же значения, что и параметр CURLOPT_HTTPAUTH
CURLOPT_PROXYPORT	Номер порта для соединения с прокси-сервером; используется совместно с CURLOPT_PROXY
CURLOPT_PROXYTYPE	Тип прокси-сервера: устанавливать соединение через HTTP-протокол (CURLPROXY_HTTP) или через сокет (CURLPROXY_SOCKS5)
CURLOPT_RESUME_FROM	Задает в файле позицию байта, с которого начнется передача данных

***Таблица 19.2** (окончание)*

Параметр	Описание
CURLOPT_SSL_VERIFYHOST	Строгость проверки имени, указанного в сертификате удаленного сервера (при установке SSL-соединения): 1 — предполагает только проверку существования имени; 2 — кроме того, и проверку соответствия имени хоста
CURLOPT_SSLVERSION	Версия SSL; допустимые значения 2 и 3. По умолчанию версия SSL определяется автоматически, но в некоторых случаях требуется явное указание
CURLOPT_TIMECONDITION	Способ интерпретации значения параметра CURLOPT_TIMEVALUE. Допустимые значения: TIMECOND_IFMODSINCE и TIMECOND_ISUNMODSINCE. Применяется только для протокола HTTP
CURLOPT_TIMEOUT	Максимальное время выполнения операции в секундах
CURLOPT_TIMEVALUE	Время в секундах, прошедшее с полуночи 1 января 1970 г. Значение будет использовано в соответствии со значением параметра CURLOPT_TIMECONDITION (по умолчанию TIMECOND_IFMODSINCE)

***Таблица 19.3.** Строковые параметры CURL-соединения*

Параметр	Описание
CURLOPT_CAINFO	Имя файла, содержащего один или более сертификатов, которые будут использованы при проверке подлинности удаленного сервера. Имеет значение только совместно с параметром CURLOPT_SSL_VERIFYPEER
CURLOPT_CAPATH	Путь к каталогу с сертификатами. Имеет значение только совместно с параметром CURLOPT_SSL_VERIFYPEER
CURLOPT_COOKIE	Содержимое HTTP-заголовка Cookie. Для установки нескольких значений cookie можно использовать несколько вызовов функции curl_setopt()
CURLOPT_COOKIEFILE	Имя файла, содержащего данные cookie. Данные могут быть либо в формате Netscape, либо HTTP-заголовков
CURLOPT_COOKIEJAR	Имя файла, в который сохраняются несессионные cookie, доступные при следующем сеансе соединения с сервером
CURLOPT_CUSTOMREQUEST	Метод, который будет использован в HTTP-запросе вместо GET или HEAD. Используется для отправки запросов DELETE или других, редко используемых. Допустимыми значениями являются GET, POST и т. д.
CURLOPT_ENCODING	Значение HTTP-заголовка Accept-Encoding, задающего метод сжатия HTTP-ответа. Допустимые значения: identity, deflate, gzip

***Таблица 19.3** (продолжение)*

Параметр	Описание
CURLOPT_FTPPORT	Значение IP-адреса для команды PORT протокола FTP. Допустимые значения: IP-адрес, имя хоста, имя сетевого интерфейса (под UNIX), или просто — для использования IP-адреса по умолчанию
CURLOPT_INTERFACE	Имя используемого сетевого интерфейса. Может быть именем интерфейса, IP-адресом или именем хоста
CURLOPT_KRB4LEVEL	Уровень безопасности KRB4 (Kerberos 4) для протокола FTP. Допустимы следующие значения (в порядке возрастания безопасности): clear, safe, confidential, private. Если переданное значение не входит в этот список, используется private. Установка параметра в NULL запрещает безопасность KRB4
CURLOPT_POSTFIELDS	Строка данных для POST-запроса
CURLOPT_PROXY	Имя прокси-сервера, через который будут направляться запросы
CURLOPT_PROXYUSERPWD	Строка с именем пользователя и паролем к прокси-серверу HTTP в виде [username]:[password]
CURLOPT_RANDOM_FILE	Путь к файлу, используемому генератором случайных чисел для работы протокола SSL
CURLOPT_RANGE	Координаты фрагмента загружаемого файла в формате "X-Y" (вместо X и Y указываются позиции байта в файле). Одна из координат может быть опущена, например: "X-". Протокол HTTP также поддерживает передачу нескольких фрагментов файла, это задается в виде "X-Y,N-M"
CURLOPT_REFERER	Значение HTTP-заголовка Referer
CURLOPT_SSL_CIPHER_LIST	Список шифров, используемых SSL, например: RC4-SHA, TLSv1
CURLOPT_SSLCERT	Имя файла с SSL-сертификатом в формате PEM
CURLOPT_SSLCERTPASSWD	Пароль к файлу SSL-сертификата, заданному параметром CURLOPT_SSLCERT
CURLOPT_SSLCERTTYPE	Формат сертификата. Допускаются следующие форматы: PEM (по умолчанию), DER и ENG
CURLOPT_SSLKEY	Имя файла, содержащего закрытый SSL-ключ
CURLOPT_SSLKEYPASSWD	Секретный пароль, необходимый для использования закрытого SSL-ключа
CURLOPT_SSLKEYTYPE	Формат закрытого SSL-ключа. Допускаются следующие форматы: PEM (по умолчанию), DER и ENG
CURLOPT_URL	URL, с которым будет производиться операция. Значение этого параметра также может быть задано при вызове функции curl_init()

Таблица 19.3 *(окончание)*

Параметр	Описание
CURLOPT_USERAGENT	Задает значение HTTP-заголовка User-Agent
CURLOPT_USERPWD	Строка с именем пользователя и паролем в виде [username]:[password]

Таблица 19.4. *Параметры CURL-соединения, являющиеся массивами*

Параметр	Описание
CURLOPT_HTTP200ALIASES	Массив с HTTP-заголовками группы 200 (успешно выполненный запрос)
CURLOPT_HTTPHEADER	Массив со всеми HTTP-заголовками
CURLOPT_POSTQUOTE	Массив с FTP-командами, которые будут выполнены после выполнения основного запроса
CURLOPT_QUOTE	Массив с FTP-командами, которые будут выполнены перед выполнением основного запроса

Таблица 19.5. *Параметры CURL-соединения, использующие в качестве значений дескриптор файла, возвращаемого функцией* `fopen()`

Параметр	Данные, хранимые в файле
CURLOPT_FILE	Результат операции
CURLOPT_INFILE	Данные для передачи
CURLOPT_WRITEHEADER	Полученные HTTP-заголовки
CURLOPT_STDERR	Сообщения об ошибках

Если не задан ни один из параметров CURLOPT_RETURNTRANSFER, CURLOPT_FILE или CURLOPT_WRITEHEADER, функция curl_exec() по умолчанию выводит результат непосредственно в окно браузера. Так как результат запроса необходимо вывести непосредственно в браузер, листинг 19.6 можно переписать более компактно, не устанавливая никаких параметров (листинг 19.7).

Листинг 19.7. Альтернативное решение

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://www.php.net");
  // Получаем содержимое страницы
  echo curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

19.4. Получение HTTP-заголовков с сервера

Часто при работе с данными удаленного хоста не требуется тело ответа, представляющее собой объемную HTML-страницу или файл: достаточно получить HTTP-заголовки, сообщающие полезную информацию о сервере и его требованиях для работы с его содержимым (установка cookie, сессий или требования произвести аутентификацию).

Модифицируем функцию из листинга 19.3 для получения лишь заголовков HTTP-ответа (листинг 19.8).

Листинг 19.8. Загружаем только заголовки HTTP-ответа

```
<?php
  // Функция получения HTTP-заголовков
  function get_content($hostname, $path)
  {
    $line = "";
    // Устанавливаем соединение, имя которого передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос серверу
      $headers = "GET $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      $end = $false;
      // Получаем ответ
      while (!$end)
      {
        $line = fgets($fp, 1024);
        if (trim($line) == "") $end = true;
        else $out[] = $line;
      }
      fclose($fp);
    }
    return $out;
  }
  $hostname = "www.php.net";
  $path = "/";
```

```
  // Устанавливаем большее время работы скрипта.
  // Пока страница не будет загружена целиком, она не будет отображена
  set_time_limit(180);
  // Вызываем функцию
  $out = get_content($hostname, $path);
  // Выводим содержимое массива
  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

Результат работы функции может выглядеть следующим образом:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Sat, 14 Jul 2007 13:03:07 GMT
    [2] => Server: Apache/1.3.37 (Unix) PHP/5.2.1
    [3] => X-Powered-By: PHP/5.2.1
    [4] => Last-Modified: Sat, 14 Jul 2007 12:37:24 GMT
    [5] => Content-language: en
    [6] => Set-Cookie: COUNTRY=RUS%2C82.208.77.226; expires=Sat, 21-Jul-2007
13:03:07 GMT; path=/; domain=.php.net
    [7] => Connection: close
    [8] => Transfer-Encoding: chunked
    [9] => Content-Type: text/html;charset=utf-8
)
```

Первая строка является стандартным ответом сервера о том, что запрос успешно обработан (код ответа 200). Если запрашиваемый ресурс не существует, то будет возвращен код ответа 404 (HTTP/1.1 404 Not Found).

Второй заголовок сообщает время формирования документа на сервере, необходимое для кэширования, которое рассматривается далее.

Третий заголовок сообщает тип и версию Web-сервера. Как можно видеть, портал **http://www.php.net** работает на сервере под управлением UNIX, используя в качестве Web-сервера Apache 1.3.37, а также PHP версии 5.2.1, которая на момент написания книги находилась в разработке.

Четвертый заголовок сообщает, что страница сгенерирована при помощи PHP версии 5.2.1.

Пятый заголовок сообщает о дате последней модификации страницы, впрочем, при динамическом формировании содержимого сайта он не актуален.

Шестой заголовок сообщает, что браузеру передаются данные на английском языке.

Седьмой заголовок устанавливает cookie с именем `COUNTRY` и значением `"RUS,82.208.77.193"`, содержащим страну пользователя и его IP-адрес сроком

на неделю для домена `.php.net`. Данная информация необходима для раздела **downloads** сайта **http://www.php.net**, в котором посетителю предлагается ближайший к нему сервер.

Восьмой заголовок передает клиенту просьбу закрыть соединение после получения ответа.

Девятый заголовок `Transfer-Encoding` (имеющий значение `chuncked`) указывает получателю, что ответ разбит на фрагменты.

Десятый заголовок `Content-Type` указывает тип загружаемого документа (`text/html`) и его кодировку (`charset=utf-8`).

В листинге 19.8 для получения заголовков использован запрос GET, однако для этой цели в протоколе HTTP предусмотрен специальный метод HEAD. Таким образом, скрипт из листинга 19.8 можно переписать более простым способом (листинг 19.9).

> ***Замечание***
>
> Метод HEAD может быть запрещен на HTTP-сервере, поэтому иногда целесообразно воспользоваться методом GET, отсекая тело документа, как это представлено в листинге 19.8.

Листинг 19.9. Использование метода HEAD

```
<?php
  // Функция получения HTTP-заголовков
  function get_content($hostname, $path)
  {
    $line = "";
    // Устанавливаем соединение, имя которого передано в параметре $hostname
    $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
    // Проверяем успешность установки соединения
    if (!$fp) echo "$errstr ($errno)<br />\n";
    else
    {
      // Формируем HTTP-запрос серверу
      $headers = "HEAD $path HTTP/1.1\r\n";
      $headers .= "Host: $hostname\r\n";
      $headers .= "Connection: Close\r\n\r\n";
      // Отправляем HTTP-запрос серверу
      fwrite($fp, $headers);
      // Получаем ответ
      while (!feof($fp))
      {
        $out[] = fgets($fp, 1024);
      }
```

```
      fclose($fp);
    }
    return $out;
  }
  $hostname = "www.php.net";
  $path = "/";
  // Устанавливаем большее время работы скрипта.
  // Пока страница не будет загружена целиком, она не отобразится
  set_time_limit(180);
  // Вызываем функцию
  $out = get_content($hostname, $path);
  // Выводим содержимое массива
  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

Еще более компактно рассматриваемую задачу можно решить с использованием расширения CURL, установив при помощи функции `curl_setopt()` параметры `CURLOPT_HEADER` и `CURLOPT_NOBODY`, первый из которых требует включения в результат HTTP-заголовков, а второй — игнорирования тела HTTP-документа (листинг 19.10).

Листинг 19.10. Использование CURL

```
<?php
  function get_content($hostname)
  {
    // Задаем адрес удаленного сервера
    $curl = curl_init($hostname);

    // Вернуть результат в виде строки
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
    // Включить в результат HTTP-заголовки
    curl_setopt($curl, CURLOPT_HEADER, 1);
    // Исключить тело HTTP-документа
    curl_setopt($curl, CURLOPT_NOBODY, 1);

    // Получаем HTTP-заголовки
    $content = curl_exec($curl);
    // Закрываем CURL-соединение
    curl_close($curl);

    // Преобразуем строку $content в массив
```

```
    return explode("\r\n", $content);
  }

  $hostname = "http://www.php.net";
  $out = get_content($hostname);

  echo "<pre>";
  print_r($out);
  echo "</pre>";
?>
```

19.5. Определение размера файла на удаленном хосте

Одной из распространенных задач является определение размера файла на удаленном хосте. Для этого воспользуемся функцией `get_content()` из листинга 19.8, с помощью которой узнаем количество байтов в файле по HTTP-заголовку `Content-Length` (листинг 19.11).

Листинг 19.11. Определение размера файла на удаленном хосте

```
<?php
  // Получаем HTTP-заголовки
  $hostname = "http://www.softtime.ru/files/configs.zip";
  $out = get_content($hostname);

  // Объединяем содержимое массива в одну строку
  $lines = implode(" ",$out);
  // Определяем количество байтов в закачиваемом файле
  // по регулярному выражению
  preg_match("|Content-Length:[\s]+([\d]+)|i", $lines, $matches);
  // Выводим результат
  echo "Количество байтов в архиве - ".$matches[1];
?>
```

Результат работы функции:

```
Количество байтов в архиве - 26421
```

19.6. Отправка данных методом POST

Создадим простейшую форму, состоящую из одного текстового поля `name` и кнопки для отправки данных (листинг 19.12).

Листинг 19.12. HTML-форма

```
<form action='handler.php' method='post'>
Имя : <input type='text' name='name'><br>
Пароль : <input type='text' name='pass'><br>
<input type='submit' value='Отправить'>
</form>
```

После заполнения текстового поля и нажатия на кнопку **Отправить** данные методом POST отправляются обработчику handler.php, код которого представлен в листинге 19.13.

Листинг 19.13. Обработчик handler.php

```
<?php
  echo "Имя - $_POST[name] <br>";
  echo "Пароль - $_POST[pass] <br>";
?>
```

Обработчик выводит в окно браузера текст, введенный в текстовые поля `name` и `pass` HTML-формы. HTML-форма помогает пользователю сформировать POST-запрос, который затем отсылается браузером. Такой запрос может быть сформирован и скриптом.

При обращении к серверу при помощи метода POST помимо HTTP-заголовка `POST /path HTTP/1.1` необходимо передать заголовок `Content-Length`, указав в нем количество байтов в области данных.

Метод POST, в отличие от метода GET, посылает данные не в строке запроса, а в области данных, после заголовков. Передача нескольких переменных аналогична методу GET: группы `имя=значение` объединяются при помощи символа амперсанда (`&`). Учитывая, что HTML-форма принимает параметр `name=Игорь`, `pass=пароль` строка данных может выглядеть следующим образом:

```
name=Игорь&pass=пароль
```

Кроме этого, необходимо учитывать, что данные передаются в текстовом виде, поэтому все национальные символы следует подвергать кодированию при помощи функции `urlencode()`.

Скрипт, отправляющий данные методом POST через сокеты, представлен в листинге 19.14.

ЗАМЕЧАНИЕ

Из-за ошибки библиотеки сокетов протокол HTTP 1.1 под Windows работает медленно, при работе скрипта под управлением данной операционной системы лучше использовать версию протокола HTTP 1.0.

Листинг 19.14. Отправка данных методом POST через сокеты

```
<?php
  $hostname = "localhost";
  $path = "/puzzles/handler.php";
  $line = "";
  // Устанавливаем соединение, имя которого передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Данные HTTP-запроса
    $data =
      "name=".urlencode("Игорь")."&pass=".urlencode("пароль")."\r\n\r\n";
    // Заголовок HTTP-запроса
    $headers = "POST $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    $headers .= "Content-type: application/x-www-form-urlencoded\r\n";
    $headers .= "Content-Length: ".strlen($data)."\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers.$data);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Результат работы скрипта может выглядеть следующим образом:

```
HTTP/1.1 200 OK
Date: Wed, 16 Nov 2005 18:00:44 GMT
Server: Apache/1.3.33 (Win32)
X-Powered-By: PHP/5.0.4
Transfer-Encoding: chunked
Content-Type: text/html

22
Имя — Игорь
Пароль - пароль
```

Остается только удалить HTTP-заголовки, как это описано ранее, и результат будет идентичен обращению к обработчику из HTML-формы.

> ***Замечание***
>
> Такого рода скрипты используются для автопостинга — автоматического размещения рекламных или провокационных сообщений в гостевых книгах и форумах в значительных количествах. Простейшие средства защиты, такие как проверка реферера или "прошивка" HTML-формы сессией, могут легко игнорироваться, как это будет демонстрироваться в последующих главах. Самым эффективным способом защиты от такого вида атак является автоматическая генерация изображения с кодом, который помещается в сессию. Пока пользователь не введет код в HTML-форму, сервис не срабатывает. Изображение может быть дополнено помехами, которые не помешают его прочитать "живому" посетителю, но потребуют от злоумышленника решения серьезной задачи распознавания образов.

В листинге 19.15 приводится решение рассматриваемой задачи с использованием CURL.

Листинг 19.15. Загрузка POST-данных с использованием CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/puzzles/5/handler.php");

  // Передача данных осуществляется методом POST
  curl_setopt($curl, CURLOPT_POST, 1);
  // Задаем POST-данные
  $data = "name=".urlencode("Игорь").
          "&pass=".urlencode("пароль")."\r\n\r\n";
  curl_setopt($curl, CURLOPT_POSTFIELDS, $data);

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

Для того чтобы сообщить CURL о том, что данные будут передаваться на сервер методом POST, необходимо задать параметр `CURLOPT_POST`. POST-данные устанавливаются при помощи параметра `CURLOPT_POSTFIELDS`.

19.7. Фальсификация реферера

Переходы пользователей со стороннего сайта фиксируются в реферере, получить доступ к которому можно при помощи элемента суперглобального массива `$_SERVER['HTTP_REFERER']`. Так, если страница 1.php содержит ссылку на страницу

2.php: `<a href='2.php'>Ссылка</a>`, то, разместив в файле 2.php следующий код из листинга 19.16, мы получим результат, приведенный на рис. 19.3.

Листинг 19.16. Вывод реферера

```
<?php
  echo "Вы перешли на текущую страницу с ".$_SERVER['HTTP_REFERER'];
?>
```

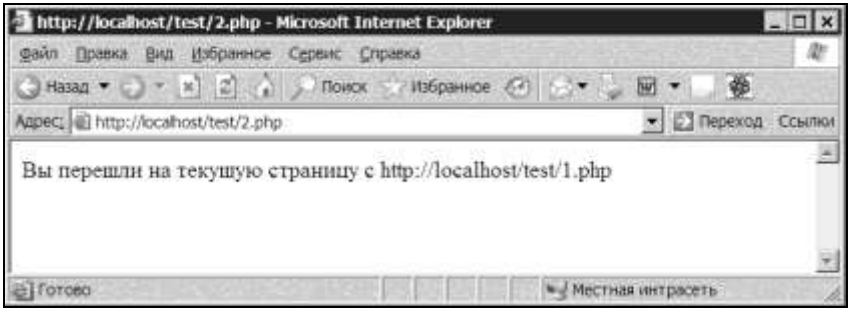


Рис. 19.3. Вывод реферера

Реферер иногда используется для простейшей защиты HTML-формы от автопостинга — заполнения HTML-формы при помощи скрипта. Рассмотрим это на примере. Пусть имеется HTML-форма авторизации в файле index.php, код которого представлен в листинге 19.17.

Листинг 19.17. HTML-форма системы авторизации

```
<table>
   <form action='handler.php' method='post'>
   <tr>
     <td>Имя:</td>
     <td><input type='text' name='name'></td>
   </tr>
   <tr>
     <td>Пароль:</td>
     <td><input type='password' name='pass'></td>
   </tr>
   <tr>
     <td> </td>
     <td><input type='submit' value='Войти'></td>
   </tr>
   </form>
</table>
```

Для защиты HTML-формы от заполнения на стороннем сайте в обработчик HTML-формы handler.php можно добавить проверку по рефереру, входит ли в его состав имя текущего сервера (листинг 19.18).

Листинг 19.18. Защита обработчика HTML-формы при помощи референера

```
<?php
  // Формируем путь к HTML-форме на текущем сервере
  $html = "http://".$_SERVER['SERVER_NAME'];
  // Проверяем, переданы ли данные с текущего сервера
  if (substr($_SERVER['HTTP_REFERER'], 0, strlen($html)) == $html)
  {
    if ($_POST['name'] == 'admin' && $_POST['pass'] == 'admin')
    {
       // Обработчик HTML-формы
    }
  }
  else
  {
    echo "Результат из формы не может быть обработан";
  }
?>
```

Однако такой способ защиты редко используется в реальной практике. Браузер или брандмауэр могут скрывать референер, поэтому такого рода защита может приводить к некорректной работе Web-приложения для легитимных пользователей. С другой стороны, референер легко подделывается: достаточно при помощи сокетов отправить HTTP-заголовок `Referer` (листинг 19.19) или воспользоваться CURL, установив при помощи функции `curl_setopt()` параметр `CURLOPT_REFERER` (листинг 19.20).

Листинг 19.19. Отправка референера при помощи сокетов

```
<?php
  $hostname = "localhost";
  $path = "/test/handler.php";
  $line = "";

  // Передаем методом POST имя пользователя (admin), его пароль (admin),
  // скрытое поле session_id ($SID).
  // В заголовках передаем cookie PHPSESSID.
  // Устанавливаем соединение, имя которого передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
```

```
  else
  {
    // Данные POST-запроса
    $data = "name=admin&pass=admin&\r\n\r\n";
    // Формируем HTTP-заголовки для передачи его серверу
    $headers = "POST $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    $headers .= "Content-type: application/x-www-form-urlencoded\r\n";
    $headers .= "Content-Length: ".strlen($data)."\r\n";
    // Подделываем реферер
    $headers .= "Referer: http://localhost/test/index.php\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers.$data);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Листинг 19.20. Отправка реферера при помощи CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/test/handler.php");

  // Передача данных осуществляется методом POST
  curl_setopt($curl, CURLOPT_POST, 1);
  // Задаем POST-данные
  $data = "name=admin&pass=admin&\r\n\r\n";
  curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
  // Устанавливаем реферер
  curl_setopt($curl, CURLOPT_REFERER,
              "http://localhost/test/index.php");

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

19.8. Фальсификация пользовательского агента

При обращении PHP-скрипта к страницам сайта при помощи файловых функций сервер делает в переменную окружения `USER_AGENT` запись вида: `PHP 6.0`. В результате скрипт получает доступ к этому идентификатору через элемент суперглобального массива `$_SERVER['USER_AGENT']`. Однако этот способ проверки не может считаться универсальным, т. к. переменную окружения устанавливает клиент при помощи HTTP-заголовка `User-Agent`. Любой HTTP-заголовок, который передает клиент, может быть подделан. В листинге 19.21 приводится пример, в котором скрипт, обращаясь к другому скрипту, выдает себя за пользователя операционной системы Windows XP, использующего браузер Internet Explorer версии 6.0. Для этого в качестве HTTP-заголовка `User-Agent` передается следующая строка:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

Листинг 19.21. Фальсифицируем пользовательский агент

```
<?php
  $hostname = "localhost";
  $path = "/test/index.php";
  $line = "";

  // Устанавливаем соединение, имя которого передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Формируем HTTP-заголовки для передачи серверу
    $headers = "GET $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    // Подделываем пользовательский агент, маскируясь
    // под пользователя Windows XP
    $headers .= "User-Agent: Mozilla/4.0 ".
                "(compatible; MSIE 6.0; Windows NT 5.1)\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers);
    // Получаем ответ
    while (!feof($fp))
    {
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

В листинге 19.22 представлено решение той же задачи с использованием расширения CURL. Для этого при помощи функции `curl_setopt()` устанавливается параметр `CURLOPT_USERAGENT` — его значение и будет передаваться серверу в HTTP-заголовке `User-Agent`.

Листинг 19.22. Фальсификация пользовательского агента при помощи CURL

```
<?php
  // Задаем адрес удаленного сервера
  $curl = curl_init("http://localhost/test/handler.php");

  // Устанавливаем реферер
  $useragent = "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1";
  curl_setopt($curl, CURLOPT_USERAGENT, $useragent);

  // Выполняем запрос
  curl_exec($curl);
  // Закрываем CURL-соединение
  curl_close($curl);
?>
```

19.9. Фальсификация cookie

Механизм действия cookie заключается в том, что, получив указания от сервера, браузер создает в оперативной памяти (если это сессионный cookie) или на жестком диске файл с заданными параметрами. После этого при каждом запросе браузер оправляет серверу HTTP-заголовок `Cookie` с именем и значением cookie. Так как за установку HTTP-заголовка несет ответственность клиентская сторона, cookie легко можно подделать, отправив HTTP-заголовок следующего формата:

```
Cookie: name1=value1; name2=value2; ...
```

где *`name1`*, *`name2`*, ... — имена cookie, а *`value1`*, *`value2`*, ... — их значения.

Пусть имеется скрипт, который проверяет наличие cookie, установленного у посетителя при аутентификации. В cookie `name` устанавливается имя пользователя, а установленные cookie `admin`, `editor` и `user` позволяют провести авторизацию для администратора, редактора и пользователя соответственно (листинг 19.23).

Листинг 19.23. Предоставление доступа к информации по cookie

```
<?php
  if (isset($_COOKIE['name']))
  {
     if (isset($_COOKIE['admin']))
     {
        echo "Здравствуйте, администратор $_COOKIE[name] !<br>";
```

```
    }
    if (isset($_COOKIE['editor']))
    {
       echo "Здравствуйте, редактор $_COOKIE[name] !<br>";
    }
    if (isset($_COOKIE['user']))
    {
       echo "Здравствуйте, пользователь $_COOKIE[name] !<br>";
    }
  }
?>
```

Скрипт, осуществляющий авторизацию с правами администратора, представлен в листинге 19.24.

> ***Замечание***
>
> Содержимое сессий не может быть фальсифицировано, т. к. хранящиеся в них данные не покидают пределов сервера. Однако фальсификации может быть подвергнут уникальный идентификатор сессии SID, который, как правило, передается через cookie. Таким образом скрипт может вести себя как браузер, поддерживая сессию, или, если злоумышленник получил доступ к идентификатору действующей сессии (например, в результате XSS-атаки), получать доступ к данным другого пользователя.

Листинг 19.24. Подделка cookie

```
<?php
  $hostname = "localhost";
  $path = "/test/1.php";

  // Устанавливаем соединение, имя которого передано в параметре $hostname
  $fp = fsockopen($hostname, 80, $errno, $errstr, 30);
  // Проверяем успешность установки соединения
  if (!$fp) echo "$errstr ($errno)<br />\n";
  else
  {
    // Формируем HTTP-заголовки для передачи серверу
    $headers = "GET $path HTTP/1.1\r\n";
    $headers .= "Host: $hostname\r\n";
    // Подделываем cookie
    $headers .= "Cookie: name=cheops; admin=1;\r\n";
    $headers .= "Connection: Close\r\n\r\n";
    // Отправляем HTTP-запрос серверу
    fwrite($fp, $headers);
    // Получаем ответ
    while (!feof($fp))
    {
```

```
      $line .= fgets($fp, 1024);
    }
    fclose($fp);
  }
  echo $line;
?>
```

Для того чтобы подделать cookie, скрипт отправляет HTTP-заголовок:

```
Cookie: name=cheops; admin=1;\r\n
```

в котором передается имя пользователя `cheops` и cookie `admin` со значением 1. Результат работы скрипта представлен на рис. 19.4. HTTP-заголовки ответа, которые предваряют вывод, не фильтруются, но могут быть легко удалены в случае необходимости.

Не составляет труда авторизоваться также с правами редактора или пользователя: достаточно заменить имя cookie `admin` на `editor` или `user`. Более того, в условиях скрипта из листинга 19.25 можно произвести авторизацию одновременно для всех трех видов пользователей. Для этого достаточно отправить HTTP-заголовок вида:

```
Cookie: name=cheops; admin=1; editor=1; user=1;\r\n
```

Результат работы такого скрипта представлен на рис. 19.5.



Рис. 19.4. Авторизация с правами администратора



Рис. 19.5. Авторизация одновременно всех трех видов пользователей

19.10. Работа с доменами и IP-адресами

Все хосты в сети Интернет имеют уникальные IP-адреса, однако для удобства вместо них используются доменные имена, которые при помощи DNS-серверов привязываются к IP-адресам. В протоколе HTTP 1.1 появилась возможность привязывать к одному IP-адресу несколько доменных имен.

Помимо функции `fsockopen()`, рассмотренной ранее, PHP располагает несколькими функциями, предназначенными для работы с IP-адресами и доменными именами (табл. 3.6). Кроме того, при помощи сокетов можно обратиться к Whois-сервису, который предоставляет информацию об IP-адресах.

***Таблица 3.6.** Функции для работы с IP-адресами и доменными именами*

Функция	Описание
`checkdnsrr($hostname [, $type])`	Находит на DNS-сервере записи ресурсов вида `$type` для хоста `$hostname`
`gethostbyname($hostname)`	Возвращает для доменного имени `$hostname` соответствующий ему IP-адрес
`gethostbynamel($hostname)`	Возвращает для доменного имени `$hostname` соответствующие ему IP-адреса
`gethostbyaddr($ip_address)`	Для IP-адреса `$ip_address` возвращает соответствующее ему доменное имя
`ip2long($ip_address)`	Преобразует IP-адрес `$ip_address` в целое число
`long2ip($proper_address)`	Преобразует целое число в IP-адрес `$proper_address`

19.11. Определение IP-адреса по сетевому адресу

Функция `gethostbyname()` принимает в качестве аргумента строку, представляющую доменное имя компьютера, и возвращает соответствующий IP-адрес. Функция имеет следующий синтаксис:

```
gethostbyname($hostname)
```

В листинге 19.25 приводится пример получения IP-адреса сервера сайта **http://www.yandex.ru**.

Листинг 19.25. Использование функции `gethostbyname()`

```
<?php
  $hostname = "www.yandex.ru";
  $ip_address = gethostbyname($hostname);
  echo "IP-адрес $hostname: $ip_address";
?>
```

Если компьютер подключен к Интернету, результатом работы скрипта из листинга 19.25 может быть строка:

```
IP-адрес www.yandex.ru: 213.180.204.11
```

Многие компьютеры имеют несколько IP-адресов; особенно типична такая ситуация для серверов. Получить полный список IP-адресов, соответствующих данному имени компьютера, можно с помощью функции `gethostbynamel()`, действующей аналогично функции `gethostbyname()`. Функция имеет следующий синтаксис:

```
gethostbynamel($hostname)
```

Другая ситуация, в которой полезно применение этой функции, — если одно имя DNS соответствует нескольким компьютерам. Это случается при работе с DNS-серверами, поддерживающими механизм кругового распределения нагрузки, при котором одно имя DNS-сервера отображается на несколько компьютеров в локальной сети этого сервера.

Возвращаемый список IP-адресов функция `gethostbynamel()` помещает в массив (листинг 19.26).

Листинг 19.26. Использование функции `gethostbynamel()`

```
<?php
  $hostname = "www.yandex.ru";
  $ip_addresses = gethostbynamel($hostname);
  echo "<pre>";
  print_r($ip_addresses);
  echo "</pre>";
?>
```

Если компьютер подключен к Интернету, результатом работы скрипта из листинга 19.26 может быть следующий дамп:

```
Array
(
    [0] => 213.180.204.11
)
```

19.12. Определение сетевого адреса по IP-адресу

Функция `gethostbyaddr()` принимает в качестве аргумента IP-адрес `$ip_address` и возвращает соответствующее ему имя хоста. Функция имеет следующий синтаксис:

```
gethostbyaddr($ip_address)
```

В листинге 19.27 приводится пример использования функции `gethostbyaddr()` применительно к локальному хосту.

Листинг 19.27. Использование функции `gethostbyaddr()`

```
<?php
  $ip_address = "127.0.0.1";
  $hostname = gethostbyaddr($ip_address);
  echo "Имя хоста с IP-адресом $ip_address: $hostname";
?>
```

Результатом работы скрипта из листинга 19.27 будет строка:

```
Имя хоста с IP-адресом 127.0.0.1: localhost
```

19.13. Следование реферальному серверу

Сервис Whois позволяет определить, на кого зарегистрирован тот или иной IP-адрес. Для этого достаточно послать IP-адрес по 43 порту соответствующего Whois-сервера. Существует множество Whois-серверов, каждый из которых несет ответственность за IP-адреса той или иной части света. Главным сервером является **whois.arin.net**. Если главный Whois-сервис не содержит информации о запрашиваемом IP-адресе, то в отчете обязательно присутствует строка `ReferralServer`, которая указывает адрес Whois-сервиса, ответственного за нужный диапазон IP-адресов. Поэтому обращение к Whois-сервису удобно выделить в отдельную функцию `whois()`, которая будет принимать два параметра: адрес Whois-сервиса и запрашиваемый IP-адрес. Если полученный отчет будет содержать ссылку на реферальный сервис, функция будет осуществлять рекурсивный спуск (листинг 19.28).

Листинг 19.28. Рекурсивное следование реферальному серверу

```
<center>
 <form method='post'>
 <input type='text' name='ip' size='35'>
 <input type='submit' value='Введите IP-адрес'>
 </form>
</center>
<?php
if (!empty($_POST['ip'])) echo whois("whois.arin.net",$_POST['ip']);

function whois($url,$ip)
{
  // Соединение с сокетом TCP, ожидающим на сервере
  // "whois.arin.net" по порту 43.
  // В результате возвращается дескриптор соединения $sock.
  $sock = fsockopen($url, 43, $errno, $errstr);
  if (!$sock) exit("$errno($errstr)");
  else
  {
    echo $url."<br>";
```

```
    // Записываем строку из переменной $_POST["ip"]
    // в дескриптор сокета.
    fputs ($sock, $ip."\r\n");
    // Осуществляем чтение из дескриптора сокета
    $text = "";
    while (!feof($sock))
    {
      $text .= fgets ($sock, 128)."<br>";
    }
    // Закрываем соединение
    fclose ($sock);

    // Ищем реферальный сервер
    $pattern = "|ReferralServer: whois://([^\n<:]+)|i";
    preg_match($pattern, $text, $out);
    if (!empty($out[1])) return whois($out[1], $ip);
    else return $text;
  }
}
?>
```

Скрипт всегда начинает поиск с **whois.arin.net**, при необходимости осуществляя повторные запросы к реферальным сервисам. При этом перед отчетом выводится список Whois-сервисов, которые посещает скрипт. Так для адреса 213.115.162.82, который соответствует сайту корпорации AB MySQL, отчет может выглядеть так, как это представлено на рис. 19.6.

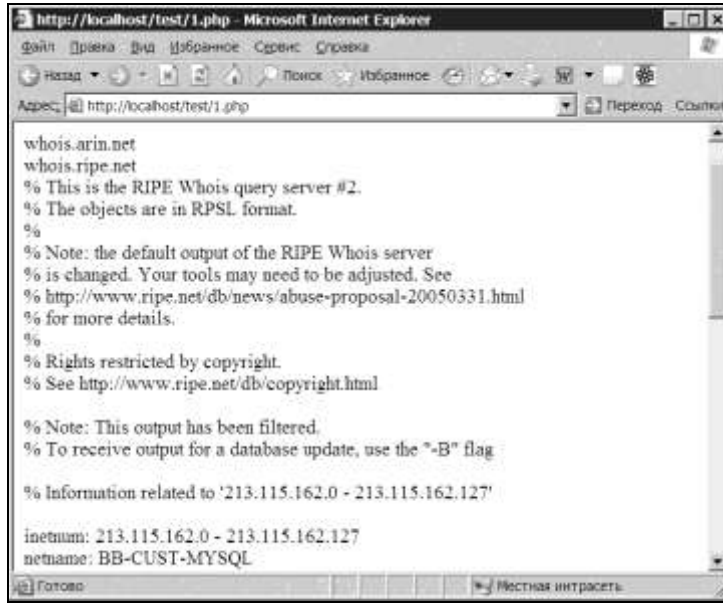


Рис. 19.6. Принадлежность IP-адреса 213.115.162.82

Как видно из рис. 19.6, скрипт посещает два Whois-сервиса: **whois.arin.net** и **whois.ripe.net**.

Для IP-адреса 192.131.90.161, который соответствует сетевому адресу **http://www.w3c.org**, отчет может выглядеть так, как это представлено на рис. 19.7. Из отчета видно, что информация по данному IP-адресу была найдена в Whois-сервисе по адресу **whois.apnic.net**.

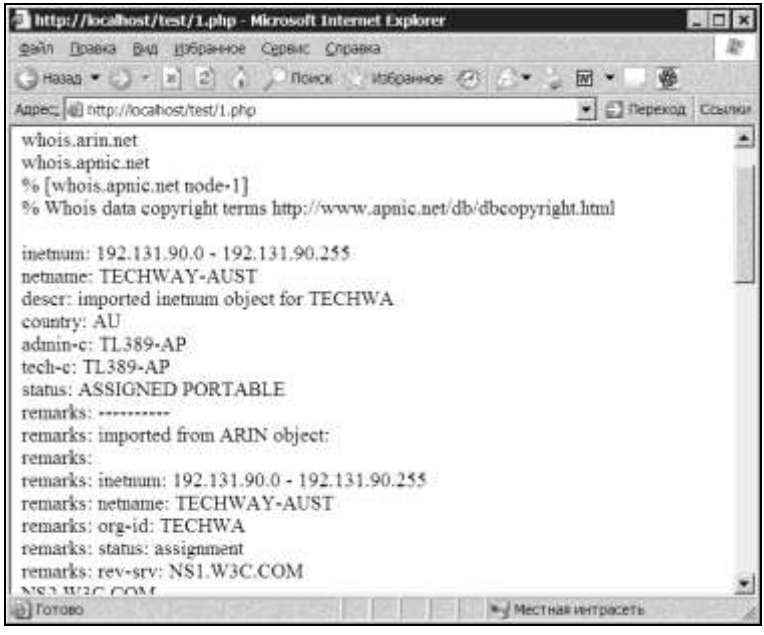


Рис. 19.7. Принадлежность IP-адреса 192.131.90.161



Глава 20

Взаимодействие с интерпретатором PHP

Интерпретатор PHP имеет множество параметров настройки, которые влияют на его поведение. Изменять настройки можно как прямым редактированием конфигурационного файла php.ini, так и при помощи специальных функций, которые будут рассмотрены в данной главе.

20.1. Настройки конфигурационного файла php.ini

Конфигурационный файл php.ini состоит из секций и директив. Секции заключаются в квадратные скобки, например, `[PHP]`, после которых следуют директивы, имеющие такой формат:

```
directive = value
```

Здесь `directive` — это название директивы, а `value` — ее значение. Все строки, в начале которых располагается точка с запятой (`;`), считаются комментариями и игнорируются.

Допускается не указывать значение `value`. В этом случае, директива инициализируется пустой строкой. Этого же результата можно добиться, присвоив ей значение `none`.

Файл php.ini начинается с директивы `[PHP]`, которая позволяет настроить параметры ядра, после чего следуют директивы расширений, которые могут подключаться как внешние библиотеки (такие как `[MySQL]` или `[PostgresSQL]`), так и расширения, включенные в состав ядра (например, `[Date]` или `[Session]`, настраивающие порядок работы с датой и сессиями).

Если PHP подключен к Web-серверу в качестве модуля, изменения в конфигурационном файле php.ini вступают в силу только после перезагрузки Web-сервера. В случае если PHP подключен как внешнее CGI-приложение, изменения вступают в силу немедленно.

По умолчанию, интерпретатор PHP последовательно ищет конфигурационный файл в нескольких местах:

- по пути, указанному в директиве `PHPIniDir` конфигурационного файла httpd.conf Web-сервера Apache (директива действует, начиная с Apache 2.2);
- по пути, указанному в переменной окружения `PHPRC` (начиная с версии PHP 5.2);

- в текущем каталоге (если скрипт выполняется под управлением Web-сервера);
- в каталоге C:\Windows\ в случае Windows и в каталоге компиляции в случае любой другой операционной системы.

Если PHP-скрипт запускается вне среды Web-сервера, указать путь к конфигурационному файлу php.ini можно при помощи параметра `-c`, например:

```
php.exe -c C:/PHP/php.ini D:/scripts/file.php
```

В следующих нескольких разделах будут рассмотрены директивы ядра `[PHP]`.

20.1.1. Параметры языка PHP

В табл. 20.1 представлены наиболее часто используемые директивы группы управления параметрами языка. В скобках указывается значение, которое может принимать директива.

Таблица 20.1. *Директивы управления параметрами языка*

Директива	Описание
`engine`	Включает (`On`) или выключает (`Off`) выполнение PHP-скриптов под управлением Web-сервера Apache
`short_open_tag`	Включает (`On`) или выключает (`Off`) возможность использования короткого тега `<?` вместо `<?php` *(см. главу 2)*
`asp_tags`	Включает (`On`) или выключает (`Off`) возможность использования ASP-тегов `<%` и `%>` вместо `<?php` и `?>` *(см. главу 2)*. Директива исключена в PHP 6
`precision`	Указывает количество знаков, которое отводится под дробное число, в случае, если вывод не форматируется при помощи специальных функций, таких как `printf()`, `sprintf()` и т. п.
`output_buffering`	Включает (`On`) или выключает (`Off`) автоматическую буферизацию вывода *(см. главу 17)*. В качестве значения директива может принимать число байтов, по достижению которых буфер автоматически очищается, а его содержимое отправляется клиенту

Директива `engine` по умолчанию принимает значение `On`, установка данной директивы в значение `Off` приводит к тому, что PHP-скрипты перестают интерпретироваться под управлением Web-сервера. Попытка обратиться к PHP-скрипту вместо выдачи HTML-результата приведет к выводу окна, предлагающего сохранить файл (рис. 20.1).

Директивы `short_open_tag` и `asp_tags` определяют доступность различных тегов, обрамляющих PHP-скрипт. Традиционно директива `short_open_tag`, позволяющая использовать короткий вариант `<?`, включена (`On`). Директиву имеет смысл отключить, если PHP-скрипты используются совместно с XML-файлами, которые используют теги `<?xml` и `?>`.

Директива `asp_tags`, позволяющая использовать теги языка ASP `<%` и `%>`, отключена `Off`. Начиная с версии PHP 6, возможность использования ASP-тегов и сама директива `asp_tags` исключаются из языка.

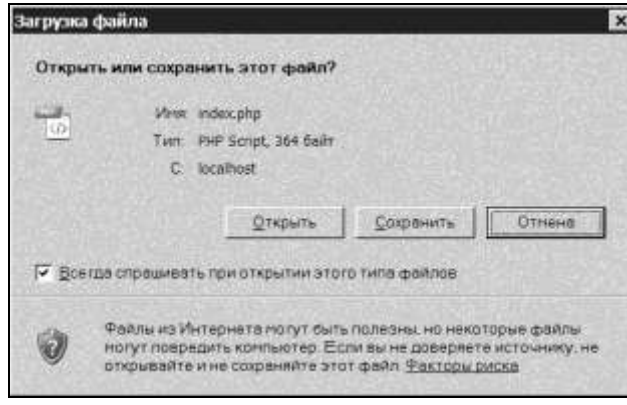


Рис. 20.1. Отключение интерпретации PHP-файла

Директива `precision` позволяет задать количество знаков, которое отводится под вывод вещественного числа. В листинге 20.1 приводится скрипт, выводящий два вещественных числа в научном и стандартном форматах.

> ***Замечание***
>
> Директива `precision` не влияет на точность вычисления, она определяет только количество символов после запятой в случае неформатного вывода вещественного числа.

Листинг 20.1. Вывод вещественных чисел

```
<?php
  echo 10.23456;
  echo "<br />";
  echo 10.23456E+20;
?>
```

В случае, если значение директивы `precision` равно 4, скрипт выведет следующую пару чисел:

```
10.23
1.023E+21
```

Если значение директивы `precision` превышает количество цифр в числе, они выводятся без изменений (в отличие от форматирования, дополнительные нули не добавляются):

```
10.23456
1.023456E+21
```

Директива `output_buffering`, отключенная по умолчанию (`Off`), позволяет включить (`On`) буферизацию вывода. По умолчанию, PHP-интерпретатор отправляет клиенту данные сразу после их обработки, такой не буферизированный вывод может порождать

некоторые проблемы. С одной стороны, данные отправляются небольшими порциями, что может приводить к замедлению вывода, с другой стороны, это жестко регламентирует отправку HTTP-заголовков, которые должны отправляться всегда перед данными. Рассмотрим работу директивы `output_buffering` на примере скрипта из листинга 20.2.

Листинг 20.2. Отправка HTTP-заголовков после вывода данных

```
<?php
  echo "hello world!";
  session_start();
?>
```

В скрипте осуществляется попытка отправки данных перед функцией `session_start()`, которой для регистрации сессии необходимо отправить клиенту HTTP-заголовок (напомним, что SID-сессии передается через cookies, которые в свою очередь устанавливаются при помощи HTTP-заголовка `Set-Cookie`). При отключенной буферизации попытка выполнения скрипта из листинга 20.2 приводит к ошибке "Cannot send session cookie - headers already sent by" (рис. 20.2).

Включив буферизацию (`output_buffering = On`) в конфигурационном файле php.ini, можно заставить помещать все данные в буфер, тогда интерпретатор получит возможность сначала отправить HTTP-заголовки и лишь потом данные, независимо от порядка их вывода в скрипте (рис. 20.3).

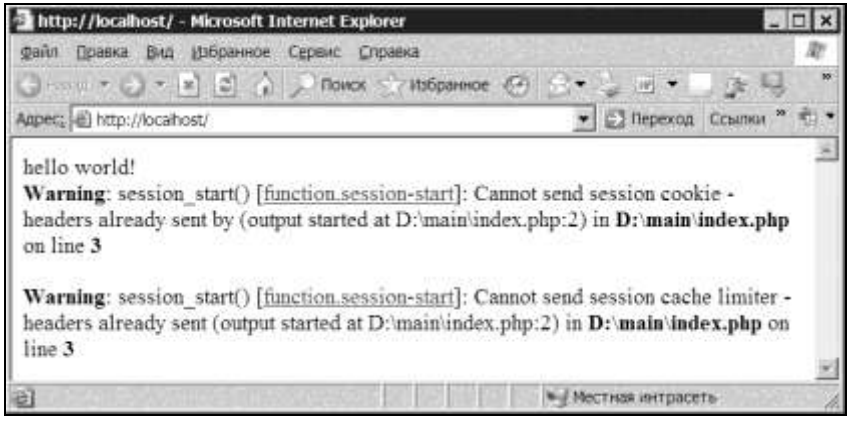


Рис. 20.2. HTTP-заголовки должны отправляться перед данными

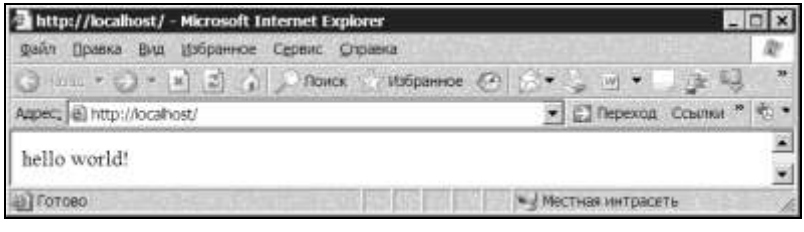


Рис. 20.3. Буферизация вывода позволяет использовать HTTP-заголовки в произвольном месте

Вместо значения `On` директива `output_buffering` может принимать максимальное количество байтов в буфере, по достижению которых буфер автоматически очищается, а его содержимое отправляется клиенту. Строка "hello world!" в однобайтовой кодировке занимает 12 байт, поэтому при значении `output_buffering` больше 12, скрипт из листинга 20.2 выполняется без ошибок, при значениях директивы меньше 12 — с ошибкой.

20.1.2. Директивы безопасности

Один Web-сервер может поддерживать множество сайтов. Такое соседство порождает потенциальную опасность: скрипты одного сайта могут намеренно или случайно обращаться к скриптам соседнего сайта. Задачу разграничения прав традиционно решают средствами операционной системы и Web-сервера. Однако PHP имеет некоторые настройки, позволяющие упростить процесс такой настройки.

Безопасный режим для ограничения доступа PHP к файлам использует UNIX-права доступа. UNIX — многопользовательская операционная система, в которой может быть несколько пользователей, которые в свою очередь объединяются в группы (любой пользователь входит, как минимум, в одну группу). Все файлы и каталоги снабжаются двумя атрибутами — владелец файла и группа пользователей, которые могут работать с файлом. Поэтому права доступа для каждого файла или каталога выставляются для трех категорий пользователей:

- владелец файла;
- группа владельцев;
- остальные пользователи, не являющиеся владельцем и не входящие в группу владельцев.

В рамках каждой из трех групп существуют три вида разрешений:

- `r` — право чтения данного файла;
- `w` — право записи/изменения данного файла;
- `x` — право выполнения данного файла, если он является сценарием или программой (для каталога право его просмотра).

Таким образом, права доступа могут выглядеть так: `rwxr--r--`. Такая запись означает, что владелец файла может читать файл, записывать и выполнять, а все остальные пользователи только читать. UNIX-права доступа можно задавать восьмеричным числом. При этом праву чтения соответствует цифра 4, праву записи — 2, а исполнению — 1. Общие права для групп задаются суммой этих чисел, так 6 (4 + 2) обеспечивает возможность чтения и записи, а 7 (4 + 2 + 1) — предоставляет полный доступ к файлу или каталогу. Тогда для каталога восьмеричное число 0755 означает, что владелец каталога имеет полный доступ (`rwx`), а все остальные имеют право читать файлы в нем и просматривать содержимое каталога (`r-x`).

Безопасный режим в первую очередь проверяет, совпадают ли владелец скрипта и файла или каталога, которыми скрипт оперирует. При помощи директивы `safe_mode_gid` можно смягчить это ограничение: присвоив данной директиве значение `On`, можно потребовать проверять совпадение группы владельцев.

Следует отметить, что ограничение на чтение и модификацию файлов распространяется не только на файловые функции, такие как `file_get_contents()`, `put_get_contents()`, `fopen()`, `fwrite()`, `fread()`, но и на директивы `include`, `include_once`, `require` и `require_once`. Такое поведение может быть не очень удобным, особенно, если общие для множества проектов включаемые файлы хранятся в отдельном каталоге (в первую очередь имеется в виду библиотека PEAR с готовыми PHP-решениями). Для того чтобы снять ограничение на включаемые файлы, используется директива `safe_mode_include_dir` — на каталоги, указанные в этой директиве не распространяется проверка владельца пользователя и группы владельцев.

PHP-скрипт может не только самостоятельно обратиться к чужому файлу, но и использовать функции, выполняющие команды операционной системы, такие как `system()`, `exec()`, `shell_exec()`, т. е. скрипт может попытаться изменить чужой файл при помощи средств операционной системы (рис. 20.4).

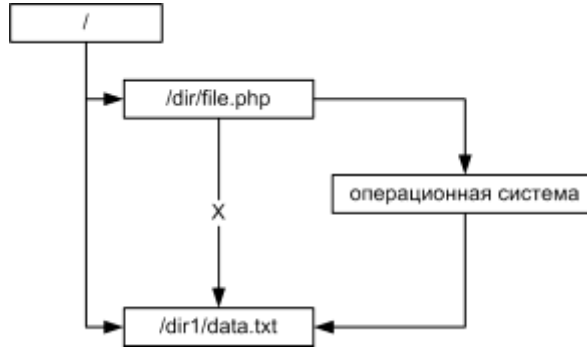


Рис. 20.4. Попытка обращения к чужому файлу через команды операционной системы

Безопасный режим предотвращает модификацию чужих файлов, в том числе и при помощи функций, способных выполнять команды операционной системы. Так как запрещается выполнение вообще любых команд операционной системы, это может создавать определенные неудобства. При помощи директивы `safe_mode_exec_dir` можно указать каталог с исполняемыми файлами, на которые защищенный режим не будет распространяться.

Еще одним важным ресурсом являются переменные окружения, при помощи которых операционная система и среда выполнения (в нашем случае Web-сервер) могут обмениваться с исполняемыми файлами информацией. Операционная система или Web-сервер могут заполнять переменные окружения, а исполняемый модуль (например, PHP) может читать и использовать эту информацию. В качестве примера можно привести обработку GET-запроса. Получив GET-параметры, Web-сервер создает переменную окружения `QUERY_STRING` и помещает в нее GET-параметры. Теперь PHP получает возможность прочитать значение из переменной окружения и заполнить суперглобальные массивы `$_SERVER` и `$_GET`. Переменных окружения может быть достаточно много, сам PHP-скрипт, как будет показано далее в этой главе, также имеет возможность изменять переменные окружения (рис. 20.5).

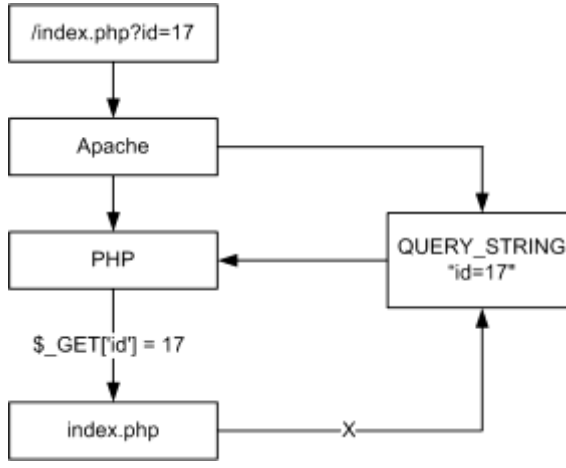


Рис. 20.5. Взаимодействие PHP и Apache при помощи переменных окружения

Замечание

Запрет на изменение переменных окружения необходим главным образом для запрета изменения таких важных параметров, как объем памяти и время выполнения, выделяемые на один экземпляр скрипта.

Разумеется, попытка исправить переменную окружения `QUERY_STRING` постфактум, как это показано на рис. 20.5, не может привести к какому-либо вреду, однако некоторые переменные окружения могут влиять на безопасность. Поэтому в безопасном режиме разрешается изменять лишь те переменные окружения, которые начинаются с префикса, указанного в директиве `safe_mode_allowed_env_vars`. По умолчанию директива разрешает изменять переменные окружения, начинающиеся с префикса `PHP_`:

```
safe_mode_allowed_env_vars = PHP_
```

Если некоторые переменные окружения необходимо запретить изменять, даже если они начинаются с префикса `PHP_`, можно воспользоваться директивой `safe_mode_protected_env_vars` — любые переменные окружения (отделенные друг от друга двоеточием), перечисленные в этой директиве, запрещено изменять в защищенном режиме, даже если они начинаются с префикса, заданного директивой `safe_mode_allowed_env_vars`.

В табл. 20.2 представлен полный список директив, позволяющих настраивать защищенный режим PHP.

***Таблица 20.2.** Директивы защищенного режима*

Директива	Описание
`safe_mode`	Включает (`On`) или выключает (`Off`) защищенный режим, ограничивающий доступ PHP к чужим файлам и некоторым функциям

Таблица 20.2 *(окончание)*

Директива	Описание
safe_mode_gid	Включает (On) или выключает (Off) более мягкий вариант защищенного режима — вместо проверки пользователя файла, проверяется группа-пользователь
safe_mode_include_dir	На каталоги, указанные в данной директиве, не распространяется проверка пользователя. Если необходимо указать несколько каталогов, их разделяют символом, используемым в переменных окружения в текущей операционной системе (двоеточие для UNIX-подобных операционных систем и точка с запятой для Windows)
safe_mode_exec_dir	Исполняемые файлы из каталога, указанного в директиве safe_mode_exec_dir, можно выполнять при помощи функций system(), exec() и т. п. Все остальные исполняемые файлы в защищенном режиме будут игнорироваться
safe_mode_allowed_env_vars	Позволяет задать префикс. Переменные окружения, начинающиеся с него, могут быть изменены из PHP-скрипта в защищенном режиме. Изменение всех остальных переменных запрещается. Если директива принимает пустую строку, разрешается изменение любых переменных окружения
safe_mode_protected_env_vars	Позволяет задать переменные окружения, запрещенные к модификации в защищенном режиме. Значения этих переменных остаются защищенными, даже если их модификация разрешена директивой safe_mode_allowed_env_vars

В качестве демонстрации защищенного режима рассмотрим запрет на выполнение команд операционной системы. Попробуем прочитать содержимое текущего каталога при помощи команды dir (для Windows) или ls -l (для Linux). Проще всего воспользоваться функцией shell_exec(), которая выполняет команду и возвращает результат вывода команды в стандартный поток (листинг 20.3).

> ***Замечание***
>
> В *разд. 3.6* рассматривались обратные кавычки, которые также позволяют выполнять команды операционной системы и внешние исполняемые файлы. Вызову shell_exec("dir") будет эквивалентна команда `dir`.

Листинг 20.3. Выполнение команды операционной системы

```
<?php
  $text = shell_exec("dir");
  echo "<pre>";
```

```
  echo htmlspecialchars(convert_cyr_string($text, 'd', 'w'));
  echo "</pre>";
?>
```

Формат вывода команды предназначен для командной строки, поэтому просто вывода `echo shell_exec("dir")` недостаточно. Во-первых, в браузере переводы строк рассматриваются как обычные пробельные символы, поэтому их следует либо заменить при помощи функции `nl2br()` на тег `<br>`, либо обрамить вывод в теги `<pre>` и `</pre>`. Во-вторых, результат выполнения функции возвращается в кодировке cp866 (DOS), что не очень удобно, т. к. сервер, как правило, настроен на кодировку либо cp1251 (Windows), либо KOI8-R. Поэтому кодировка вывода в листинге 20.3 меняется при помощи функции `convert_cyr_string()`. В-третьих, в выводе команды `dir` для обозначения каталога используется последовательность `<DIR>`, которая воспринимается браузером как тег и в конечном итоге приводит к ступенчатому отображению результата. Поэтому результат пропускается через функцию `htmlspecialchars()` для того, чтобы предотвратить интерпретацию последовательности `<DIR>`. На рис. 20.6 представлен результат выполнения скрипта в штатном режиме.

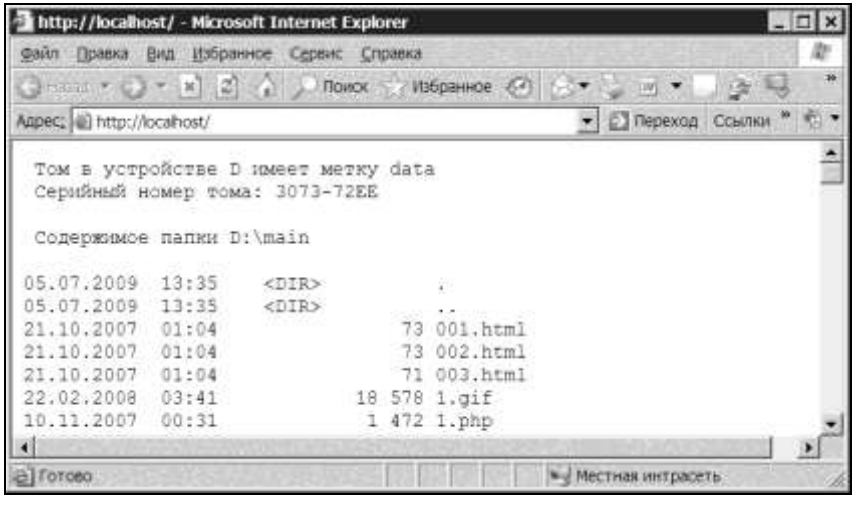


Рис. 20.6. Результат выполнения команды `dir`

Если в конфигурационном файле php.ini включить директиву `safe_mode`, вместо содержимого текущего каталога скрипт из листинга 20.3 выдаст предупреждение, сообщающее о невозможности выполнения функции `shell_exec()` в безопасном режиме (рис. 20.7).

Замечание

В защищенном режиме использование обратных кавычек также запрещено.

PHP предоставляет администраторам ряд директив, которые работают независимо от того, включен защищенный режим или нет (табл. 20.3).

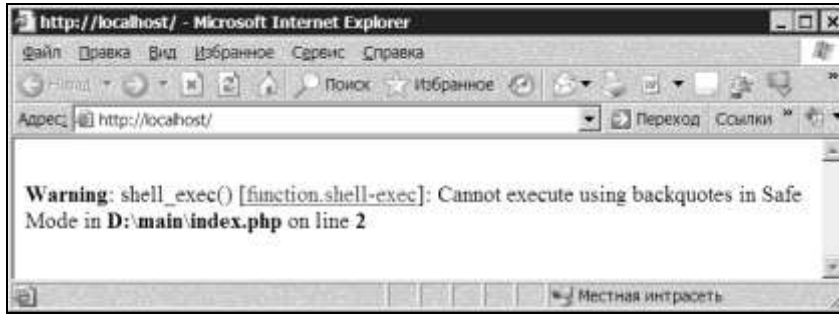


Рис. 20.7. Выполнение команд операционной системы запрещено в защищенном режиме

Таблица 20.3. *Директивы безопасности*

Директива	Описание
`open_basedir`	Позволяет задать список каталогов, в которых разрешено выполнение PHP-скриптов и доступ к файлам. Если необходимо указать несколько каталогов, их разделяют символом, используемым в переменных окружения в текущей операционной системе (двоеточие для UNIX-подобных операционных систем и точка с запятой для Windows)
`disable_functions`	Позволяет задать список функций (разделенных запятой), запрещенных к выполнению
`disable_classes`	Позволяет задать список классов (разделенных запятой), запрещенных к использованию

Директива `open_basedir` позволяет задать список каталогов, в которых разрешено выполнение PHP-скриптов и обращение к файлам при помощи файловых функций (разрешение автоматически распространяется и на подчиненные каталоги). Указанные в директиве каталоги выступают в качестве префикса. Например,

```
open_basedir = "D:/main/pro"
```

Путь "D:/main/pro" разрешает доступ как в каталог D:/main/pro/, так и в D:/main/projects/ и D:/main/progress/. Для того чтобы ограничить доступ только каталогом pro, следует в конце строки явно указать слеш — "D:/main/pro/".

> ***Замечание***
>
> Используя директиву `open_basedir` совместно с виртуальными хостами Web-сервера, можно добиться разграничения PHP по виртуальным хостам таким образом, чтобы скрипты одного сайта не могли получить доступ к файлам другого сайта *(см. разд. 20.2)*.

Директива `disable_functions` позволяет задать список функций, запрещенных к использованию. Например, следующее значение запретит выполнение функций `set_time_limit()` и `putenv()`:

```
disable_functions  = "set_time_limit,putenv"
```

20.1.3. Настройка подсветки PHP-кода

В распоряжение разработчиков PHP предоставляет функции, обеспечивающие подсветку PHP-кода (табл. 20.4).

Таблица 20.4. *Подсветка PHP-кода*

Функция	Описание
`highlight_string($code [, $return])`	Функция подсвечивает синтаксис содержимого PHP-скрипта `$code` и выводит в окно браузера. Если необязательный параметр `$return` принимает значение `TRUE`, вместо непосредственного вывода функция возвращает строку с результатом
`highlight_file($filename [, $return])`	Функция открывает PHP-файл `$filename`, подсвечивает его содержимое и выводит в окно браузера. Если необязательный параметр `$return` принимает значение `TRUE`, вместо непосредственного вывода функция возвращает строку с результатом

Пусть имеется файл test.php, который необходимо вывести с подсветкой синтаксиса (листинг 20.4).

Листинг 20.4. Файл test.php

```
<?php
  if (!$flag)
  {
    // Пишем любой код
    echo "Hello";
    $var = 1;
  }
  else break;
?>
```

В листинге 20.5 приводится пример использования функции `highlight_file()`, которая извлекает содержимое файла test.php и выводит его в окно браузера.

Листинг 20.5. Использование функции `highlight_file()`

```
<?php
  highlight_file("test.php");
?>
```

На рис. 20.8 демонстрируется результат выполнения скрипта из листинга 20.5.



Рис. 20.8. Функция `highlight_file()` подсвечивает синтаксис PHP-скрипта

Если вывод результата функции непосредственно в месте ее вызова не целесообразен (или необходима предварительная обработка получившегося HTML-кода), можно передать второму необязательному параметру функции значение `true` — в этом случае результат можно сохранить в строке (листинг 20.6).

Листинг 20.6. Использование второго параметра функции `highlight_file()`

```
<?php
  $code = highlight_file("test.php", true);
  // Промежуточный код
  ...
  // Вывод результата
  echo $code;
?>
```

Единственное неудобство в использовании функции заключается в том, что цвета программных элементов жестко заданы: комментарии — оранжевый, переменные и числа — синий, строки — красный, ключевые слова — зеленый и т. д. Изменить цвета по умолчанию можно в конфигурационном файле php.ini при помощи директив, представленных в табл. 20.5.

Замечание

Директива `highlight.bg` исключена, начиная с PHP 6.0.

Цвет задается в HTML-формате, т. к. подсвечиваемые элементы обрамляются `span`-тегами вида

```
<span style="color: #0000BB">$var = 1;</span>
```

В качестве цвета может использоваться шестнадцатеричная нотация #RRBBGG, задающая красный (R), синий (B) и зеленый (G) цвета двухзначным шестнадцатеричным числом (в десятичной системе счисления пробегающим значение от 0 до 255). Кроме этого, могут использоваться предопределенные цвета HTML, такие как `red`, `blue` и т. п.

Таблица 20.5. *Директивы подсветки PHP-кода*

Директива	Описание
highlight.string	Определяет цвет строк, по умолчанию это красный цвет (#DD0000)
highlight.comment	Определяет цвет комментариев, по умолчанию это оранжевый цвет (#FF9900)
highlight.keyword	Определяет цвет ключевых слов, по умолчанию это зеленый цвет (#007700)
highlight.bg	Определяет цвет фона, по умолчанию это белый цвет (#FFFFFF)
highlight.default	Определяет цвет остальных элементов PHP-кода, по умолчанию это синий цвет (#0000BB)
highlight.html	Определяет цвет HTML-кода, по умолчанию это черный цвет (#000000)

20.1.4. Кэш файловой системы

Некоторые файловые функции *(см. разд. 13.5—13.6)* кэшируют результаты своей работы. Директивы, представленные в табл. 20.6, позволяют управлять размером этого кэша и частотой его обновления.

ЗАМЕЧАНИЕ

Функция `clearstatcache()` позволяет очистить кэш досрочно.

Таблица 20.6. *Директивы управления кэшем файловой системы*

Директива	Описание
realpath_cache_size	Определяет размер кэша, по умолчанию — 16 Кбайт
realpath_cache_ttl	Определяет частоту обновления кэша, по умолчанию — 120 секунд

20.1.5. Взаимодействие с клиентом

Директивы данной группы определяют параметры взаимодействия с клиентом (табл. 20.7).

Таблица 20.7. *Директивы взаимодействия с клиентом*

Директива	Описание
ignore_user_abort	Если директива включена (`On`), скрипт не завершает работу после разрыва соединения с клиентом
expose_php	Разрешает (`On`) или запрещает (`Off`) сообщать клиенту, что сайт использует PHP при помощи HTTP-заголовка

20.1.6. Ограничение ресурсов

Сервер может поддерживать множество сайтов, которые в свою очередь могут обслуживать множество клиентов. Такое положение дел вынуждает ограничивать скрипты как по количеству используемой памяти, так и по времени выполнения.

> ***Замечание***
>
> Директивы в табл. 20.8 вводят ограничение на процессорное время. Это означает, что время ожидания ответа базы данных или ответа удаленного сервера не учитывается.

Таблица 20.8. *Директивы ограничения ресурсов*

Директива	Описание
`max_execution_time`	Определяет максимальное количество процессорного времени (в секундах), отводимого на выполнение скрипта. Если директива принимает значение 0, скрипт выполняется бессрочно. По умолчанию принимает значение 30 секунд
`max_input_time`	Максимальное количество процессорного времени (в секундах), отводимое на разбор GET-, POST-данных и загруженных файлов. Если директива принимает значение –1, по умолчанию принимается значение 60 секунд
`memory_limit`	Максимальное количество памяти, выделяемое под один экземпляр скрипта

В подавляющем большинстве случаев 30 секунд более чем достаточно для выполнения скрипта, задаваемого директивой `max_execution_time`. Если все же этот лимит превышается, работа скрипта останавливается, а в окно браузера выводится предупреждение (рис. 20.9).

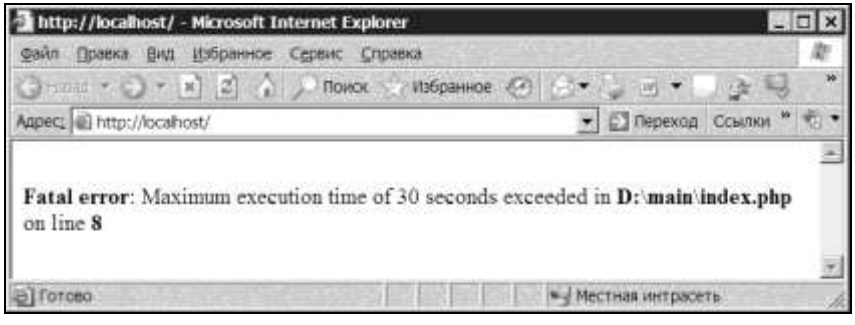


Рис. 20.9. Превышение ограничения в 30 секунд процессорного времени

Схожим образом действует ограничение на количество выделяемой памяти, задаваемое директивой `memory_limit`. Данное ограничение предназначено для того, чтобы неэффективные скрипты не могли в короткое время исчерпать память сервера.

Замечание

Чаще всего с ограничениями по памяти разработчики сталкиваются при попытке прочитать гигантского размера файлы в переменную скрипта или при работе с объемными изображениями средствами библиотеки GDLib *(см. главу 31)*.

На рис. 20.10 приводится типичный результат работы скрипта, исчерпавшего весь отведенный ему лимит оперативной памяти.

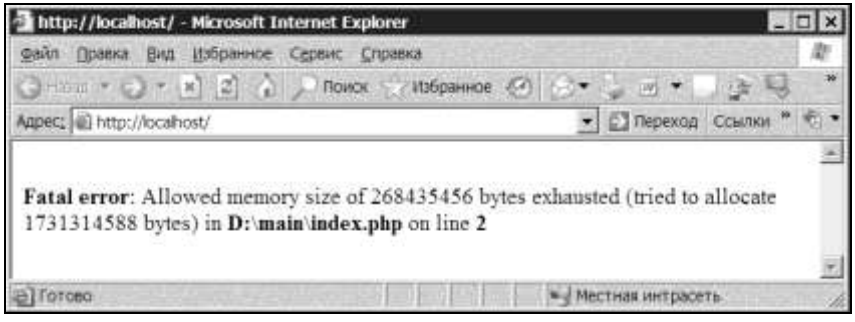


Рис. 20.10. Достижение предела выделенной оперативной памяти

20.1.7. Обработка ошибок

Директивы группы обработки ошибок позволяют настроить уровень обработки ошибок и методы сообщения об ошибках интерпретатором PHP. В табл. 20.9 приводится список доступных директив.

***Таблица 20.9.** Директивы обработки ошибок*

Директива	Описание
`error_reporting`	Устанавливает уровень обработки ошибок. Возможные значения более подробно описываются в *разд. 24.1*
`display_errors`	Включает (`On`) или отключает (`Off`) отображение ошибок в окно браузера
`display_startup_errors`	Включает (`On`) или отключает (`Off`) отображение ошибок запуска. Этот тип ошибок управляется отдельно от всех остальных типов ошибок
`log_errors`	Включает (`On`) или отключает (`Off`) сохранение ошибок в журнал, местоположение которого определяется директивой `error_log`
`log_errors_max_len`	Определяет максимальное количество байтов, которое сохраняется в журнал ошибок. Значение 0 снимает ограничение на размер сохраняемого сообщения
`ignore_repeated_errors`	Включает (`On`) или отключает (`Off`) игнорирование повторяющихся ошибок (произошедших в том же самом файле, на той же самой строке)

Таблица 20.9 *(окончание)*

Директива	Описание
track_errors	Если директива включена (On), сообщение о последней ошибке сохраняется в переменную $php_errormsg
error_log	Определяет местоположение журнала ошибок. Это может быть обычный текстовый файл или служба регистрации событий. Если директива принимает значение syslog, сообщения об ошибках сохраняются в журналах службы syslog в UNIX и журнале событий в Windows

Зачастую рекомендуется сохранять сообщения об ошибках не только на рабочих серверах, но и во время разработки. Последнее особенно полезно, если сообщение об ошибке скрывается дизайном или теряется в результате редиректа. Для того чтобы включить сохранение сообщений в журнал, необходимо произвести настройки конфигурационного файла php.ini в соответствии с листингом 20.7.

Замечание

Более подробная настройка директивы error_reporting обсуждается в *разд. 24.1*.

Листинг 20.7. Настройка директив обработки ошибок

```
; Реагируем на все сообщения об ошибках за исключением замечаний
error_reporting = E_ALL & ~E_NOTICE
; Включаем отображение ошибок в окне браузера
display_errors = On
; Включаем сохранение сообщений об ошибках
log_errors = On
; Ограничиваем длину сообщения об ошибках 1 килобайтом
log_errors_max_len = 1024
; Предписываем сохранение сообщений об ошибках в текстовый файл
error_log = "D:/php.log"
```

Теперь все ошибки, помимо отображения в окне браузера, будут дублироваться в том числе в файл php.log; если директиве error_log присвоить значение syslog, сообщения будут сохраняться в службе регистрации событий (листинг 20.8).

Листинг 20.8. Регистрация ошибок операционной системой

```
; Реагируем на все сообщения об ошибках за исключением замечаний
error_reporting = E_ALL & ~E_NOTICE
; Включаем отображение ошибок в окне браузера
display_errors = On
; Включаем сохранение сообщений об ошибках
log_errors = On
; Ограничиваем длину сообщения об ошибках 1 килобайтом
```

```
log_errors_max_len = 1024
; Предписываем сохранение сообщений системной службой регистрации событий
error_log = syslog
```

Получить доступ к ошибкам можно при помощи консоли службы регистрации событий: **Пуск | Настройка | Панель управления | Администрирование | Просмотр событий**. В консоли следует выбрать раздел **Приложение** и ориентироваться на сообщения с источником PHP (рис. 20.11).

Двойной щелчок по событию позволяет извлечь сообщение об ошибке (рис. 20.12).

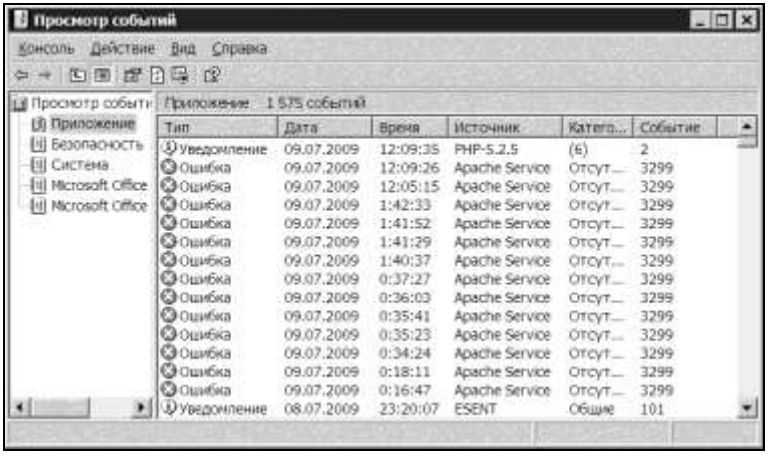


Рис. 20.11. Консоль службы регистрации событий



Рис. 20.12. Сообщение об ошибке

20.1.8. Обработка данных

Директивы обработки данных влияют на предварительную обработку информации, поступающей скрипту извне, а также результата, сгенерированного скриптом, перед отправкой его клиенту. В табл. 20.10 представлен список директив данной группы.

Замечание

Из PHP 6 исключен ряд устаревших директив, таких как `register_globals`, `register_long_arrays` и `magic_quotes_gpc`.

Таблица 20.10. *Директивы обработки данных*

Директива	Описание
`arg_separator.output`	Задает разделитель, используемый в URL, которые генерируются PHP. В HTML для этого используется амперсанд `&`, в XHTML — последовательность `&`
`arg_separator.input`	Задает разделитель, используемый для получения параметров из URL. Не рекомендуется задавать в этом параметре что-то отличное от `&`, т. к. PHP может потерять возможность получать GET- и POST-параметры через соответствующие суперглобальные массивы `$_GET` и `$_POST`
`register_globals`	Включает (`On`) или выключает (`Off`) трансляцию переменных окружения, GET- и POST-данных, а также cookies в глобальные переменные. Данная директива является нерекомендуемой к включению и исключена из PHP 6
`variables_order`	Задает порядок трансляции внешних параметров в глобальные переменные, при включенной директиве `register_globals`. Так как параметры из разных источников могут обладать одинаковыми именами, глобальной переменной присваивается значение из последнего источника. По умолчанию, директива принимает значение EGPCS, означающее, что сначала обрабатываются переменные окружения операционной системы, потом GET-параметры, затем POST-параметры, cookies и в конечном итоге переменные окружения Web-сервера (соответствующие суперглобальному массиву `$_SERVER`)
`request_order`	Задает порядок обработки заполнения суперглобального массива `$_REQUEST` внешними параметрами. Так как параметры из разных источников могут обладать одинаковыми именами, элементу массива `$_REQUEST` с этим ключом присваивается значение из последнего источника. По умолчанию, директива принимает значение `GP`, означающее, что сначала обрабатываются GET-, а затем POST-параметры
`register_long_arrays`	Включает (`On`) или отключает (`Off`) возможность использования длинных массивов. В PHP 3 использовались длинные варианты суперглобальных массивов, так вместо `$_GET` использовался `$HTTP_GET_VARS`, вместо `$_POST` — `$HTTP_GET_VARS`. Директива `register_long_arrays` позволяет включить поддержку старых вариантов суперглобальных массивов. Данная директива является нерекомендуемой к включению и исключена из версии PHP 6

Таблица 20.10 *(продолжение)*

Директива	Описание
`register_argc_argv`	Включает (`On`) или выключает (`Off`) автоматическую регистрацию переменных `$argv` и `$argc`, содержащих соответственно массив с параметрами командной строки и число параметров. Такой режим очень удобен при использовании PHP в качестве интерпретатора системных скриптов, которые часто принимают дополнительные параметры. Однако при выполнении скрипта в условиях Web-сервера эти лишние переменные бесполезны и лишь расходуют ресурсы
`auto_globals_jit`	Если директива включена (`On`), суперглобальный массив `$_SERVER` создается только в момент обращения к нему из скрипта. В противном случае (`Off`) массив `$_SERVER` воссоздается каждый раз перед началом выполнения, независимо от того, используется он в скрипте или нет. Для работы включенного режима `auto_globals_jit` необходимо, чтобы были отключены директивы `register_globals`, `register_long_arrays` и `register_argc_argv`
`post_max_size`	Задает максимальный размер POST-данных, которые может принять PHP-скрипт
`magic_quotes_gpc`	Включает (`On`) или выключает (`Off`) режим "магических кавычек" для внешних данных. Если этот режим включен, PHP экранирует кавычки для любых данных, поступающих извне (GET/POST-параметры, cookies и переменные окружения). С одной стороны, это позволяет автоматически экранировать кавычки, которые могут послужить причиной сбоя или SQL-инъекции при помещении данных в СУБД. С другой стороны, нестандартное экранирование кавычек в разных СУБД, а также непоследовательное применение данной директивы на разных Web-серверах приводит к лишней работе. Приходится либо отключать этот режим из скрипта, либо всякий раз проверять, включен режим или нет. Данная директива является нерекомендуемой к включению и исключена из версии PHP 6
`magic_quotes_runtime`	Включает (`On`) или выключает (`Off`) режим "магических кавычек" для данных, которые поступают от источников в момент выполнения скрипта (например, от СУБД или команды операционной системы, выполненной в обратных кавычках или функции `system()`)
`magic_quotes_sybase`	Включает (`On`) или выключает (`Off`) режим "магических кавычек" с экранированием данных в стиле СУБД Sybase: кавычки `'` экранируются удвоением `''`, вместо `\'`
`auto_prepend_file`	Задает имя файла, который автоматически добавляется в начало любого PHP-файла. Файл вызывается так, будто он был включен при помощи инструкции `include`
`auto_append_file`	Задает имя файла, который автоматически добавляется в конец любого PHP-файла. Файл вызывается так, будто он был включен при помощи инструкции `include`. Если скрипт останавливается при помощи `exit()`, файл не включается

Таблица 20.10 *(окончание)*

Директива	Описание
`default_mimetype`	Задает MIME-тип, отправляемый клиенту в HTTP-заголовке `Content-Type`. По умолчанию, принимает значение `"text/html"`
`default_charset`	Задает кодировку, отправляемую клиенту в HTTP-заголовке `Content-Type`. Чтобы отключить отправку данного HTTP-заголовка, необходимо установить директиве пустое значение

Директива `register_globals` позволяет транслировать GET/POST-параметры, cookies и переменные окружения в глобальные значения. Пусть скрипту передается GET-параметр `id` следующим образом: `index.php?id=17`. В этом случае вывести значение параметра можно при помощи скрипта, представленного в листинге 20.9.

Листинг 20.9. Вывод GET-параметра в окно браузера

```
<?php
  echo $_GET['id'];
?>
```

Если директива `register_globals` принимает значение `On`, вывести значение GET-параметра можно при помощи глобальной переменной `$id` (листинг 20.10).

Листинг 20.10. Вывод GET-параметра `id` при включенной директиве `register_globals`

```
<?php
  echo $id;
?>
```

Несмотря на все удобство такой трансляции глобальных переменных, PHP-сообщество было вынуждено отказаться от такого режима работы из-за его потенциальной опасности. В листинге 20.11 приводится пример системы аутентификации, которая после проверки логина и пароля, передаваемых скрипту методом POST, устанавливает переменной `$access` значение 1, если доступ разрешен, и значение 0, если аутентификация закончилась неудачей.

Листинг 20.11. Гипотетическая система аутентификации

```
<?php
  if ($_POST['name'] == "name" && $_POST['pass'] == "pass")
  {
    $access = 1;
```

```
  }
  if ($access)
  {
    // Доступ к закрытой части документа
  }
?>
```

Если скрипту из листинга 20.11 передать GET- или POST-параметр `access` со значением, отличным от нуля, то аутентификация будет автоматически пройдена, независимо от того, какие логин и пароль были переданы для проверки. Разумеется, брешь в безопасности можно легко устранить, явно инициализировав переменную `$access` в начале скрипта, однако в объемных проектах, состоящих из тысячи файлов, проблема может быть не столько очевидной. В любом случае, начиная с версии PHP 6, директива `register_globals` исключена из файла php.ini.

Еще одной неоднозначной директивой является `magic_quotes_gpc`, позволяющая включить режим "магических кавычек". Директива вводилась для того, чтобы автоматически экранировать кавычки, предотвращая нарушение синтаксиса SQL-запросов и SQL-инъекции при помещении данных, переданных со стороны клиента в базу данных. Однако де-факто директива породила очередную проблему: некоторые виды СУБД (например Sysbase) требуют экранировать одиночную кавычку `'` не обратным слешем `\'`, а повторением кавычки `''`. Директиву можно включить на сервере или отключить в зависимости от используемой СУБД и предпочтений системного администратора. В результате от переносимого PHP-приложения требуется обработка внешних параметров в зависимости от того, включен режим "магических кавычек" или нет. Выяснить это помогает специальная функция `get_magic_quotes_gpc()`, которая возвращает `TRUE`, если режим включен, и `FALSE` в противном случае. В листинге 20.12 представлен типичный код обработки внешних параметров.

> ***ЗАМЕЧАНИЕ***
>
> Функция `mysql_escape_string()` из состава библиотеки mysql осуществляет экранирование кавычек.

Листинг 20.12. Обработка параметров в зависимости от состояния директивы `magic_quotes_gpc`

```
<?php
  ...
  if (!get_magic_quotes_gpc())
  {
    // Режим "магических кавычек" отключен
    $_POST['name'] = mysql_escape_string($_POST['name']);
    $_POST['pass'] = mysql_escape_string($_POST['pass']);
  }
  ...
?>
```

Как видно из листинга 20.12, если режим "магических кавычек" включен, блок `if` не выполняется, если отключен — выполняется. Начиная с PHP 5.3, директива `magic_quotes_gpc` исключается из php.ini, а функция `get_magic_quotes_gpc()` всегда возвращает `FALSE`.

Следует учитывать, что некоторые ранние версии PHP 6 не включали функцию `get_magic_quotes_gpc()`. В этом случае ее необходимо определить, проверив перед этим, не определена ли она ранее (для совместимости с другими версиями PHP) (листинг 20.13).

Листинг 20.13. Заглушка для функции `get_magic_quotes_gpc()`

```
<?php
  if (!function_exists('get_magic_quotes_gpc'))
  {
    function get_magic_quotes_gpc()
    {
      return false;
    }
  }
?>
```

Определенный интерес представляют директивы `auto_prepend_file` и `auto_append_file`, позволяющие подключить соответственно в начало и конец PHP-файла любой другой PHP-документ.

Пусть имеются два PHP-файла: php_start.php (листинг 20.14) и php_end.php (листинг 20.15), которые соответственно выводят фразы "Start PHP" и "End PHP".

> ***Замечание***
>
> В скрипты, подключаемые директивами `auto_prepend_file` и `auto_append_file`, обычно помещают более практичные вещи, такие как библиотеки общих файлов, системы подсчета времени генерации страницы, счетчики посещения и т. п.

Листинг 20.14. Файл php_start.php

```
<?php
  echo "Start PHP<br />";
?>
```

Листинг 20.15. Файл php_end.php

```
<?php
  echo "End PHP<br />";
?>
```

Назначим директиве `auto_prepend_file` скрипт php_start.php, а директиве `auto_append_file` — скрипт php_end.php:

```
auto_prepend_file = D:/php_start.php
auto_append_file = D:/php_end.php
```

Теперь содержимое этих файлов будет подключаться в начале и конце любого скрипта так, как если бы они содержали соответствующие `include`-инструкции (листинг 20.16).

Листинг 20.16. Скрипт index.php

```
<?php
  echo "Contents<br />";
?>
```

Результат выполнения скрипта из листинга 20.16 представлен на рис. 20.13.

Скрипт php_end.php подключается лишь в том случае, если обработчик PHP штатно доходит до конца файла. Если же его работа прерывается, например при помощи `exit()`, файл php_end.php не подключается (листинг 20.17).

Листинг 20.17. Завершение работы скрипта при помощи `exit()`

```
<?php
  echo "Contents<br />";
  exit();
?>
```

Результат выполнение скрипта из листинга 20.17 представлен на рис. 20.14.



Рис. 20.13. Результат выполнения скрипта из листинга 20.16

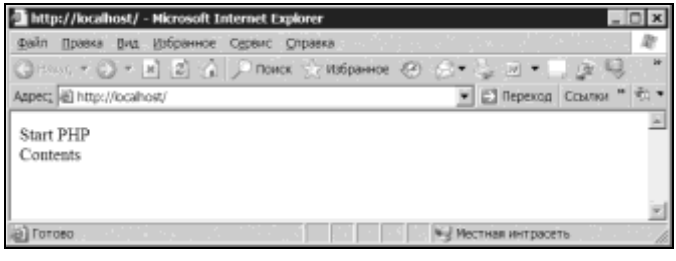


Рис. 20.14. Результат выполнения скрипта из листинга 20.17

20.1.9. Загрузка файлов

Директивы управления загрузкой файлов позволяют настроить параметры, влияющие на обработку файлов, загруженных через поле `file` Web-формы (табл. 20.11).

Таблица 20.11. *Директивы управления загрузкой файлов*

Директива	Описание
`file_uploads`	Включает (`On`) или отключает (`Off`) загрузку файлов по протоколу HTTP (через поле `file`)
`upload_tmp_dir`	Задает путь к каталогу, в который помещаются временные файлы
`upload_max_filesize`	Задает максимальный размер загружаемого на сервер файла

20.1.10. Сетевой доступ

Директивы управления сетевым доступом позволяют настроить параметры, которые влияют на возможности функций, поддерживающих удаленное обращение по сети (табл. 20.12).

Таблица 20.12. *Директивы сетевого доступа*

Директива	Описание
`allow_url_fopen`	Включает (`On`) или отключает (`Off`) возможность обращения файловых функций по сетевому пути (начинающемуся с http://)
`allow_url_include`	Включает (`On`) или отключает (`Off`) возможность обращения инструкций `include` и `require` по сетевому пути (начинающемуся с http://)
`from`	Определяет логин для анонимного FTP-доступа. Традиционно для такого доступа используется e-mail
`user_agent`	Задает пользовательский агент `USER_AGENT`, который отправляет PHP при обращении к URL при помощи любой функции, поддерживающей сетевой доступ
`default_socket_timeout`	Задает время ожидания ответа сервера

20.1.11. Подключение расширений

Помимо ядра языка, PHP поддерживает подключение множества библиотек расширений. Некоторые расширения жестко компилируются вместе с ядром в единый модуль, некоторые подключаются отдельно в виде внешних библиотек. В табл. 20.13 представлены директивы данной группы.

Таблица 20.13. *Директивы управления расширениями*

Директива	Описание
`extension_dir`	Задает путь к каталогу с библиотеками расширений
`extension`	Задает название библиотеки расширения, для каждой из библиотек необходима отдельная директива `extension`

По умолчанию, начиная с PHP 5, все внешние расширения отключены. Это означает, что если имеется потребность работать с базой данных MySQL и динамической графикой (GDLib), следует явно подключить эти расширения в конфигурационном файле php.ini.

За подключение расширений несет ответственность директива `extension`. Для подключения расширения необходимо снять комментарий (символ `#`) напротив его. Так, для подключения расширения для работы с MySQL необходимо снять комментарий напротив директивы:

```
extension=php_mysql.dll
```

> ***Замечание***
>
> Ряд расширений, таких как расширение для шифрования данных php_mcrypt.dll, требует дополнительных динамических библиотек, которые необходимо будет поместить в каталог C:\Windows\system32. Часть библиотек расположена в C:\php, часть придется загрузить из Интернета. Узнать список расширений, требующих сторонних библиотек, а также названия этих библиотек можно в файле C:\php\snapshot.txt.

Помимо этого, необходимо настроить директиву, указав путь к библиотекам расширения:

```
extension_dir = "C:/php/ext/"
```

Можно оставить значение директивы `extension_dir` по умолчанию `./` и скопировать необходимые библиотеки из каталога C:\php\ext в каталог C:\php\. Этот прием полезен, когда PHP не может найти динамические библиотеки (рис. 20.15).

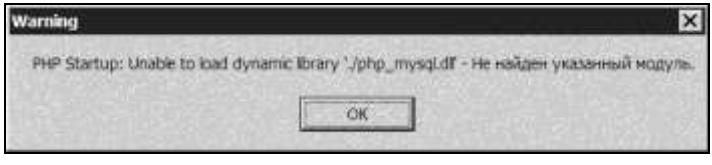


Рис. 20.15. PHP не может загрузить расширение php_mysql.dll

20.1.12. Настройка сессии

В секции `[Session]` конфигурационного файла php.ini сосредоточены директивы, управляющие поведением сессии. В табл. 20.14 приводится список наиболее часто используемых директив.

Таблица 20.14. *Директивы управления сессией*

Директива	Описание
`session.save_handler`	Задает обработчик механизма сессий, по умолчанию директива принимает значение `files` (сохранение данных сессии в файле в стерилизованном виде). Данная директива предназначена для смены механизма сессии (если он реализован в виде альтернативного расширения)
`session.save_path`	Задает путь к каталогу, в который сохраняются файлы сессии
`session.use_cookies`	Включает (`On`) или отключает (`Off`) использование cookies для хранения *уникального идентификатора сессии* SID. Если директива отключена, SID передается через GET-параметр
`session.cookie_secure`	Если директива включена (`On`), cookies с SID сессии могут передаваться только через защищенный канал (https:// протокол HTTP поверх SSL-протокола)
`session.use_only_cookies`	Если директива включена (`On`), PHP использует только cookies для передачи SID сессии, передача их через GET-параметры запрещается
`session.name`	Задает имя сессии по умолчанию, если оно не задается явно функцией `session_name()`
`session.auto_start`	Включает (`On`) или отключает (`Off`) автоматический старт сессии в каждом PHP-файле. В отключенном состоянии сессию необходимо инициировать самостоятельно при помощи функции `session_start()`
`session.cookie_lifetime`	Задает время жизни cookies с SID. Если директива принимает значение 0, используются сессионные cookies, которые сохраняются до момента выключения браузера
`session.cookie_path`	Задает область действия Cookie с SID на сайте, по умолчанию принимает значение /, означающее, что Cookie действуют начиная с корня, и охватывает весь сайт
`session.cookie_domain`	Задает домен действия cookies
`session.serialize_handler`	Задает обработчик сериализации сессии. По умолчанию принимает значение `php`, означающее, что для сериализации используется стандартная функция `serialize()`
`session.gc_probability`	Задает вероятность старта процесса "сборки мусора" при очередной инициализации сессии. Вероятность определяется отношением `session.gc_probability`/`session.gc_divisor`. По умолчанию директива принимает значение 1

Таблица 20.14 *(окончание)*

Директива	Описание
session.gc_divisor	Задает вероятность старта процесса "сборки мусора" при очередной инициализации сессии. Вероятность определяется отношением session.gc_probability/session.gc_divisor. По умолчанию директива принимает значение 100
session.gc_maxlifetime	Задает время, после которого сохраненные данные рассматриваются как устаревшие и подлежащие уничтожению "сборщиком мусора". По умолчанию директива принимает значение 1440

20.1.13. Настройка даты и времени

Директивы группы настройки даты и времени располагаются в секции [Date] (табл. 20.15).

> ***ЗАМЕЧАНИЕ***
>
> Параметры из табл. 20.15 используются функциями date_sunrise(), date_sunset() и date_sun_info() *(см. разд. 11.2)*.

Таблица 20.15. *Директивы управления датой и временем*

Директива	Описание
date.timezone	Задает часовой пояс по умолчанию
date.default_latitude	Задает широту по умолчанию
date.default_longitude	Задает долготу по умолчанию
date.sunrise_zenith	Задает зенит (90° для точки горизонта) по умолчанию для функции date_sunrise()
date.sunset_zenith	Задает зенит (90° для точки горизонта) по умолчанию для функции date_sunset()

20.2. Изменение настроек php.ini средствами Apache

В предыдущем разделе были рассмотрены наиболее часто редактируемые директивы конфигурационного файла php.ini. Сложность заключается в том, что конфигурационный файл php.ini присутствует в единственном экземпляре, а сервер может поддерживать сотни сайтов, для каждого из которых могут требоваться различные настройки. Одним из решений данной проблемы является настройка директив конфигурационного файла php.ini при помощи конфигурационных файлов httpd.conf и .htaccess.

Для того чтобы иметь возможность изменять настройки файла php.ini из конфигурационных файлов httpd.conf и .htaccess, необходимо, чтобы директива `AllowOverride` контейнера `<Directory />` имела значение `Options` или `All` (листинг 20.18).

Листинг 20.18. Настройка директивы `AllowOverride` в httpd.conf

```
<Directory />
    Options FollowSymLinks
    AllowOverride All
    Order deny,allow
    Satisfy all
</Directory>
```

Конфигурационный файл http.conf осуществляет глобальную настройку Web-сервера Apache, в то время как настройки конфигурационного файла .htaccess действуют на текущий каталог (а также вложенные в него подкаталоги).

Установка настроек PHP из конфигурационных файлов httpd.conf и .htaccess осуществляется четырьмя директивами:

- `php_admin_value` — устанавливает строковый параметр php.ini;
- `php_admin_flag` — устанавливает логические параметры php.ini (т. е. те, которые принимают значения `On` или `Off`);
- `php_value` — устанавливает строковый параметр php.ini;
- `php_flag` — устанавливает логические параметры php.ini (т. е. те, которые принимают значения `On` или `Off`).

Директивы `php_admin_value` и `php_admin_flag` в отличие от `php_value` и `php_flag`, допускаются к использованию только в конфигурационном файле httpd.conf. Параметр, установленный при помощи данных директив, не может быть изменен впоследствии при помощи конфигурационного файла .htaccess. Директивы `php_value` и `php_flag` могут быть использованы везде, а установленные ими значение переназначены.

В листинге 20.19 приводится пример установки директиве `open_basedir` значения "/www". Таким образом, область действия PHP ограничивается каталогом /www, и никто кроме администратора сервера не может изменить эту настройку.

Листинг 20.19. Ограничение действия PHP каталогом /www

```
<Directory /www>
  php_admin_value open_basedir /www
</Directory>
```

Как правило, директиву `open_basedir` применяют более гибко, поместив ее в виртуальном хосте сайта и передав в качестве значения путь к каталогу, где расположен виртуальный хост, можно добиться поведения PHP, при котором скрипты одного сайта не имеют доступа к файлам соседнего виртуального хоста (листинг 20.20).

Листинг 20.20. Разграничение виртуальных хостов

```
# Первый виртуальный хост
<VirtualHost *:80>
    ServerAdmin suport@domain1.ru
    DocumentRoot "/www/domain1/"
    ServerName domain1.ru
    php_admin_value open_basedir "/www/domain1/"
    ErrorLog "/www/domain1/logs/domain1-error.log"
    CustomLog "/www/domain1/logs/domain1-access.log" common
</VirtualHost>
# Второй виртуальный хост
<VirtualHost *:80>
    ServerAdmin suport@domain2.ru
    DocumentRoot "/www/domain2/"
    ServerName domain2.ru
    php_admin_value open_basedir "/www/domain2/"
    ErrorLog "/www/domain1/logs/domain2-error.log"
    CustomLog "/www/domain1/logs/domain2-access.log" common
</VirtualHost>
```

Для изменения на уровне .htaccess удобнее воспользоваться директивами `php_value` и `php_flag`. Удобство использования конфигурационного файла .htaccess заключается в том, что для его применения не требуется редактирование файла httpd.conf, к которому имеет доступ только администратор хостинга. Использование .htaccess, как правило, доступно любому разработчику, имеющему FTP-доступ к сайту.

В листинг 20.21 приводится пример включения директивы `register_long_arrays`, разрешающей использование длинных вариантов суперглобальных массивов, а также директивы `auto_prepend_file`, подключающей к каждому PHP-файлу библиотеку include.utils.php.

> ***Замечание***
>
> При использовании директивы `auto_prepend_file` следует указывать полный путь к включаемому файлу от корня диска.

Листинг 20.21. Конфигурационный файл .htaccess

```
php_flag register_long_arrays on
php_value auto_prepend_file /www/domain/include/include.utils.php
```

20.3. Функции управления интерпретатором PHP

Несмотря на то, что конфигурационные файлы Apache позволяют достаточно гибко изменять параметры php.ini, они лишены динамичности PHP-кода и не могут использовать условные выражения. Поэтому PHP предоставляет разработчикам широкий спектр функций, позволяющих влиять на параметры PHP непосредственно из скриптов (табл. 20.16).

> ***ЗАМЕЧАНИЕ***
>
> Функции, влияющие на обработку ошибок, рассмотрены в *главе 24*.

Таблица 20.16. *Функции управления интерпретатором PHP*

Директива	Описание
`ini_set($varname, $newvalue)`	Устанавливает директиве `$varname` новое значение `$newvalue`. В случае успеха возвращает старое значение директивы. Если установить новое значение не удается, возвращается `FALSE`
`ini_get($varname)`	Возвращает текущее значение директивы `$varname`
`ini_get_all($extension)`	Возвращает ассоциативный массив с директивами расширения `$extension`
`ini_restore($varname)`	Восстанавливает значение директивы `$varname`, присваивая ей значение, установленное в конфигурационном файле php.ini

В листинге 20.22 демонстрируется типичное использование функций управления интерпретатором PHP.

Листинг 20.22. Использование функций управления интерпретатором PHP

```
<?php
  // Читаем содержимое директивы post_max_size
  echo ini_get("post_max_size");
  // Устанавливаем новое значение post_max_size
  ini_set("post_max_size", "32M");
  // Восстанавливаем первоначальное значение
  ini_restore("post_max_size");
?>
```

Если необходимо запретить изменение директив из скрипта, следует либо воспользоваться безопасным режимом *(см. разд. 20.1.2)*, либо определить окончательное значение директивы при помощи директивы `php_admin_value` на уровне конфигурационного файла httpd.conf *(см. разд. 20.2)*.

Помимо функций, позволяющих явно управлять значениями директив конфигурационного файла, PHP предоставляет множество справочных и специализированных функций по конкретным директивам (табл. 20.17).

***Таблица 20.17.** Справочные функции*

Директива	Значение
`memory_get_usage()`	Возвращает количество байтов, которые занимает текущий скрипт
`php_sapi_name()`	Возвращает тип подключения PHP к Web-серверу. Если PHP подключен как внешнее CGI-приложение, возвращается строка `"cgi"`, если как модуль — строка `"apache"`
`php_uname([$mode])`	Возвращает информацию об операционной системе, в которой выполняется PHP. Необязательный параметр `$mode` принимает значения, совпадающие с параметрами утилиты `uname`
`phpinfo()`	Возвращает подробную информацию о PHP, его расширениях, установленных переменных окружения в виде PHP-страницы
`phpversion([$extension])`	Возвращает версию PHP. Можно узнать версию отдельного расширения, если передать его имя при помощи необязательного параметра `$extension`

Используя функцию `memory_get_usage()`, можно отслеживать количество занятой скриптом оперативной памяти, чтобы не превысить предел, заданный директивой `memory_limit`, и не привести к возникновению критической ошибки.

Функция `php_uname()` возвращает информацию об операционной системе, в которой выполняется PHP-скрипт. При помощи единственного необязательного параметра можно задать тип возвращаемой информации:

- `'s'` — название операционной системы;
- `'n'` — имя хоста;
- `'r'` — версия PHP;
- `'v'` — версия операционной системы;
- `'m'` — архитектура машины, например, i386;
- `'a'` — включает всю вышеперечисленную информацию.

В листинге 20.23 приводится пример использования функции `phpinfo()`, позволяющей получать подробнейшую информацию о текущей версии PHP в удобной для анализа форме.

Листинг 20.23. Использование функции `phpinfo()`

```
<?php
  phpinof();
?>
```

Результат работы функции `phpinfo()` представлен на рис. 20.16.



Рис. 20.16. Результат работы функции `phpinfo()`



Глава 21

Объекты и классы

Классы и объекты являются ключевыми конструкциями объектно-ориентированной технологии. В этой главе будет подробно рассмотрен процесс создания и использования классов и объектов.

Последние два десятилетия в IT-индустрии получил широкое распространение объектно-ориентированный подход. Его введение связано со все возрастающим объемом программных систем, с которыми приходится сталкиваться разработчикам.

> ***ЗАМЕЧАНИЕ***
>
> Данная глава кратко рассматривает объектно-ориентированные возможности PHP. Всестороннему рассмотрению предмета посвящена наша отдельная книга "Объектно-ориентированное программирование на PHP".

21.1. Введение в объектно-ориентированное программирование

Независимо от языка программирования объектно-ориентированный подход имеет ряд общих принципов, а именно:

- возможность создавать *абстрактные типы данных*, позволяющая наряду с предопределенными типами данных (такими как `integer`, `bool`, `double`, `string`) вводить собственные типы данных (классы) и объявлять "переменные" таких типов данных (объекты). Создавая свои типы данных, программист оперирует не машинными терминами (переменная, функция), а объектами реального мира, поднимаясь тем самым на новый абстрактный уровень. Яблоки и людей нельзя умножать друг на друга, однако низкоуровневый код запросто позволит совершить такую логическую ошибку, тогда как при использовании абстрактных типов данных такая операция становится невозможной;
- *инкапсуляция*, допускающая взаимодействие пользователя с абстрактными типами данных только через их интерфейс и скрывающая внутреннюю реализацию объекта, не допуская влияния на его внутреннее состояние. Память человека ограничена и не может содержать все детали огромного проекта, тогда как использование инкапсуляции позволяет разработать объект и использовать его, не заботясь о внутренней реализации, прибегая только к небольшому числу интерфейсных методов;

- *наследование*, позволяющее развить существующий абстрактный тип данных — класс, создав на его основе новый класс. При этом новый класс автоматически получает возможности уже существующего класса. Зачастую абстрактные типы данных слишком сложны, поэтому прибегают к их последовательной разработке, выстраивая иерархию классов от общего к частному;
- *полиморфизм*, допускающий построение целых цепочек и разветвленных деревьев, наследующих друг другу абстрактных типов данных (классов). При этом весь набор классов будет иметь ряд методов с одинаковыми названиями: любой из классов данного дерева гарантированно обладает методом с таким именем. Этот принцип помогает автоматически обрабатывать массивы данных разного типа.

Абстрактные типы данных необходимы для того, чтобы дать программисту вводить в программу переменные с желаемыми свойствами, т. к. возможностей существующих в языке типов данных зачастую не хватает. Связи между объектами реального мира зачастую настолько сложны, что для их эффективного моделирования необходим отдельный язык программирования. Разрабатывать специализированный язык программирования для каждой прикладной задачи — очень дорогое удовольствие. Поэтому в языки программирования вводится объектно-ориентированный подход, который позволяет создавать свой мини-язык путем создания классов и их объектов. Переменными такого мини-языка программирования являются программные *объекты*, в качестве типа для которых выступает класс. *Класс* описывает состав объекта — переменные и функции, которые обрабатывают переменные и тем самым определяют поведение объекта (рис. 21.1).

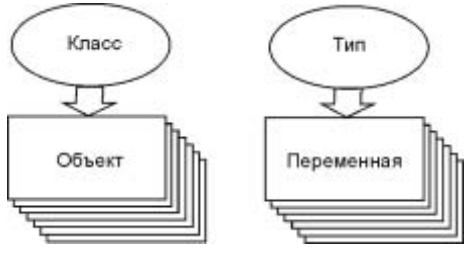


Рис. 21.1. Переменные объявляются при помощи типа, объекты при помощи класса

21.2. Создание класса

Класс объявляется при помощи ключевого слова `class`, после которого следует уникальное имя класса и тело класса в фигурных скобках. В теле класса объявляются переменные и функции класса, которые соответственно называются *методами* и *членами*. В листинге 21.1 приводится общий синтаксис объявления класса.

Листинг 21.1. Объявление класса

```
<?php
  class имя_класса
  {
    // Члены и
    // методы класса
  }
?>
```

Важной особенностью PHP является то, что PHP-скрипты могут включаться в документ при помощи тегов <?php и ?>. Один документ может содержать множество включений этих тегов, однако класс должен объявляться в одном неразрывном блоке <?php и ?>. Попытка разорвать объявление класса приводит к генерации интерпретатором ошибки разбора "Parse error: parse error, unexpected ';', expecting T_FUNCTION".

Так как прерывать объявление класса недопустимо, его не удастся механически разбить и при помощи инструкций include(), include_once(), require(), require_once(). Допускается, однако, использование этих конструкций внутри методов.

> ***ЗАМЕЧАНИЕ***
>
> Напомним, что при помощи инструкций include(), include_once(), require(), require_once() можно включать в состав PHP-скриптов другие PHP-скрипты. Это позволяет разбивать объемные многострочные файлы на множество мелких файлов, которые программисту проще воспринять. При отсутствии включаемого файла инструкция include() генерирует предупреждение, однако не останавливает работу скрипта, в то время как require() в этом случае аварийно завершает работу приложения. Допускается множественное включение файлов друг в друга, что может приводить к запутанным ситуациям и многократному включению файлов в приложение. Суффикс once означает, что файл будет включен лишь один раз, и повторный вызов инструкции include_once() или require_once() игнорируется. Это особенно удобно для включения библиотек функций и классов, повторное объявление которых вызывает ошибку.

21.3. Создание объекта

Объект объявляется при помощи ключевого слова new, за которым следует имя класса. В листинге 21.2 создается пустой класс emp и объявляется объект $obj данного класса.

> ***ЗАМЕЧАНИЕ***
>
> После имени класса могут следовать необязательные круглые скобки.

Листинг 21.2. Создание класса emp и объявление объекта $obj данного класса

```
<?php
  class emp {}
  $obj = new emp;
?>
```

Замечание

Имена классов, в отличие от имен переменных и объектов, не зависят от регистра, поэтому можно использовать для объявления классов имена `emp`, `Emp()`, `EMP()`. Однако использование вместо объекта `$obj` переменной `$Obj` приведет к ошибке — интерпретатор PHP будет считать, что в программу введена новая переменная.

Объект `$obj` является обычной переменной PHP, правда, со специфичными свойствами. Как и любую другую переменную, объект можно передавать в качестве параметра функции, использовать как элемент массива или присваивать ему новое значение. В листинге 21.3 приводится пример, в котором объекту `$obj` по мере выполнения программы присваивается некоторое числовое значение, и `$obj` становится переменной числового типа.

Листинг 21.3. Объект — это обычная переменная

```
<?php
  class emp {}
  $obj = new emp();
  $obj = 3;
  echo $obj; // 3
?>
```

Объект существует до конца времени выполнения скрипта или пока не будет уничтожен явно при помощи конструкции `unset()` (листинг 21.4). Использование конструкции `unset()` может быть полезным, если объект занимает большой объем оперативной памяти и ее следует освободить, чтобы не допустить переполнения.

Листинг 21.4. Создание и уничтожение объекта

```
<?php
  // Объявление класса
  class emp {}
  // Создание объекта класса
  $obj = new emp();
  // Уничтожение объекта
  unset($obj);
?>
```

В отличие от других языков программирования объект, объявленный внутри блока, ограниченного фигурными скобками, существует и вне его, не подвергаясь уничтожению при выходе из блока.

21.4. Инкапсуляция. Спецификаторы доступа

Как было показано в предыдущем разделе, класс может использоваться без методов и членов, однако пользы в этом случае от него не больше, чем от переменной, которая не может хранить значения. Как правило, классы содержат члены и методы. Часть из них будет использоваться внешним разработчиком, часть предназначена только для внутреннего использования в рамках класса. PHP предоставляет специальные ключевые слова — *спецификаторы доступа*, позволяющие указать, какие члены и методы доступны извне, а какие нет. Это позволяет реализовать принцип инкапсуляции.

Открытые члены класса объявляются спецификатором доступа `public` и доступны как методам класса, так и внешнему по отношению к классу коду. *Закрытые* методы и члены класса объявляются при помощи спецификатора `private` и доступны только в рамках класса; обратиться к ним извне невозможно.

> ***Замечание***
>
> Помимо спецификаторов `public` и `private` в PHP поддерживается спецификатор `protected`, используемый при наследовании и подробно рассматриваемый далее. В более ранних версиях для объявления члена класса использовалось ключевое слово `var`. Члены, объявленные с его помощью, были открытыми. В текущих версиях PHP по-прежнему допускается использовать ключевое слово `var` для объявления членов класса, однако это не рекомендуется, а само ключевое слово признано устаревшим. С большой долей вероятности оно будет исключено из PHP 6.

В листинге 21.5 представлен класс `employee` (сотрудник), который содержит четыре члена:

- `$surname` — фамилия сотрудника;
- `$name` — имя сотрудника;
- `$patronymic` — отчество сотрудника;
- `$age` — возраст сотрудника.

Листинг 21.5. Класс `employee`

```
<?php
  class employee
  {
    public $surname;
    public $name;
    public $patronymic;
    private $age;
  }
?>
```

Рассмотрим пример. Пусть класс `employee` располагается в файле class.employee.php. В листинге 21.6 объявляется объект `$emp` класса `employee`, при этом члены класса получают необходимые значения и выводятся в окно браузера.

Листинг 21.6. Обращение к членам класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  // Присваиваем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  // $emp->age = 23; // Ошибка

  // Выводим члены класса
  echo $emp->surname." ".$emp->name." ".$emp->patronymic."<br>"
?>
```

Как видно из листинга 21.6, при обращении к члену класса используется оператор `->`, после которого следует ввести имя этого члена. При обращении к членам класса не добавляется символ `$`.

Обращение к закрытому члену класса `$age` завершится ошибкой "Fatal error: Cannot access private property employee::$age". На рис. 21.2 приводится схематическое изображение процесса доступа к членам объекта, снабженным разными спецификаторами доступа.

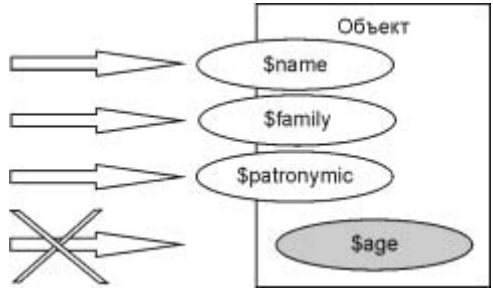


Рис. 21.2. Открытые и закрытые члены объекта

21.5. Методы класса. Член *$this*

Популярность объектно-ориентированного подхода заключается в том, что классы представляют собой не просто контейнеры для хранения переменных (в качестве таких контейнеров в PHP могут выступать ассоциативные массивы), но также позволяют включать методы, обрабатывающие как открытые, так и закрытые члены класса.

Метод класса представляет собой обычную функцию PHP, которую предваряет один из спецификаторов доступа. В листинге 21.7 приводится пример класса `cls_mth`, в состав которого входит метод `show_message()`. Задача метода сводится к выводу в окно браузера сообщения "Hello, world!".

> ***Замечание***
>
> В листинге 22.7 можно опустить спецификатор доступа `public` перед методом `show_message()`. В отличие от других объектно-ориентированных языков, если в PHP не уточняется спецификатор, считается, что метод объявлен со спецификатором `public`. Именно потому, что в разных языках программирования используются разные подходы, не рекомендуется опускать спецификатор доступа при объявлении метода.

Листинг 21.7. Объявление метода класса

```
<?php
  class cls_mth
  {
    public function show_message()
    {
      echo "Hello, world!";
    }
  }

  $obj = new cls_mth();

  $obj->show_message();
?>
```

В листинге 21.7 метод класса не использует никаких членов, однако, как правило, методы вводят именно для обработки членов класса. Для обращения к любому элементу класса внутри метода следует применять конструкцию `$this->`. Член `$this`, который неявно присутствует в каждом классе, является ссылкой на текущий объект класса.

Вернемся к классу `employee`, определенному в листинге 21.5. Класс содержит закрытую переменную `$age`, определяющую возраст сотрудника. Создадим четыре метода (листинг 21.8):

- `get_age()` — метод, возвращающий значение закрытого члена `$age`;
- `set_age()` — метод, возвращающий значение закрытого члена `$age` и проверяющий его принадлежность к числовому типу данных и промежутку от 18 до 65;
- `get_info()` — метод, возвращающий фамилию и имя сотрудника;
- `get_full_info()` — метод, возвращающий фамилию, имя и возраст сотрудника.

> ***Замечание***
>
> Следует обратить внимание на то, что открытые методы и члены расположены в начале класса, а закрытые — в конце. Это негласное правило позволяет сосредоточить внимание программиста в первую очередь на открытом интерфейсе.

Листинг 21.8. Использование открытых методов для доступа к закрытому члену класса

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      return $this->age;
    }
    public function set_age($val)
    {
      $val = intval($val);
      if ($val >= 18 && $val <= 65)
      {
         $this->age = $val;
         return true;
      }
      else return false;
    }
    public function get_info()
    {
      return $this->surname." ".$this->name." ".$this->patronymic;
    }
    public function get_full_info()
    {
      return "{$this->get_info()} ({$this->get_age()})";
    }

    // Закрытые члены
    private $age;
  }
?>
```

Подход, представленный в листинге 21.8, позволяет удостовериться, что закрытый член `$age` получил корректное значение. Рассмотрим подробнее методы `get_age()` и `set_age()`. Метод `get_age()` не принимает ни одного параметра — единственная

его задача — обратиться к закрытому члену `$age` и вернуть это значение вызвавшему его коду. На первый взгляд, такой подход может показаться избыточным — вместо непосредственного обращения к члену `$age` вводится функция-посредник, на вызов которой затрачиваются лишние ресурсы. Однако лишние затраты на проектирование методов доступа к закрытым членам, а также вызов функций вместо прямого обращения с лихвой окупаются в дальнейшем. Например, для вычисления текущего возраста сотрудника на основании сведений из базы данных, актуальных на момент заполнения анкеты (вероятно, несколько лет назад), не придется менять весь код, использующий класс `employee`: достаточно изменить внутреннюю реализацию метода `get_age()`, после чего изменения автоматически отразятся на всей системе.

При проектировании методов, которые обращаются к внутренним членам класса, необходимо следить за тем, что к членам класса следует обращаться через префикс `$this->`, а при обращении к параметрам метода данный префикс не требуется. В большинстве случаев интерпретатор PHP не сообщит о наличии ошибок, если при обращении к внутреннему члену класса не используется префикс `$this->`, а при обращении к параметру метода префикс `$this->` используется. При этом будет создана новая переменная с нулевым значением. Эта ошибка очень распространена и связана с фундаментальной особенностью PHP — отсутствием типизации: использование переменной с любым именем приводит к созданию переменной с этим именем.

> ***Замечание***
>
> Для того чтобы заставить PHP сообщать о попытках обращения к необъявленным переменным, необходимо в конфигурационном файле php.ini добавить к директиве `error_reporting` константу `E_NOTICE` для отображения замечаний или включить отображение всех видов ошибок, предупреждений и замечаний, установив значение директивы в `E_ALL`.

Метод `set_age()` из листинга 21.8 принимает единственный параметр `$val`, который при помощи функции `intval()` приводится к числовому типу. После этого осуществляется проверка, принадлежит ли параметр `$val` интервалу от 18 до 65. Если параметр удовлетворяет этому условию, закрытый член класса `$this->age` получает новое значение, а метод возвращает значение `true`, свидетельствующее об успешном выполнении операции. Если новое значение не удовлетворяет условию, установленному для члена `$this->age`, метод `set_age()` возвращает значение `false`, говорящее о неудачном завершении операции.

Следует обратить внимание на формирование строк в методах `get_info()` и `get_full_info()`. Строки в PHP объединяются при помощи символа точки (`.`), при этом все типы, отличные от строкового, приводятся к нему автоматически. Однако это не единственный способ формирования строки: можно прибегнуть к так называемой *интерполяции*, при которой переменная вставляется в строку, заключенную в двойные кавычки. Например, переменная `$name = "Hello"`, подставленная в строку `"$name, world!"`, приводит к формированию строки `"Hello, world!"`.

> ***Замечание***
>
> В одиночных кавычках значение переменной не интерполируется, поэтому строка `'$name, world!'` отображается как есть — `'$name, world!'`.

Аналогично можно подставлять в строку значения членов класса. Однако конструкция `$this->имя_члена` достаточно сложна для интерпретации. Для того чтобы интерпретатор мог корректно различить обращение к члену класса из строки, необходимо заключить переменную в фигурные скобки, как это продемонстрировано в методе `get_full_info()`.

В листинге 21.9 приводится пример использования класса `employee` из листинга 21.8.

Листинг 21.9. Использование класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  // Передаем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  if (!$emp->set_age(23)) exit("Ошибка вычисления возраста");

  echo $emp->get_full_info(); // Борисов Игорь Иванович (23)
?>
```

21.6. Дамп объекта

При отладке объектно-ориентированных приложений зачастую требуется проанализировать текущее состояние объекта. Для этого предусмотрена функция `print_r()`. В листинге 21.10 с ее помощью выводится структура объекта `$emp` класса `employee`. Для удобства восприятия вызов функции `print_r()` обрамляется HTML-тегами `<pre>` и `</pre>`, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Листинг 21.10. Получение структуры объекта класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
```

```
  $emp->patronymic = "Иванович";
  $emp->set_age(23);

  // Выводим структуру объекта
  echo "<pre>";
  print_r($emp);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.10 являются следующие строки:

```
employee Object
(
    [surname] => Борисов
    [name] => Игорь
    [patronymic] => Иванович
    [age:private] => 23
)
```

Таким образом, метод возвращает все члены объекта, в том числе и закрытые. Закрытые члены помечаются спецификатором `private`.

> ***ЗАМЕЧАНИЕ***
>
> Функцию `print_r()` можно использовать для просмотра не только структуры объектов, но и структуры массивов (как ассоциативных, так и не ассоциативных), а также для вывода значения обычной переменной.

21.7. Специальные методы класса

Помимо методов, создаваемых разработчиком класса, объектно-ориентированная модель PHP предоставляет разработчику специальные методы. Специальные методы позволяют разработчику задать неявные свойства поведения объектов и выполняются, как правило, автоматически.

В табл. 21.1 представлен список специальных методов, предоставляемых объектно-ориентированной моделью PHP. Более подробно каждый из методов обсуждается в последующих разделах.

> ***ЗАМЕЧАНИЕ***
>
> Каждый специальный метод предваряется двумя символами подчеркивания.

Таблица 21.1. *Специальные методы классов*

Метод	Описание
`__construct()`	Конструктор класса; метод, который автоматически выполняется в момент создания объекта до вызова всех остальных методов класса

***Таблица 21.1** (окончание)*

Метод	Описание
`__destruct()`	Деструктор класса; метод, который автоматически выполняется в момент уничтожения объекта
`__autoload()`	Перегружаемая функция, не являющаяся методом класса; позволяет автоматически загружать класс при попытке создания его объекта
`__set()`	Аксессор; метод, предназначенный для установки значения члену класса
`__get()`	Аксессор; метод, предназначенный для чтения значения члена класса
`__isset()`	Позволяет задать логику проверки существования свойства при помощи конструкции `isset()`
`__unset()`	Позволяет задать логику удаления свойства при помощи конструкции `unset()`
`__call()`	Позволяет задать динамический метод
`__toString()`	Позволяет интерполировать (подставлять) объект в строку
`__set_state()`	Позволяет осуществить экспорт объекта
`__clone()`	Позволяет управлять клонированием объекта
`__sleep()`	Позволяет управлять поведением объекта при его сериализации при помощи функции `serialize()`
`__wakeup()`	Позволяет управлять поведением объекта при его восстановлении из сериализованного состояния при помощи функции `unserialize()`

21.8. Функции для работы с методами и классами

Помимо методов, входящих в состав классов, PHP предоставляет большое количество внешних функций, облегчающих работу с классами и объектами (табл. 21.2); часть из них более подробно обсуждаются в последующих разделах данной главы.

***Таблица 21.2.** Список функций для работы с классами и объектами*

Функция	Описание
`call_user_method_array(`*`$method_name`*`, $obj [, $par])`	Осуществляет вызов метода *`$method_name`* объекта `$obj` с массивом параметров `$par`. Функция признана устаревшей и будет исключена из последующих версий языка PHP; вместо нее рекомендуется использовать функцию `call_user_func_array()`

Таблица 21.2 *(продолжение)*

Функция	Описание
call_user_func_array($*methor_name*, $*par*)	Осуществляет вызов метода или функции $*method_name* с массивом параметров $*par*
call_user_method($*method_name*, $*obj* [, $*par* [, $*par1*, ...]])	Осуществляет вызов метода $*method_name* объекта $*obj* с параметрами $*par*, $*par1* и т. д. Функция признана устаревшей и будет исключена из последующих версий языка PHP; вместо нее рекомендуется использовать функцию call_user_func()
call_user_func($*method_name* [, $*par* [, $*par1*, ...]])	Осуществляет вызов метода или функции $*method_name* с параметрами $*par*, $*par1* и т. д.
class_exists($*class_name*)	Возвращает TRUE, если класс с именем $*class_name* объявлен, и FALSE — в противном случае
get_class_methods($class_name)	Возвращает массив с именами методов класса $*class_name*
get_class_vars($*class_name*)	Возвращает массив с именами и значения членов класса $*class_name*
get_class($*obj*)	Возвращает имя класса, которому принадлежит объект $*obj*
get_declared_classes()	Возвращает массив с именами объявленных классов
get_declared_interfaces()	Возвращает массив с именами объявленных интерфейсов
get_object_vars($*obj*)	Возвращает массив с именами и значениями членов объекта $*obj*
get_parent_class($*obj*)	Возвращает имя базового класса для объекта или класса $obj
interface_exists($*interface_name* [, $*autoload*])	Возвращает TRUE, если интерфейс с именем $*interface_name* существует, и FALSE — в противном случае. Если необязательный параметр $*autoload* равен TRUE, функция пытается загрузить интерфейс при помощи функции __autoload()
is_a($*obj*, $*class_name*)	Принимает в качестве первого параметра объект $*obj*, а в качестве второго — имя базового класса $*class_name* и возвращает TRUE, если объект является экземпляром класса $*class_name* или экземпляром его потомка, и FALSE — в противном случае

***Таблица 21.2** (окончание)*

Функция	Описание
`is_callable($arr [, $syntax_only [, $callable_name]])`	Определяет, может ли быть вызван метод класса
`is_object ($obj)`	Возвращает `TRUE`, если переменная `$obj` является объектом, и `FALSE` — в противном случае
`is_subclass_of($obj, $class_name)`	Принимает в качестве первого параметра объект `$obj`, а в качестве второго — имя базового класса `$class_name` и возвращает `TRUE`, если объект является экземпляром потомка класса `$class_name`, и `FALSE` — в противном случае
`method_exists($obj, $method_name)`	Возвращает `TRUE`, если объект `$obj` обладает методом `$method_name`, и `FALSE` — в противном случае
`print_r($obj [, $return])`	Возвращает дамп объекта `$obj`. Если переменная `$return` принимает значение `TRUE`, функция возвращает результат в виде строки, в противном случае результат выводится непосредственно в окно браузера
`property_exists($class_name, $var)`	Возвращает `TRUE`, если переменная `$var` является членом класса `$class_name`, и `FALSE` — в противном случае

21.9. Конструктор. Метод *__construct()*

Конструктор — это специальный метод класса, который автоматически выполняется в момент создания объекта до вызова всех остальных методов класса. Данный метод используется главным образом для инициализации объекта, обеспечивая согласованность его членов.

Для объявления конструктора в классе необходимо создать метод с именем `__construct()`. В листинге 21.11 приводится пример объявления класса `cls`, содержащего конструктор, который выводит сообщение "Вызов конструктора" и инициализирует закрытый член `$var`.

> ***Замечание***
>
> В предыдущих версиях PHP имя конструктора совпадало с именем класса. В рамках обратной совместимости в PHP допускается использование старого метода объявления конструктора, однако такой подход признан устаревшим и может быть исключен из следующих версий языка.

Листинг 21.11. Использование конструктора

```
<?php
  class cls
  {
    private $var;
    public function __construct()
    {
      echo "Вызов конструктора<br>";
      $this->var = 100;
    }
  }

  $obj = new cls();

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.11 являются следующие строки:

```
Вызов конструктора
cls Object
(
    [var:private] => 100
)
```

Важно понимать, что вызов конструктора производится автоматически во время выполнения оператора `new`. Это позволяет разработчику класса быть уверенным, что члены класса получат корректную инициализацию. Создавать специальный метод для инициализации объекта считается дурным тоном — внешний программист может забыть его вызвать.

Обычно в объектно-ориентированных языках программирования явный вызов конструктора вообще не допускается, поскольку это противоречит принципу инкапсуляции, однако в PHP конструктор можно вызвать не только в самом классе, но и из внешнего кода (листинг 21.12).

Листинг 21.12. Вызов конструктора внутри класса и из внешнего кода

```
<?php
  class cls
  {
    private $var;
```

```
    public function __construct()
    {
      echo "Вызов конструктора<br>";
      $this->var = 100;
    }
    public function anarhist()
    {
      $this->__construct();
    }
  }

  $obj = new cls();
  $obj->__construct();
  $obj->anarhist();
?>
```

Представленный в листинге 21.12 скрипт вернет следующие строки:

```
Вызов конструктора
Вызов конструктора
Вызов конструктора
```

В первый раз конструктор вызывается неявно при создании объекта `$obj`, во второй раз — явно (метод является открытым), в третий раз вызов происходит из метода `anarhist()`. Следует избегать манипулирования конструктором напрямую. Если одни и те же действия могут выполняться как конструктором, так и каким-либо другим методом, предпочтительнее определить отдельный метод для выполнения этого набора действий.

21.10. Параметры конструктора

Конструктор, как и любой другой метод, может принимать параметры, которые передаются ему оператором `new` в круглых скобках, следующих после имени класса. В листинге 21.13 приводится класс `point`, предназначенный для моделирования точки в двумерной декартовой системе координат. Два его члена `$X` и `$Y` определяют координаты точки по оси абсцисс и ординат соответственно.

Листинг 21.13. Передача параметров конструктору

```
<?php
  class point
  {
    public function __construct($x, $y)
    {
      $this->X = $x;
```

```
      $this->Y = $y;
    }
    public fuction get_x()
    {
      return $this->X;
    }
    public fuction get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
  }
?>
```

В листинге 21.14 демонстрируется использование конструктора, инициализирующего закрытые члены класса.

Листинг 21.14. Передача параметров конструктору

```
<?php
  // Подключаем реализацию класса point
  require_once("class.point.php");

  // $obj = new point(); // Вывод предупреждения
  $obj = new point(10, 20);
  echo $obj->get_x()." ".$obj->get_y(); // 10 20
?>
```

Важно отметить, что если указать параметры при объявлении объекта, то интерпретатор PHP выведет в окно браузера предупреждение "Missing argument 1 for point::__construct()" ("Пропущен первый аргумент для конструктора класса point").

PHP не поддерживает перегрузку методов — создание нескольких разных конструкторов (или других методов) с разным количеством аргументов. Однако можно обойти это ограничение, передавая конструктору несколько аргументов, не все из которых обязательны. В листинге 21.15 класс `point` переработан таким образом, что может принимать:

- два аргумента — в этом случае значения закрытых членов `$X` и `$Y` определяются внешним программистом;
- один аргумент — при этом внешний программист определяет только значение закрытого члена `$X`, а член `$Y` получает нулевое значение;
- ни одного аргумента — оба закрытых члена `$X` и `$Y` получают нулевые значения.

Листинг 21.15. Использование параметров по умолчанию

```
<?php
  class point
  {
    public function __construct($x = 0, $y = 0)
    {
      $this->X = $x;
      $this->Y = $y;
    }
    public function get_x()
    {
      return $this->X;
    }
    public function get_y()
    {
      return $this->Y;
    }

    private $X;
    private $Y;
  }
?>
```

Если один из параметров конструктору не передается, он получает значение по умолчанию.

21.11. Деструктор. Метод *__destruct()*

Деструктор — это специальный метод класса, который автоматически выполняется в момент уничтожения объекта. Данный метод вызывается всегда самым последним и используется главным образом для корректного освобождения зарезервированных конструктором ресурсов.

Для объявления деструктора в классе необходимо создать метод с именем `__destruct()`. В листинге 21.16 приводится пример объявления класса `cls`, конструктор которого выводит сообщение "Вызов конструктора", метод `print_msg()` — сообщение "Вызов метода", а деструктор — "Вызов деструктора".

> ***ЗАМЕЧАНИЕ***
>
> Деструктор появился только в версии PHP 5.0.0. Допускается объявление закрытого деструктора, однако при этом попытка уничтожения объекта заканчивается выводом предупреждения "Warning: Call to private point::__destruct() from context " during shutdown ignored".

Листинг 21.16. Использование деструктора

```
<?php
  class cls
  {
    public function __construct()
    {
      echo "Вызов конструктора<br>";
    }
    public function print_msg()
    {
      echo "Вызов метода<br>";
    }
    public function __destruct()
    {
      echo "Вызов деструктора<br>";
    }
  }
?>
```

В листинге 21.17 приводится пример создания объекта класса `cls`.

Листинг 21.17. Создание объекта, в котором реализован деструктор

```
<?php
  // Подключаем реализацию класса cls
  require_once("class.cls.php");

  $obj = new cls();
  $obj->print_msg();
  echo "Произвольный текст<br>";
?>
```

Скрипт, представленный в листинге 21.17, выводит в окно браузера следующие строки:

```
Вызов конструктора
Вызов метода
Произвольный текст
Вызов деструктора
```

Как можно видеть, деструктор выполняется в последнюю очередь и уничтожает объект при завершении работы скрипта.

> ***ЗАМЕЧАНИЕ***
>
> Явно вызывать деструктор не следует, поскольку в этом случае он будет вызван дважды. Для однозначного вызова деструктора лучше воспользоваться конструкцией `unset()`, уничтожающей объект.

21.12. Автозагрузка классов. Функция *__autoload()*

Обычно класс оформляется в виде отдельного файла, который вставляется в нужном месте при помощи конструкции `require_once()`. Если используется большое количество классов, то в начале скрипта выстраивается целая вереница конструкций `require_once()`, что может быть не очень удобно, особенно если путь к классам приходится часто изменять. Начиная с PHP 5, разработчику предоставляется специальная функция `__autoload()`, которая позволяет задать путь к каталогу с классами и автоматически подключать классы при обращении к ним в теле программы. Данная функция принимает в качестве единственного параметра имя класса (листинг 21.18).

> ***ЗАМЕЧАНИЕ***
>
> Функция `__autoload()` не является методом класса. Это независимая функция, которую PHP-разработчик может перегружать.

Листинг 21.18. Автозагрузка классов

```
<?php
  // Функция автозагрузки классов
  function __autoload($classname)
  {
    require_once("class/class.$classname.php");
  }
  ...
?>
```

21.13. Аксессоры. Методы *__set()* и *__get()*

Неписаным правилом объектно-ориентированного программирования является использование только закрытых членов, доступ к которым осуществляется через открытые методы класса. Это позволяет скрыть внутреннюю реализацию класса, ограничить диапазон значений, которые можно присваивать члену, и сделать член доступным только для чтения.

Неудобство заключается в том, что для каждого из членов приходится создавать отдельный метод для чтения и присваивания нового значения, имена которых зачастую не совпадают с именами членов.

Выходом из ситуации является использование свойств, обращение к которым выглядит точно так же, как к открытым членам класса. Для их реализации необходимо перегрузить специальные методы `__get()` и `__set()`, которые часто называют *аксессорами*. Метод `__get()`, предназначенный для чтения свойства, принимает единственный параметр, который служит ключом. Метод `__set()` позволяет присвоить свойству новое значение и принимает два параметра, первый из которых является ключом, а второй — значением свойства.

В примере из листинга 21.19 при помощи метода `__set()` объекту присваиваются новые свойства, которые помещаются в массив `$this->arr`, а перегруженный метод `__get()` позволяет извлечь их из массива.

> ***ЗАМЕЧАНИЕ***
>
> Следует обратить внимание, что методы `__set()` и `__get()` можно объявлять как закрытыми, так и открытыми.

Листинг 21.19. Использование методов `__set()` и `__get()`

```
<?php
  class cls
  {
    private $arr = array();

    private function __get($index)
    {
      return $this->arr[$index];
    }

    private function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }
 }
?>
```

Как видно из листинга 21.19, класс `cls` перехватывает все обращения к членам объекта и создает соответствующий элемент в закрытом массиве `$arr`. В листинге 21.20 демонстрируется, как обращение к члену `$name` приводит к созданию соответствующего элемента массива.

Листинг 21.20. Обращение к несуществующему элементу `$name`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Hello, world!<br>";
  echo $obj->name;
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.20 являются следующие строки:

```
Hello world!
cls Object
(
    [arr:private] => Array
        (
            [name] => Hello, world!
        )
)
```

Любая попытка присвоить члену значение приводит к созданию нового элемента закрытого массива `$arr`. Интересно, что когда член в классе уже существует, то аксессоры `__set()` и `__get()` перехватывают обращение к нему, если он является закрытым (имеет спецификатор доступа `private`), и не перехватывают, если он является открытым (`public`).

> ***ЗАМЕЧАНИЕ***
>
> Перегрузка обоих методов `__get()` и `__set()` не обязательна, допускается перегружать только один из них.

21.14. Проверка существования члена класса. Метод *__isset()*

Убедиться в существовании члена из внешнего кода при помощи конструкции `isset()` можно только в том случае, если он открыт. Тем не менее, при использовании аксессора `__set()` полезно знать, существует член или нет и по отношению к закрытым переменным класса. Для решения этой задачи предусмотрен специальный метод класса `__isset()`, который принимает в качестве единственного параметра имя свойства и возвращает `true`, если свойство с таким именем существует, и `false` — в противном случае. В листинге 21.21 представлена реализация метода `__isset()` для класса `cls` из листинга 21.19.

> ***ЗАМЕЧАНИЕ***
>
> Метод `__isset()` доступен в PHP, начиная с версии 5.1.0.

Листинг 21.21. Перегрузка метода `__isset()`

```
<?php
  class cls
  {
    ...
    private function __isset($index)
    {
```

```
      return isset($this->$index);
    }
    ...
  }
?>
```

21.15. Уничтожение члена класса. Метод *__unset()*

Для уничтожения членов класса служит специальный метод `__unset()`. Вернемся к классу `cls`, рассмотренному в листинге 21.19. Устанавливаемые при помощи метода `__set()` значения свойств помещаются в закрытый массив `$arr`. Уничтожение отдельных элементов закрытого массива не допускается, однако перегрузка метода `__unset()` позволяет снабдить класс такой возможностью (листинг 21.22).

> ***ЗАМЕЧАНИЕ***
>
> Метод `__unset()` доступен в PHP, начиная с версии 5.1.0.

Листинг 21.22. Перегрузка метода `__unset()`

```
<?php
  class cls
  {
    private $arr = array();

    private function __get($index)
    {
      return $this->arr[$index];
    }

    private function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }

    private function __isset($index)
    {
      return isset($this->arr[$index]);
    }

    private function __unset($index)
    {
```

```
      unset($this->arr[$index]);
    }
 }
?>
```

Теперь, несмотря на то, что массив `$arr` является закрытым членом, его элементы можно уничтожать при помощи конструкции `unset()` (листинг 21.23).

Листинг 21.23. Использование конструкции `unset()`

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Новое свойство класса";

  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  unset($obj->name);

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.23 будут следующие строки:

```
cls Object
(
    [arr:private] => Array
        (
            [name] => Новое свойство класса
        )
)
cls Object
(
    [arr:private] => Array
        (
        )
)
```

Как можно видеть, использование конструкции `unset()` приводит к уничтожению динамического свойства.

21.16. Динамические методы. Метод *__call()*

Специальный метод `__call()` предназначен для создания динамических методов: если в классе не перегружен метод `__call()`, обращение к несуществующему методу не приведет к ошибке, а передаст управление методу `__call()`. В качестве первого параметра метод `__call()` принимает имя вызываемого метода, а в качестве второго — массив, элементами которого являются параметры, переданные при вызове метода.

При помощи специального метода `__call()` можно эмулировать методы с переменным количеством параметров. В листинге 21.24 представлен класс `minmax`, который предоставляет пользователю два метода: `min()` и `max()`, принимающие произвольное количество числовых параметров и определяющие минимальное и максимальное значения соответственно.

Листинг 21.24. Использование специального метода `__call()`

```
<?php
  class minmax
  {
    private function __call($method, $arg)
    {
      if (!is_array($arg)) return false;
      $value = $arr[0];
      if ($method == "min")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if ($agr[$i] < $value) $value = $agr[$i];
        }
      }
      if ($method == "max")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if ($agr[$i] > $value) $value = $agr[$i];
        }
      }
      return $value;
    }
  }
?>
```

В примере из листинга 21.24 в зависимости от вызываемого метода — `min()` или `max()` — используются два разных алгоритма для поиска результата. Кроме того,

для класса `minmax` допустим вызов метода с произвольным именем, и если оно отлично от `min` или `max`, будет возвращаться первый аргумент. Независимо от имени метода, если не передано ни одного аргумента, возвращается значение `FALSE`.

В листинге 21.25 приводится пример использования класса `minmax` для получения максимального и минимального значений последовательности.

Листинг 21.25. Использование класса `minmax`

```
<?php
  require_once("class.minmax.php");

  $obj = new minmax();
  echo $obj->min(43, 18, 5, 61, 23, 10, 56, 36); // 5
  echo "<br>";
  echo $obj->max(43, 18, 5, 61, 23); // 61
?>
```

Взаимодействие специального метода `__call()` с уже существующими в классе методами отличается от аксессоров `__get()` и `__set()`: если метод (закрытый или открытый) существует, то `__call()` не задействуется. Для демонстрации последнего правила снабдим класс `minmax` двумя методами: открытым методом `min()` и закрытым методом `max()` (листинг 21.26).

Листинг 21.26. Модифицированный вариант класса `minmax`

```
<?php
  class minmax
  {
    public function min($val, $val1, $val3)
    {
      echo "Вызов открытого метода min()";
    }
    private function max($val, $val1, $val3)
    {
      echo "Вызов открытого метода max()";
    }
    private function __call($method, $arg)
    {
      if (!is_array($arg)) return false;
      $value = $arg[0];
      if ($method == "min")
      {
        for($i = 0; $i < count($arg); $i++)
```

```
        {
          if ($arg[$i] < $value) $value = $arg[$i];
        }
      }
      if ($method == "max")
      {
        for($i = 0; $i < count($arg); $i++)
        {
          if ($arg[$i] > $value) $value = $arg[$i];
        }
      }
      return $value;
    }
  }
?>
```

При выполнении кода из листинга 21.26 при обращении к методу `min()` будет выведена фраза "Вызов открытого метода min()", а обращение к методу `max()` заканчивается ошибкой обращения к закрытому методу: "Fatal error: Call to private method minmax::max()".

21.17. Интерполяция объекта. Метод *__toString()*

Специальный метод `__toString()` позволяет интерполировать (подставлять) объект в строку. Для подстановки значений переменных необходимо заключить строку в двойные кавычки (листинг 21.27).

> ***ЗАМЕЧАНИЕ***
>
> Следует отметить, что значение переменной не интерполируется, если вместо двойных кавычек используются одинарные.

Листинг 21.27. Интерполяция переменной

```
<?php
  $str = "12345";
  echo "str = $str<br>"; // str = 12345
  echo 'str = $str<br>'; // str = $str
?>
```

Такого же поведения можно добиться и от объекта, если реализовать в его классе метод `__toString()`, который преобразует объект в строку.

Модифицируем класс `employee` (листинг 21.8) таким образом, чтобы его подстановка в строку приводила к выводу фамилии сотрудника и его инициалов (листинг 21.28).

Замечание

Следует обратить внимание, что метод `__toString()` выводит результат при помощи конструкции `return`, а не `echo`.

Листинг 21.28. Использование специального метода `__toString()`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
    private function __toString()
    {
      return "{$this->surname} {$this->name[0]} {$this->patronymic[0]}";
    }
    private function __get($index)
    {
      return $this->$index;
    }

    public $surname;
    public $name;
    private $patronymic;
  }
?>
```

В листинге 21.29 приводится пример интерполяции объекта класса `employee`, при этом вместо объекта `$obj` будет подставлена фраза `Борисов Игорь Иванович`.

Листинг 21.29. Интерполяция объекта

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");

  echo "Сотрудник $obj недавно принят на работу";
?>
```

Следует отметить, что вызов объекта в строковом контексте возможен, только если его класс содержит реализацию метода `__toString()`, в противном случае попытка использовать объект в строке будет заканчиваться ошибкой "Catchable fatal error: Object of class employee could not be converted to string".

21.18. Экспорт объектов. Метод *__set_state()*

Среди многочисленных функций PHP существует функция `var_export()`, выводящая дампы переменных, массивов и объектов. В отличие от `print_r()` или `var_dump()` она возвращает дамп в виде PHP-кода, что позволяет использовать результат ее работы в функции `eval()`, выполняющей PHP-код в строке. Функция `var_export()` имеет следующий синтаксис:

`var_export($`*expression* `[, $`*return*`])`

В качестве первого аргумента `$`*expression* функция принимает переменную, массив или объект. По умолчанию она выводит дамп объекта, не возвращая никакого результата. Однако если в качестве второго необязательного параметра функции `$`*return* передается значение `TRUE`, вместо вывода дампа в окно браузера она возвращает его в виде строки.

В листинге 21.30 приводится пример использования функции `var_export()` для вывода дампов переменной, массива и объекта. Действие функции в этом качестве аналогично функции `print_r()`.

Листинг 21.30. Использование функции `var_export()` для вывода дампов

```
<?php
  // Переменная
  $var = 100;

  // Массив
  $arr = array(1, 2, 3, array(4, 5), array(6, 7));

  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
      $this->priv_var = $val;
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
```

```
  $obj = new cls(12, 147);

  echo "<pre>";
  var_export($var);
  var_export($arr);
  var_export($obj);
  echo "<pre>";
?>
```

Результатом работы скрипта из листинга 21.30 будут следующие дампы переменной `$var`, массива `$arr` и объекта `$obj`:

```
100
array (
  0 => 1,
  1 => 2,
  2 => 3,
  3 => array (
    0 => 4,
    1 => 5,
  ),
  4 => array (
    0 => 6,
    1 => 7,
  ),
)
cls::__set_state(
array(
   'publ_var' => 12,
   'priv_var' => 147,
))
```

В листинге 21.31 приводится пример создания копии массива `$arr` при помощи функции `var_export()`.

> ***ЗАМЕЧАНИЕ***
>
> К сожалению, из-за ошибки реализации функции `var_export()` приходится самостоятельно заботиться об удалении последней запятой из дампов массивов и объектов. Вероятно, эта досадная ошибка будет устранена в более поздних версиях PHP.

Листинг 21.31. Создание копии массива `$arr`

```
<?php
  $arr = array(1, 2, 3, array(4, 5), array(6, 7));
  $str = var_export($arr, true);
```

```
  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);
  // Создаем массив $copy
  eval('$copy = '.$str.';');

  echo "<pre>";
  print_r($copy);
  echo "</pre>";
?>
```

Как видно из листинга 21.31, скрипт создает копию массива `$arr` и помещает ее в массив `$copy`. Массив воссоздается путем динамического выполнения PHP-кода при помощи функции `eval()`. Результатом работы скрипта будут следующие строки:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => Array
        (
            [0] => 4
            [1] => 5
        )
    [4] => Array
        (
            [0] => 6
            [1] => 7
        )
)
```

По отношению к объекту функция `var_export()` действует несколько иным образом: поскольку невозможно предугадать заранее код, который бы воспроизводил объект, функция возвращает код вызова специального метода `__set_state()`, разработкой которого должен озаботиться автор класса. В качестве единственного параметра методу `__set_state()` передается ассоциативный массив со всеми членами объекта. В листинге 21.32 приводится пример использования метода `__set_state()`: получив значения объекта в массиве `$arr_obj`, метод `__set_state()` выводит все значения в окно браузера.

> ***Замечание***
>
> Специальный метод `__set_state()` введен в PHP, начиная с версии 5.1.0.

Листинг 21.32. Использование специального метода `__set_state()`

```
<?php
  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
      $this->priv_var = $val;
    }
    public function __set_state($arr_obj)
    {
      foreach($arr_obj as $key => $value)
      {
        echo "$key => $value<br>";
      }
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
  $obj = new cls(12, 147);

  // Возвращаем вызов метода __set_state()
  $str = var_export($obj, true);

  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);

  // Вызываем метод __set_state()
  eval($str.';');
?>
```

Результатом работы скрипта из листинга 21.32 будет список закрытых и открытых членов класса:

```
publ_var => 12
priv_var => 147
```

Однако функция `var_export()` задумывалась для возвращения дампа массива или объекта, который совместно с функцией `eval()` позволяет воспроизвести объект. Поэтому рекомендуется использовать метод `__set_state()` именно для воссоздания объекта, а не для каких-то иных целей. В листинге 21.33 приводится альтернативная реализация класса `cls`, в которой метод `__set_state()` собирает и возвращает новый объект.

Листинг 21.33. Воссоздание объекта при помощи метода `__set_state()`

```
<?php
  // Класс
  class cls
  {
    public function __construct($var, $val)
    {
      $this->publ_var = $var;
      $this->priv_var = $val;
    }
    public function __set_state($arr_obj)
    {
      return new cls($arr_obj['publ_var'], $arr_obj['priv_var']);
    }
    public  $publ_var;
    private $priv_var;
  }
  // Объект
  $obj = new cls(12, 147);

  // Возвращаем вызов метода __set_state()
  $str = var_export($obj, true);

  // Из-за ошибки реализации приходится
  // удалять последнюю запятую самостоятельно
  $str = preg_replace("|,[\s]*\)|is", ")", $str);

  // Создаем объект $new_obj - копию объекта $obj
  eval('$new_obj = '.$str.';');

  // Выводим дамп нового объекта $new_obj
  echo "<pre>";
  print_r($new_obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.33 будет следующий дамп нового объекта `$new_obj`:

```
cls Object
(
    [publ_var] => 12
    [priv_var:private] => 147
)
```

21.19. Статические члены класса

Воспользоваться членами и методами можно и без объявления объекта класса, объявив их статическими с помощью ключевого слова `static`, что делает их доступными в любой момент. Обращение к компоненту класса в этом случае производится при помощи оператора разрешения области видимости `::`. Одной из характерных особенностей статических членов классов является возможность их инициализации непосредственно при объявлении (листинг 21.34).

Листинг 21.34. Объявление статического члена класса

```
<?php
  class cls
  {
     public static $staticvar = 100;
  }

  echo cls::$staticvar; // 100
?>
```

К статическим членам класса нельзя обращаться через префикс `$this->`, и они не отображаются в списках членов объектов при выводе дампа. В листинге 21.35 приводится модифицированный вариант класса `cls`, в котором метод `set_staticvar()` осуществляет попытку изменить значение статического члена класса `$staticvar` при помощи префикса `$this->`.

Листинг 21.35. Попытка использования префикса `$this->` для доступа к статическому члену класса

```
<?php
  class cls
  {
     public static $staticvar = 100;
     public function set_staticvar($val)
     {
       $this->staticvar = $val;
     }
  }

  $obj = new cls();

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
```

```
  $obj->set_staticvar(20);

  echo cls::$staticvar; // 100

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.35 будут следующие строки:

```
cls Object
(
)
100
cls Object
(
    [staticvar] => 20
)
```

Таким образом, попытка обратиться к статическому члену класса через префикс `$this->` приводит к созданию нового члена класса, изменение значения которого не затрагивает значение статического члена.

Такое поведение связано с тем, что в отличие от других членов класса статические члены являются общими для всех объектов данного класса (рис. 21.3).

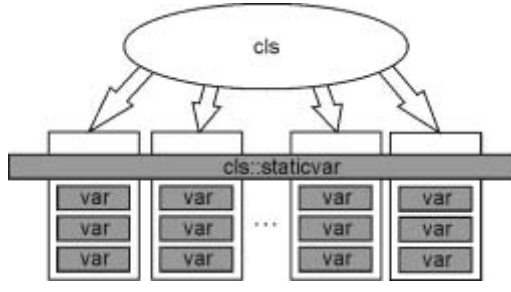


Рис. 21.3. Статический член класса является общим для всех объектов

Это означает, что изменение содержимого статической переменной в одном объекте отражается на значении данной переменной во всех объектах. Данную особенность статических переменных очень часто используют для создания счетчиков объектов или ресурсов, которые они резервируют. В листинге 21.36 представлен класс `counter`, в составе которого присутствует статическая переменная `$count`. Изначально она имеет нулевое значение, однако в конструкторе класса оно увеличивается на единицу, а в деструкторе — уменьшается. Таким образом, в каждый момент времени можно определить количество созданных объектов.

Листинг 21.36. Счетчик объектов

```
<?php
  class counter
  {
    public static $count = 0;
    public function __construct()
    {
      counter::$count++;
    }
    public function __destruct()
    {
      counter::$count--;
    }
  }
?>
```

Результат использования класса `counter` демонстрируется в листинге 21.37.

Листинг 21.37. Счетчик объектов

```
<?php
  require_once("class.counter.php");

  $obj = new counter();
  echo counter::$count."<br>"; // 1

  for($i = 0; $i < 3; $i++)
  {
    $arr[] = new counter();
    echo counter::$count."<br>"; // 2, 3, 4
  }

  // Уничтожаем массив объектов
  unset($arr);

  echo counter::$count."<br>"; // 1
?>
```

21.20. Статические методы класса

Статическими можно объявлять не только члены, но и методы класса. Для объявления статического метода также используется ключевое слово `static`, а для обращения к методу — оператор разрешения области видимости `::`. В листинге 21.38 приводится пример класса `cls`, содержащего единственный статический метод `static_method()`.

Листинг 21.38. Объявление статического метода

```
<?php
  class cls
  {
    public static function static_method()
    {
      echo "Вызов статического метода";
    }
  }

  cls::static_method(); // Вызов статического метода
?>
```

Впрочем, для вызова методов при помощи оператора разрешения области видимости `::` без создания объекта не обязательно объявлять метод статическим. Вообще-то вызов нестатического метода через оператор `::` должен приводить к генерации предупреждения, однако в текущей версии PHP этого не происходит. Возможно, это будет исправлено в следующих версиях PHP.

21.21. Константы класса

Наряду с членами классы могут содержать константы, которые определяются при помощи ключевого слова `const`. В листинге 21.39 приводится пример класса `cls`, включающего в свой состав константу `NAME`, которая содержит имя класса.

ЗАМЕЧАНИЕ

Следует обратить внимание, что имя константы не содержит символа `$`.

Листинг 21.39. Использование констант в классах

```
<?php
  class cls
  {
    const NAME = "cls";
```

```
    public function method()
    {
      // echo $this->NAME; // Ошибочное обращение
      echo self::NAME;
      echo "<br>";
      echo cls::NAME;
      echo "<br>";
    }
  }

  echo cls::NAME;
?>
```

ЗАМЕЧАНИЕ

Имена констант могут назначаться в любом регистре, однако традиционно используется верхний. Лучше придерживаться данной традиции, т. к. это позволяет значительно увеличить читабельность кода (большинство программистов будет ожидать, что имена констант в программе представлены в верхнем регистре).

Точно так же как и в случае со статическими членами классов, к константам нельзя обращаться при помощи оператора `->`; для обращения используется оператор разрешения области видимости `::`, который предваряется либо именем класса, либо ключевыми словами `self` и `parent`.

Существование констант может быть проверено при помощи функции `defined()`, которая возвращает `true`, если константа существует, и `false` — в противном случае (листинг 21.40).

ЗАМЕЧАНИЕ

При проверке классовых констант следует в обязательном порядке использовать оператор разрешения области видимости `::` и имя класса.

Листинг 21.40. Проверка существования классовых констант

```
<?php
  require_once("class.cls.php");

  if (defined("cls::NAME")) echo "Константа определена<br>"; // true
  else echo "Константа не определена<br>";

  if (defined("cls::POSITION")) echo "Константа определена<br>"; // false
  else echo "Константа не определена<br>";
?>
```

21.22. Предопределенные константы

Помимо констант, которые разработчик может вводить в класс, существуют предопределенные константы, которые определяет PHP-интерпретатор (табл. 21.3).

Таблица 21.3. *Предопределенные константы PHP*

Константа	Описание
__LINE__	Номер текущей строки в файле с PHP-скриптом
__FILE__	Полный путь к файлу с PHP-скриптом
__FUNCTION__	Имя функции, из которой вызывается константа
__CLASS__	Имя текущего класса
__METHOD__	Имя текущего метода класса

ЗАМЕЧАНИЕ

Константа __METHOD__ введена в PHP, начиная с версии 5.0.0.

В листинге 21.41 приводится пример использования предопределенных констант для идентификации текущего номера строки, файла и имени метода. Основное назначение предопределенных констант заключается в точной идентификации места возникновения ошибки.

Листинг 21.41. Использование предопределенных констант

```
<?php
  function get_point()
  {
     return "Вызов функции ".__FUNCTION__.
            "<br>файла ".__FILE__.
            "<br>в строке ".__LINE__; // 6 строка
  }

  echo get_point(); // 9 строка
?>
```

Результатом работы скрипта из листинга 21.41 будут следующие строки:

```
Вызов функции get_point
файла D:\main\oop\06\index.php
в строке 6
```

Таким образом, константа __LINE__ подставляет номер строки вызова константы, а не фактического вызова функции, из которой она вызывается. Точно так же действуют и все остальные константы — включение файла из листинга 21.41 в другие файлы при помощи инструкций include и require не приводит к смене их значений.

Как видно из листинга 21.41, предопределенные константы __LINE__, __FILE__ и __FUNCTION__ не связаны с объектно-ориентированным подходом непосредственно, однако ничто не мешает использовать их в классах (листинг 21.42).

Листинг 21.42. Использование предопределенных констант в классе

```
<?php
  class cls
  {
    public function get_point()
    {
       return "Вызов функции ".__FUNCTION__.
              "<br>файла ".__FILE__.
              "<br>в строке ".__LINE__; // 8 строка
    }
  }

  echo cls::get_point(); // 12 строка
?>
```

Результатом работы скрипта из листинга 21.42 будут следующие строки:

```
Вызов функции get_point
файла D:\main\oop\06\class.cls.php
в строке 8
```

Для использования в классах предусмотрены также специализированные константы: __CLASS__ и __METHOD__ (листинг 21.43).

Листинг 21.43. Использование констант __CLASS__ и __METHOD__

```
<?php
  class cls
  {
    public function get_point()
    {
       return "Вызов метода ".__METHOD__.
              "<br>класса ".__CLASS__.
              "<br>в файле ".__FILE__.
              "<br>в строке ".__LINE__; // 9 строка
    }
  }

  echo cls::get_point(); // 13 строка
?>
```

Результатом работы скрипта из листинга 21.43 будут следующие строки:

```
Вызов метода cls::get_point
класса cls
в файле D:\main\oop\06\class.cls.php
в строке 9
```

21.23. Клонирование объекта

Оператор присвоения = не приводит к созданию новой копии объекта: и старый, и новый объект указывают на одну и ту же область памяти. В листинге 21.44 представлена операция присвоения одного объекта класса `cls` другому, при этом изменение члена класса нового объекта `$new_obj` отражается на старом объекте `$obj`.

Листинг 21.44. Присвоение одного объекта другому

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }

  $obj = new cls();
  $new_obj = $obj;
  $new_obj->var = 200;
  echo $obj->var; // 200
?>
```

Схематично процесс присвоения объекта `$obj` объекту `$new_obj` продемонстрирован на рис. 21.4.

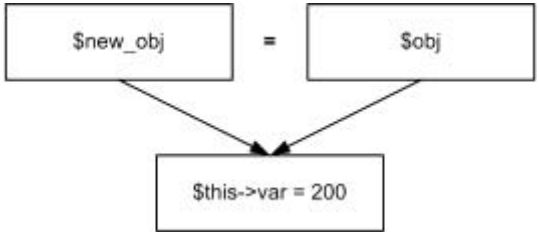


Рис. 21.4. Присвоение одного объекта другому не приводит к созданию новой копии; текущий объект получает дополнительный псевдоним

Для создания копии текущего объекта используется специальная операция — *клонирование*. Оно выполняется при помощи ключевого слова `clone`, которое располагается непосредственно перед объектом клонирования (листинг 21.45).

Листинг 21.45. Клонирование объекта

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
    }
  }

  $obj = new cls();
  $new_obj = clone $obj;
  $new_obj->var = 200;
  echo $obj->var; // 100
?>
```

Схематически процесс клонирования объекта `$obj` и получения независимой копии `$new_obj` представлен на рис. 21.5.

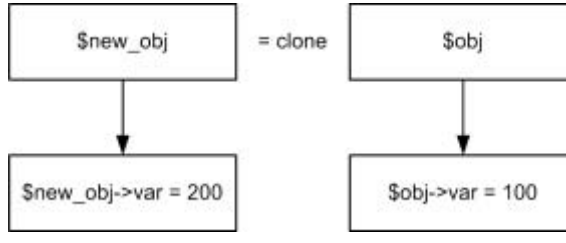


Рис. 21.5. Получение независимой копии объекта `$obj` посредством клонирования

21.24. Управление процессом клонирования. Метод *__clone()*

Клонирование объекта приводит к созданию новой копии объекта без вызова его конструктора, в чем можно убедиться, поместив в конструктор вывод отладочной записи (листинг 21.46).

Листинг 21.46. Конструктор не вызывается при клонировании

```
<?php
  class cls
  {
    public $var;
    public function __construct()
    {
      $this->var = 100;
      echo "Вызов конструктора<br>";
    }
  }

  $obj = new cls();
  $new_obj = clone $obj;
?>
```

В результате работы скрипта из листинга 21.46 в окно браузера будет выведена лишь одна запись — **Вызов конструктора**. Тем не менее, в процессе создания нового объекта может потребоваться выполнить ряд действий. Для этого PHP предоставляет специальный метод `__clone()`, который можно переопределить в классе. В листинге 21.47 приводится пример улучшенного класса `counter`, который подсчитывает количество созданных объектов и подробно рассматривался в *разд. 21.19*. При создании нового объекта статическая переменная `$count` увеличивается на единицу в конструкторе, при уничтожении объекта — на единицу уменьшается. Однако при клонировании, как было продемонстрировано ранее, конструктор не вызывается, и в системе появляются неучтенные объекты. Подобную ситуацию можно исправить перегрузкой метода `__clone()`.

> ***ЗАМЕЧАНИЕ***
>
> Метод `__clone()` не вмешивается в работу клонирования, т. е. программист не должен реализовывать механизм клонирования самостоятельно; он может лишь использовать метод `__clone()` для реакции на выполнение клонирования.

Листинг 21.47. Использование метода `__clone()`

```
<?php
  class counter
  {
    public static $count = 0;
    public function __construct()
    {
      counter::$count++;
```

```
    }
    public function __destruct()
    {
      counter::$count--;
    }
    public function __clone()
    {
      counter::$count++;
    }
  }
?>
```

Важной деталью является то, что метод `__clone()` выполняется для нового объекта. Таким образом, все изменения, производимые над объектом методом `__clone()`, будут отражаться на новом, а не на старом объекте.

21.25. Сериализация объектов

В силу особенности применения PHP время жизни объектов, как правило, очень невелико: все переменные и объекты уничтожаются после завершения скрипта. Протокол HTTP не является сессионным, т. е. каждое обращение к серверу воспринимается как обращение нового клиента, а история предыдущих обращений не сохраняется. Разработчик Web-приложений должен сам реализовывать сохранение состояния приложения для каждого из клиентов, прибегая к сессиям и cookies. В таких условиях большое значение приобретает возможность передачи объекта между несколькими сеансами клиента или даже между отдельными страницами Web-приложения.

Для сохранения объекта в формате, который позволял бы его восстанавливать впоследствии, часто прибегают к сериализации — переводу объекта в строку при помощи функции `serialize()`. Такая строка может быть сохранена в файл или базу данных, после чего сохраненный объект можно получить при помощи обратной функции `unserialize()`. Данные функции имеют следующий синтаксис:

```
serialize($obj)
unserialize($str)
```

Для демонстрации приемов работы с функциями `serialize()` и `unserialize()` создадим вспомогательный класс `cls`, содержащий единственный открытый член `$var`, инициализация которого осуществляется в конструкторе класса (листинг 21.48).

Листинг 21.48. Класс `cls`

```
<?php
  class cls
  {
    public $var;
```

```
    public function __construct($var)
    {
      $this->var = $var;
    }
  }
?>
```

В листинге 21.49 представлен скрипт, который сериализует объект `$obj` класса `cls` в строку, а строку сохраняет в файл text.obj.

> ***ЗАМЕЧАНИЕ***
>
> Сериализации могут подвергаться не только объекты, но и массивы (в том числе многомерные).

Листинг 21.49. Сериализация объекта `$obj` класса `cls`

```
<?php
  // Подключаем определение класса cls
  require_once("class.cls.php");

  // Создаем объект
  $obj = new cls(100);

  // Стерилизуем объект
  $text = serialize($obj);

  // Сохраняем объект в файл
  $fd = fopen("text.obj", "w");
  if (!$fd) exit("Невозможно открыть файл");
  fwrite($fd, $text);
  fclose($fd);
?>
```

Результатом работы скрипта из листинга 21.49 будет файл text.obj, содержащий следующую строку:

```
O:3:"cls":1:{s:3:"var";i:100;}
```

Данная строка предназначена для функции `unserialize()` и позволяет восстановить объект в другом файле (листинг 21.50).

Листинг 21.50. Восстановление объекта из строки

```
<?php
  // Подключаем определение класса cls
  require_once("class.cls.php");
```

```
  // Извлекаем сериализованный объект из файла
  $fd = fopen("text.obj", "r");
  if (!$fd) exit("Невозможно открыть файл");
  $text = fread($fd, filesize("text.obj"));
  fclose($fd);

  // Восстанавливаем объект
  $obj = unserialize($text);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.50 будет следующий дамп объекта `$obj`:

```
cls Object
(
    [var] => 100
)
```

Важно, чтобы в момент восстановления объекта, скрипт имел доступ к классу `cls`, иначе восстановление объекта будет проведено лишь частично. По сути, будет создан объект-контейнер, в котором будут присутствовать члены класса `cls`, однако отсутствовать какие бы то ни было методы:

```
__PHP_Incomplete_Class Object
(
    [__PHP_Incomplete_Class_Name] => cls
    [var] => 100
)
```

21.26. Управление сериализацией. Методы *__sleep()* и *__wakeup()*

При сохранении и восстановлении объекта при помощи функций `serialize()` и `unserialize()` может потребоваться осуществить ряд действий, например, убрать из объекта данные, которые не должны подвергаться сериализации, или скорректировать их значения, если они теряют актуальность. Для осуществления подобных действий предназначены два специальных метода, которые могут быть перегружены в классе: метод `__sleep()`, который вызывается, когда объект подвергается сериализации при помощи функции `serialize()`; и метод `__wakeup()`, который вызывается при восстановлении объекта при помощи функции `unserialize()`. Оба метода не принимают никаких параметров.

ЗАМЕЧАНИЕ

Название метода `__sleep()` образовано от английского глагола "спать", а метода `__wakeup()` — от глагола "пробуждаться".

Для демонстрации приемов работы со специальными методами `__sleep()` и `__wakeup()` создадим класс `user`, который будет иметь в своем составе следующие члены:

- `$name` — имя пользователя;
- `$password` — его пароль (если поле пустое, то пользователь перенаправляется на страницу авторизации);
- `$referrer` — последняя посещенная страница;
- `$time` — время авторизации пользователя.

В листинге 21.51 приводится возможная реализация класса `user`.

Листинг 21.51. Класс `user`

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

В листинге 21.52 демонстрируется скрипт, который подвергает сериализации объект `$obj` класса `user`.

Листинг 21.52. Сериализация объекта класса `user`

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Создаем объект
  $obj = new user("nick", "password");

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Стерилизуем объект
  $object = serialize($obj);

  // Выводим сериализованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 21.52 будут следующие строки:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676349
)
O:4:"user":4:{s:4:"name";s:4:"nick";s:8:"password";s:8:"password";s:8:"referre
r";s:17:"/oop/07/index.php";s:4:"time";i:1177676349;}
```

При сериализации объекта полезно назначать паролю `$password` пустую строку, чтобы не допускать его сохранения на жестком диске в незашифрованном виде. Для решения этой задачи как нельзя лучше подходит метод `__sleep()`, который обнулит член `$password` и вернет полученный объект функции `serialize()`. По сути метод `__sleep()` выступает в качестве фильтра, позволяя настроить процесс сериализации на избирательное сохранение информации (рис. 21.6).

В листинге 21.53 представлен модифицированный вариант класса `user`, который обнуляет значение члена `$password` при сериализации объекта.

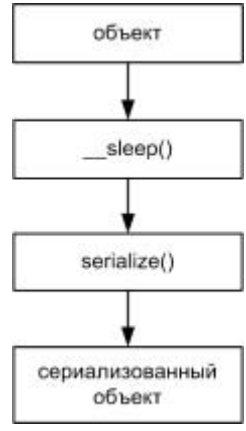


Рис. 21.6. Схема сериализации объекта

Листинг 21.53. Использование метода `__sleep()`

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $this->password = "";
      return $this;
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

Однако теперь попытка использовать функцию `serialize()` применительно к объекту класса `user` будет приводить к обнулению члена `$password`, что может быть неудобно, если планируется дальнейшее использование объекта (листинг 21.54).

Листинг 21.54. Побочный эффект сериализации

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Создаем объект
  $obj = new user("nick", "password");

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Сериализует объект
  $object = serialize($obj);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Выводим сериализованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 21.54 будут следующие строки, из которых видно, что вызов функции `serialize()` приводит к необратимой модификации объекта `$obj`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676475
)
user Object
(
    [name] => nick
```

```
    [password] =>
    [referrer] => /oop/07/index.php
    [time] => 1177676475
)
O:4:"user":4:{s:4:"nick";N;s:0:"";N;s:17:"/oop/07/index.php";N;N;}
```

Для решения этой проблемы удобно воспользоваться клонированием, редактируя и возвращая не оригинальный объект, а его копию (листинг 21.55).

Листинг 21.55. Исключение побочного эффекта сериализации

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $obj = clone $this;
      $obj->password = "";
      return $obj;
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

Теперь выполнение листинга 21.55 будет демонстрировать неизменность объекта, как после вызова функции `serialize()`:

```
user Object
(
    [name] => nick
```

```
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676630
)
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177676630
)
O:4:"user":4:{s:4:"nick";N;s:0:"";N;s:17:"/oop/07/index.php";N;N;}
```

Восстановление объекта из сериализованного состояния при помощи функции `unserialize()` приведет к созданию объекта, в котором сохраняется время авторизации пользователя `$time`. Разумно при этом обновить член `$time` при вызове функции `unserialize()`. Для решения этой задачи предназначен специальный метод `__wakeup()`, который вызывается сразу после восстановления объекта (рис. 21.7).

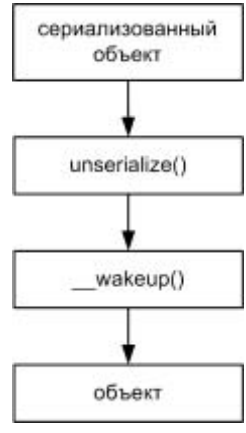


Рис. 21.7. Схема восстановления объекта

В листинге 21.56 представлен модифицированный класс `user`, который обеспечивает обновление времени авторизации пользователя при восстановлении объекта.

Листинг 21.56. Использование метода `__wakeup()`

```
<?php
  class user
  {
    // Конструктор
    public function __construct($name, $password)
```

```
    {
      $this->name     = $name;
      $this->password = $password;
      $this->referrer = $_SERVER['PHP_SELF'];
      $this->time     = time();
    }
    public function __sleep()
    {
      $obj = clone $this;
      $obj->password = "";
      return $obj;
    }
    public function __wakeup()
    {
      $this->time = time();
    }

    // Имя пользователя
    public $name;
    // Его пароль
    public $password;
    // Последняя посещенная страница
    public $referrer;
    // Время авторизации пользователя
    public $time;
  }
?>
```

В листинге 21.57 представлен скрипт, который демонстрирует восстановление объекта из сериализованного состояния с обновлением члена `$time`.

Листинг 21.57. Восстановление объекта

```
<?php
  // Подключаем реализацию класса
  require_once("class.user.php");

  // Сериализованный объект
  $object = 'O:4:"user":4:{s:4:"name";s:4:"nick";'.
            's:8:"password";s:8:"password";'.
            's:8:"referrer";s:17:"/oop/07/index.php";'.
            's:4:"time";i:1177676349;}';
```

```
  // Восстанавливаем объект
  $obj = unserialize($object);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 21.57 будет следующий дамп объекта класса `user`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177677626
)
```

В листинге 21.57 пришлось использовать сериализованный вариант, который не подвергался фильтрации через метод `__sleep()`. Это связано с тем, что в текущих версиях PHP метод `__sleep()` возвращает массив, а не объект, и при попытке возврата объекта автоматически приводит его к массиву, отбрасывая имена полей. В результате невозможно восстановить после сериализации объект, класс которого содержит перегруженный метод `__sleep()` (при отсутствии этого метода восстановление происходит штатно). Возможно, в будущих версиях PHP эта ситуация будет исправлена, пока же для использования методов `__sleep()` и `__wakeup()` следует искать альтернативные варианты.



Глава 22

Наследование

Одной из главных целей объектно-ориентированного подхода является повторное использование кода. Иногда сложно определить, требуется ли такой подход в решении поставленной задачи или нет. Однако если созданный класс используется лишь однажды в одном приложении, можно утверждать, что его создание — пустая трата времени. Объектно-ориентированное программирование значительно снижает читабельность и эффективность, поэтому окупает себя только при интенсивном использовании наследования.

Хотя при разработке новых приложений функции или участки кода могут заимствоваться из уже разработанных программ, говоря о повторном использовании кода, редко имеют в виду такой подход. Извлечение участка кода из старого проекта часто требует его серьезной переработки или подключения дополнительных файлов, которые дублируют уже существующие функциональные блоки.

> ***Замечание***
>
> В идеале класс, созданный в одном приложении, должен свободно извлекаться из него и встраиваться в другое приложение без всякой адаптации.

Код, предназначенный для повторного использования, необходимо подготовить для этого, оформив в виде библиотеки или класса. При этом повторное использование вовсе не гарантирует, что система (особенно первая) будет занимать меньше места и разрабатываться быстрее. За любое ограничение, которое накладывается на код, приходится расплачиваться. В случае, рассмотренном в предыдущей главе, платой служит более сложная организации кода и более длительные сроки его подготовки.

22.1. Наследование

Объектно-ориентированная методология предоставляет два возможных способа повторного использования кода:

- включение объектов в класс;
- использование наследования.

Наследование позволяет создать новый класс на основе уже существующего, автоматически включив в новый класс все члены и методы старого. В рамках наследования "старый" класс называется *базовым*, а вновь создаваемый класс — *производным*.

При объявлении производного класса необходимо указать имя базового класса с помощью ключевого слова `extends`. В листинге 22.1 создается базовый класс `base`, содержащий единственный член `$var` и метод `print_var()`. От класса `base` наследуется класс `derived`, содержащий, в свою очередь, член `$val` и метод `print_val()`.

Листинг 22.1. Наследование класса `derived` от класса `base`

```
<?php
  // Базовый класс
  class base
  {
    public $var;
    public function print_var()
    {
      echo $this->var;
    }
  }
  // Производный класс
  class derived extends base
  {
    public $val;
    public function print_val()
    {
      echo $this->val;
    }
  }
?>
```

В листинге 22.2 приводится пример, в котором объявляется объект базового класса `$bobj` и объект производного класса `$dobj`. Для каждого из объектов выводится список его членов и методов.

Листинг 22.2. Анализ объектов базового и производного классов

```
<?php
  // Подключаем базовый и производный классы
  require_once("class.base.php");

  // Объявляем объект базового класса
  $bobj = new base();

  // Члены и методы базового класса
```

```
  echo "<pre>";
  print_r($bobj);
  print_r(get_class_methods($bobj));
  echo "</pre>";

  // Объявляем объект производного класса
  $dobj = new derived();

  // Члены и методы производного класса
  echo "<pre>";
  print_r($dobj);
  print_r(get_class_methods($dobj));
  echo "</pre>";
?>
```

Выполнив скрипт из листинга 22.2, можно получить следующие дампы объектов `$bobj` и `$dobj`:

```
base Object
(
    [var] =>
)
Array
(
    [0] => print_var
)
derived Object
(
    [val] =>
    [var] =>
)
Array
(
    [0] => print_val
    [1] => print_var
)
```

Как видно из результатов работы скрипта, объект производного класса `derived` содержит не только собственный член `$val` и метод `print_val()`, но и член `$var` и метод `print_var()`, объявленные в базовом классе. Производный класс, в свою очередь, может выступать в качестве базового для других классов. В результате можно получить разветвленную иерархию классов, расширяя функциональность без повторного создания членов и методов, а используя уже имеющиеся из существующих классов.

22.2. Спецификаторы доступа и наследование

Члены и методы, объявленные со спецификаторами доступа `public` и `private`, при наследовании ведут себя по отношению к производному классу точно так же, как и по отношению к внешней программе. Это означает, что из производного класса доступны все методы и члены, объявленные со спецификатором доступа `public`, и не доступны компоненты, объявленные как `private`. В листинге 22.3 приводится пример попытки обратиться к закрытому члену `$var` базового класса `base` из конструктора производного класса `derived`.

Листинг 22.3. Попытка обращения к закрытому члену базового класса

```
<?php
  class base
  {
    private $var;
    public function __construct($var)
    {
      $this->var = $var;
    }
    public function print_var()
    {
      echo $this->var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      $this->var = $var;
    }
  }

  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
  echo $obj->print_var();
?>
```

В данном случае попытка обращения к закрытому члену класса не приводит к ошибке. Дело в том, что производный класс даже не видит попытки обращения к за-

крытому члену, он просто создает новый открытый член в объекте производного класса:

```
derived Object
(
    [var:private] =>
    [var] => 20
)
```

Именно поэтому обращение к методу `print_var()` не приводит к выводу переменной `$var` — данная переменная остается не инициализированной в рамках класса `base`.

Иногда удобно, чтобы член или метод базового класса, оставаясь закрытым для внешнего кода, был открыт для производного класса. В этом случае прибегают к специальному спецификатору доступа `protected`. Компоненты класса, снабженные спецификатором доступа `protected`, называют *защищенными*.

В листинге 22.4 на примерах базового класса `base` и производного класса `derived` демонстрируется использование защищенного члена `$var`.

Листинг 22.4. Использование спецификатора доступа `protected`

```
<?php
  class base
  {
    protected $var;
    public function __construct($var)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      $this->var = $var;
    }
  }

  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
  // echo $obj->var; // Ошибка
?>
```

Результатом работы скрипта из листинга 22.4 будут следующие строки:

```
derived Object
(
    [var:protected] => 20
)
```

Как видно из результатов работы скрипта, конструктор производного класса имеет возможность инициализировать член `$var` в обход конструктора базового класса, в то же время член `$var` остается закрытым по отношению к внешнему коду.

Использование спецификатора `protected` возможно по отношению как к членам класса, так и к методам. Например, если конструктор базового класса объявлен со спецификатором `protected`, то получить его объект невозможно; доступны будут только объекты производных классов (листинг 22.5).

Листинг 22.5. Запрет создания объектов базового класса

```
<?php
  class base
  {
    private $var;
    protected function __construct($var)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      parent::__construct($var);
    }
  }

  // $obj = new base(20); // Ошибка
  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Этим приемом часто пользуются, когда объект базового класса абстрактен и скрывает в себе функциональность, необходимую для производных классов, а сам по себе не имеет смысла.

22.3. Перегрузка методов

В производном классе можно создать метод с таким же названием, что и в базовом классе, который заменит метод базового класса при вызове. Такая процедура называется *перегрузкой методов*.

В листинге 22.6 приводится пример перегрузки метода `print_var()` в производном классе `derived`, который наследуется от базового класса `base`.

Листинг 22.6. Перегрузка метода `print_var()`

```
<?php
  class base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Результатом работы скрипта из листинга 22.6 будет следующая строка:

```
Вызов метода print_var() производного класса
```

Таким образом, метод `base::print_var()` не вызывается при обращении к объекту производного класса `derived`.

Однако в рамках производного класса остается возможность вызвать метод базового класса, обратившись к нему при помощи префикса `parent::`. В листинге 22.7 представлена перегрузка метода `print_var()`, при которой метод производного класса сначала вызывает метод базового класса.

Листинг 22.7. Обращение к методу базового класса

```
<?php
  class base
  {
```

```
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      parent::print_var();
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Результатом работы скрипта из листинга 22.7 будут уже две строки:

```
Вызов метода print_var() базового класса
Вызов метода print_var() производного класса
```

Таким образом, при помощи перегрузки метода можно как расширить метод базового класса, так и полностью заменить его уже новым методом.

22.4. Полиморфизм

При использовании разветвленных наследственных иерархий все объекты производных классов автоматически снабжаются методами базового класса. Производные классы могут использовать методы базового класса без изменений, адаптировать их или заменять своей собственной реализацией. Как бы ни были реализованы эти методы, можно достоверно утверждать, что все объекты наследственной иерархии классов будут обладать некоторым количеством методов с одними и теми же названиями. Это явление называется *полиморфизмом*.

При использовании полиморфизма появляется возможность программировать работу объектов, абстрагируясь от их типа. Так, в транспортной сети, независимо от того, является ли текущий объект автомобильным, железнодорожным, воздушным или водным транспортным средством, он характеризуется способностью к движению, и для всех объектов "транспорт" можно ввести единый метод движения `move()`. Каждый объект из транспортной иерархии будет обладать этим методом. Несмотря на то, что движение по асфальтовой или железной дороге, по воздуху или по реке отличается, методы будут называться одинаково, и их можно будет вызывать для каждого из объектов, чей класс является наследником класса "транспорт" (рис. 22.1).

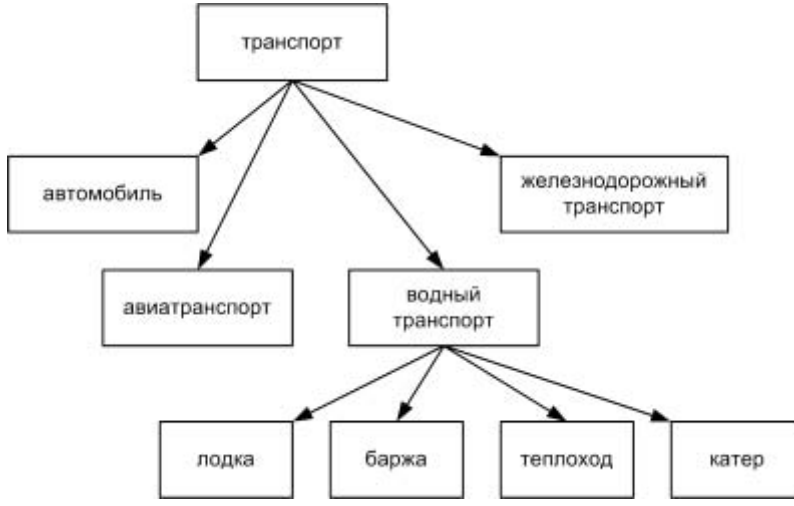


Рис. 22.1. Схема иерархии объектов "транспорт"

Скорее всего, метод `move()` придется переопределить для классов "автомобиль", "авиатранспорт", "водный транспорт" и "железнодорожный транспорт", т. к. свобода движения у этих классов транспортных средств различна и даже зависит от сезона. При дальнейшем развитии иерархии для наследников класса "водный транспорт" переопределять методы, вероятно, уже не придется, поскольку реализация метода `move()` базового класса "водный транспорт" подойдет для каждого из них.

Однако такие объемные системы, как транспортные сети, редко моделируют на PHP; чаще задачи сводятся к разработке Web-инструментария. В качестве примера рассмотрим постраничную навигацию. Объемный список неудобно отображать на странице целиком, т. к. это требует значительных ресурсов. Гораздо нагляднее выводить список, например, по 10 элементов, предоставляя ссылки на оставшиеся страницы. В большинстве случаев такая задача решается без привлечения объектно-ориентированного подхода и тем более без наследуемой иерархии и полиморфизма. Однако в Web-приложении источником списка элементов, к которому следует применить постраничную навигацию, могут выступать база данных, файл со строками, каталог с файлами изображений и т. п. Каждый раз создавать отдельную функцию для реализации постраничной навигации не всегда удобно, т. к. придется дублировать значительную часть кода в нескольких функциях. В данном случае удобнее реализовать постраничную навигацию в методе базового класса `pager`, а методы, работающие с конкретными источниками, переопределить в производных классах:

- `pager_mysql` — постраничная навигация для СУБД MySQL;
- `pager_file` — постраничная навигация для текстового файла;
- `pager_dir` — постраничная навигация для файлов в каталоге.

Иерархия классов постраничной навигации представлена на рис. 22.2.

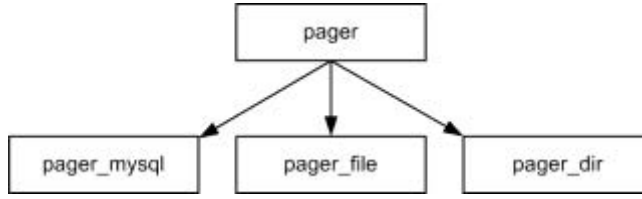


Рис. 22.2. Иерархия классов постраничной навигации

Класс `pager()` "не знает", каким образом производные классы будут узнавать общее количество элементов в списке, сколько элементов будет отображаться на одной странице, сколько ссылок находится слева и справа от текущей страницы. Поэтому помимо метода `__ToString()` класс будет содержать четыре защищенных (`protected`) метода, которые возвращают:

- `get_total()` — общее количество элементов в списке;
- `get_pnumber()` — количество элементов на странице;
- `get_page_link()` — количество элементов слева и справа от текущей страницы;
- `get_parameters()` — строку, которую необходимо передать по ссылкам на другую страницу (например, при постраничном выводе результатов поиска по ссылкам придется передавать результаты поиска).

Эти методы не имеют реализации в классе `pager` (пустые) и перегружаются производными классами, которые формируют параметры в зависимости от источника.

22.5. Абстрактные классы

Возвращаясь к иерархии классов постраничной навигации, рассмотренной в *разд. 22.4*, можно утверждать, что объект класса `pager` не имеет смысла и не может быть использован по назначению — работают только его наследники. Для того чтобы предотвратить возможность создания объекта такого класса, PHP предоставляет специальный инструмент — класс можно объявить абстрактным. Для этого объявление класса предваряют ключевым словом `abstract` (листинг 22.8).

Листинг 22.8. Объявление абстрактного класса

```
<?php
  abstract class pager
  {
    ...
  }
?>
```

Теперь попытка создать экземпляр класса `pager` (листинг 22.9) будет заканчиваться сообщением об ошибке "Fatal error: Cannot instantiate abstract class pager" ("Невозможно объявить объект абстрактного класса").

Листинг 22.9. Попытка объявления объекта абстрактного класса

```
<?php
  require_once("class.pager.php");

  // $obj = new pager(); // Ошибка
?>
```

Абстрактными следует объявлять классы, которые не существуют в реальности и объекты которых заведомо не потребуются. Например, возвращаясь к транспортной сети, объекты "транспорт", "автомобиль", "авиатранспорт", "водный транспорт", "железнодорожный транспорт" не понадобятся, и их можно сделать абстрактными. В то же время конкретные классы ("лодка", "баржа", "теплоход", "катер") могут быть полезными при оценке грузо- и пассажиропотока, расчета налогообложения и т. п. (см. рис. 22.1).

22.6. Абстрактные методы

Если во время работы над сайтом появляется новый источник данных, к которому нужно будет применить постраничную навигацию *(см. разд. 22.4)*, достаточно унаследовать от базового класса `pager` новый производный класс.

Однако продемонстрированный подход обладает некоторым изъяном: программист может забыть реализовать один из методов, который используется базовым классом. Напомним, что в базовом классе `pager` находятся только заглушки методов `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()`.

> ***Замечание***
>
> *Заглушками* в программировании называют методы, которые не выполняют никакой работы.

Для решения этой задачи в объектно-ориентированной модели PHP предусмотрены абстрактные методы, объявление которых предваряется ключевым словом `abstract`. Для абстрактных методов задают их описание и список параметров без реализации. Каждый класс, который наследуется от базового класса, содержащего абстрактные методы, обязан их реализовать. Если хотя бы один метод не реализован — работа PHP-скрипта будет остановлена, а интерпретатор сообщит о необходимости реализации абстрактного метода в производном классе. При помощи абстрактных методов исключается возможность неумышленного нарушения полиморфизма для производных классов.

В листинге 22.10 методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` класса `pager` объявляются абстрактными.

Листинг 22.10. Объявление абстрактных методов

```
<?php
  abstract class pager
  {
```

```
    abstract function get_total();
    abstract function get_pnumber();
    abstract function get_page_link();
    abstract function get_parameters();

    ...

  }
?>
```

Наличие в классе абстрактного метода требует, чтобы класс также был объявлен абстрактным.

22.7. *Final*-методы класса

Если абстрактные методы и интерфейсы предназначены для того, чтобы обеспечить обязательную перегрузку методов в производных классах, то ключевое слово `final` решает обратную задачу. Методы, объявленные в базовом классе с ключевым словом `final`, не могут быть перегружены в производном классе. В листинге 22.11 приводится пример базового класса `base`, снабженного `final`-методом `final_method()`, и производного класса `derived`.

> ***Замечание***
>
> В ранних версиях PHP допускалось объявление членов классов с ключевым словом `final`, в текущих версиях данная возможность запрещена.

Листинг 22.11. Использование ключевого слова `final`

```
<?php
  class base
  {
    public function final_method()
    {
      echo "base::final_method()";
    }
  }
  class derived extends base
  {
    // public function final_method()
    // {
    //   echo "derived::final_method()";
    // }
  }
?>
```

Замечание

Не имеет значения, в каком порядке будут перечислены ключевые слова `final` и `public`: допускается как использование последовательности `final public`, так и `public final`.

Попытка перегрузить метод `final_method()` в производном классе `derived` заканчивается ошибкой "Fatal error: Cannot override final method base::final_method()" ("Невозможно перегрузить final-метод base::final_method()").

Помимо обычных методов в качестве `final`-метода можно объявлять и специальные методы. Например, в листинге 22.12 в качестве `final`-метода объявляется конструктор класса `base`, в результате чего становится невозможным наследование производного класса `derived` с переопределенным конструктором.

Листинг 22.12. Запрет на наследование

```
<?php
  class base
  {
    public final function __construct()
    {
    }
  }
//  class derived extends base
//  {
//    Переопределяется final-метод, наследование невозможно
//    public function __construct()
//    {
//    }
//  }
?>
```

Однако прием, представленный в листинге 22.12, не является надежным способом запрета наследования от класса `base`. Достаточно не переопределять конструктор производного класса `derived`, чтобы наследование стало возможным (листинг 22.13).

Листинг 22.13. Обход запрета на наследование

```
<?php
  class base
  {
    public final function __construct()
    {
    }
  }
```

```
  class derived extends base
  {
     // final-метод не переопределяется, наследование возможно
  }
?>
```

Для того чтобы полностью запретить наследование, следует объявить `final`-класс *(см. разд. 22.8)*.

22.8. *Final*-классы

Совместно с ключевым словом `final` можно объявлять не только отдельные методы, но и целые классы. Класс, объявленный при помощи ключевого слова `final`, не может иметь наследников (листинг 22.14).

> ***Замечание***
>
> Класс не может одновременно являться и абстрактным (`abstract`), и `final`-классом.

Листинг 22.14. Объявление `final`-класса

```
<?php
  final class base
  {
    public function __construct()
    {
    }
  }
//  class derived extends base
//  {
//  }
?>
```

Попытка объявить производный класс, наследующий от `final`-класса, заканчивается сообщением об ошибке: "Fatal error: Class derived may not inherit from final class (base)" ("Класс derived не может быть унаследован от final-класса (base)").



Глава 23

Интерфейсы

Наследование и полиморфизм являются центральными идеями объектно-ориентированного программирования, позволяя наиболее эффективно организовать код для иерархических систем. Обычно в реальной практике используются лишь конечные производные классы; базовые классы иерархии необходимы лишь для сокрытия реализации и формирования общего интерфейса производных классов.

В результате базовые классы зачастую становятся абстрактными, вернее, классами, которые определяют поведение производных классов, объектов которых, однако, нельзя объявить, поскольку они не смогли бы функционировать. Абстрактные классы зачастую содержат абстрактные методы, не несущие функциональности, реализовать которую должны их потомки.

Для того чтобы не использовать классы, которые ничего кроме абстрактных методов не содержат, в PHP введена специальная конструкция — *интерфейс*, полностью состоящий из абстрактных методов. Это позволяет внести ясность в процесс разработки: классы задают поведение объектов, а интерфейсы — поведение группы классов. Класс, реализующий интерфейс, обязан реализовать методы интерфейса. Два класса, реализующих одинаковые интерфейсы, имеют одинаковый набор определяемых ими методов. Таким образом, назначение интерфейса — это реализация полиморфизма для двух классов, не имеющих общего базового класса.

23.1. Создание интерфейса

Для создания интерфейса используется ключевое слово `interface`. В листинге 23.1 демонстрируется пример интерфейса `pager` для реализации набора классов постраничной навигации. Классы, реализующие постраничную навигацию, должны содержать следующие методы:

- `get_total()` — возвращает количество элементов в списке, подвергающемся разбиению на несколько страниц;
- `get_pnumber()` — возвращает количество элементов на одной странице;
- `get_page_link()` — возвращает количество ссылок слева и справа от текущей страницы;
- `get_parameters()` — возвращает дополнительные GET-параметры, передаваемые по ссылкам.

Замечание

Интерфейс может сдержать только открытые методы; при попытке добавить в интерфейс закрытый или защищенный метод, возникает сообщение об ошибке.

Листинг 23.1. Интерфейс `pager`

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }
?>
```

Порядок создания класса, реализующего интерфейс, очень похож на наследование производного класса от базового, только вместо ключевого слова `extentds` используется ключевое слово `implements`. В листинге 23.2 объявляются два класса `pager_sample` и `pager_example`, которые реализуют интерфейс `pager`.

Листинг 23.2. Реализация интерфейса `pager` классами `pager_sample` и `pager_example`

```
<?php
   class pager_sample implements pager
   {
      public function get_total()
      {
          // ...
      }
      public function get_pnumber()
      {
          // ...
      }
      public function get_page_link()
      {
          // ...
      }
      public function get_parameters()
      {
          // ...
      }
```

```
    }
    class pager_example implements pager
    {
       public function get_total()
       {
           // ...
       }
       public function get_pnumber()
       {
           // ...
       }
       public function get_page_link()
       {
           // ...
       }
       public function get_parameters()
       {
           // ...
       }
    }
?>
```

Оба класса `pager_sample` и `pager_example` должны реализовать методы, перечисленные в интерфейсе; если хотя бы один метод остается нереализованным, выполнение скрипта заканчивается сообщением об ошибке: "Fatal error: Access type for interface method must be omitted" ("Пропущен метод интерфейса").

23.2. Интерфейсы и наследование классов

Интерфейс не исключает использование наследования. В листинге 23.3 производный класс `derived` наследуется от базового класса `base`, при этом одновременно класс `derived` реализует интерфейс `pager`, который рассматривается в *разд. 23.1*.

Листинг 23.3. Совместное использование наследования и интерфейсов

```
<?php
  require_once("interface.pager.php");

  class base
  {
    public function base_method()
    {
      echo "Метод базового класса<br>";
```

```
    }
  }
  class derived extends base implements pager
  {
    public function get_total()
    {
        // ...
    }
    public function get_pnumber()
    {
        // ...
    }
    public function get_page_link()
    {
        // ...
    }
    public function get_parameters()
    {
        // ...
    }
  }
?>
```

Объект класса `derived` позволяет вызывать не только метод `base_method()`, но и все методы интерфейса `pager`, т. к. класс `derived` обязан их реализовать.

Одна из особенностей объектно-ориентированной модели PHP заключается в том, что каждый производный класс может иметь лишь один базовый класс. Производный класс не может наследоваться сразу от двух базовых классов. Такое ограничение имеет множество объектно-ориентированных моделей. В языках программирования, допускающих множественное наследование (таких, как C++), разработчики часто сознательно отказываются от возможности множественного наследования, поскольку это усложняет и запутывает код.

Тем не менее, в большом приложении могут возникать ситуации, когда объекты разных иерархических схем должны реализовывать одинаковые методы. Это позволяет распространить явление полиморфизма на несколько не зависящих друг от друга иерархий наследуемых классов.

Рассмотрим две иерархии: товаров, предоставляемых компанией, и сотрудников, занятых в ней (рис. 23.1).

Пусть каждый из конечных классов наследуемой иерархии должен реализовывать интерфейс `pager` для обеспечения объектов классов возможностями постраничной навигации. Было бы неразумно наследовать две эти независимые иерархии от единого абстрактного класса, который бы требовал реализации методов, необходимых для по-

страничной навигации. Именно в таких случаях применяются интерфейсы, которые могут требовать реализации методов в классе, не затрагивая механизм наследования.

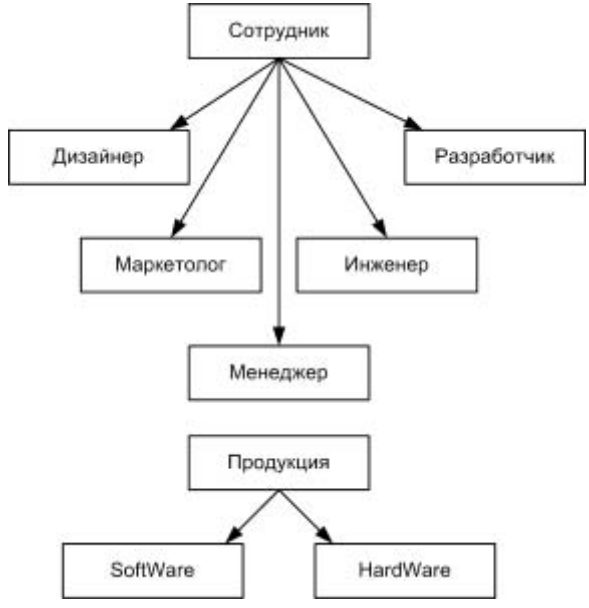


Рис. 23.1. Независимые иерархии наследуемых классов

23.3. Реализация нескольких интерфейсов

В отличие от наследования, реализация интерфейсов не ограничивается единственным интерфейсом. Класс может реализовывать любое количество интерфейсов; для этого достаточно перечислить их через запятую после ключевого слова `implements` (листинг 23.4).

> ***ЗАМЕЧАНИЕ***
>
> Именно возможность реализации нескольких интерфейсов позволяет скомпенсировать отсутствие множественного наследования в объектно-ориентированной модели языка.

Листинг 23.4. Реализация нескольких интерфейсов

```
<?php
  require_once("interface.pager.php");
  require_once("interface.print_list.php");
  require_once("interface.document.php");

  class cls implements pager, print_list, document
```

```
  {
    // ...
  }
?>
```

Как видно из листинга 23.4, класс `cls` реализует сразу три интерфейса: `pager`, `print_list` и `document`.

23.4. Проверка существования интерфейса

Прежде чем обращаться к интерфейсу, часто требуется проверить его существование. Для этого можно воспользоваться функцией `get_declared_interfaces()`, которая возвращает массив всех объявленных интерфейсов (листинг 23.5).

Листинг 23.5. Использование функции `get_declared_interfaces()`

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }

  echo "<pre>";
  print_r(get_declared_interfaces());
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 23.5 могут быть следующие строки:

```
Array
(
    [0] => Traversable
    [1] => IteratorAggregate
    [2] => Iterator
    [3] => ArrayAccess
    [4] => Serializable
    [5] => Reflector
    [6] => RecursiveIterator
    [7] => OuterIterator
    [8] => SeekableIterator
    [9] => Countable
```

```
    [10] => SplObserver
    [11] => SplSubject
    [12] => pager
)
```

Первые 12 интерфейсов являются предопределенными; последний интерфейс `pager` декларируется в элементе массива с индексом 12.

Работа с массивом, возвращаемым функцией `get_declared_interfaces()`, не всегда удобна, т. к. зачастую требуется узнать, доступен ли какой-либо конкретный интерфейс. Для этого используют функцию `interface_exists()`, которая имеет следующий синтаксис:

```
interface_exists($interface_name [, $autoload])
```

Функция возвращает `TRUE`, если интерфейс с именем *`$interface_name`* существует, и `FALSE` — в противном случае. Если необязательный параметр *`$autoload`* равен `TRUE`, функция пытается загрузить интерфейс при помощи функции `__autoload()`, если параметр равен `FALSE`, такая попытка не производится.

В листинге 23.6 приводится пример использования функции `interface_exists()`.

> ***ЗАМЕЧАНИЕ***
>
> Функция `interface_exists()` введена в PHP, начиная с версии 5.0.2.

Листинг 23.6. Использование функции `interface_exists()`

```
<?php
  require_once("interface.pager.php");

  if(interface_exists("pager"))
  {
    class cls implements pager
    {
      // ...
    }
  }
?>
```

Класс `cls` из листинга 23.6 не будет объявлен, если скрипт не найдет определение интерфейса `pager`.

23.5. Наследование интерфейсов

Интерфейсы, так же как и классы, могут наследовать друг другу при помощи ключевого слова `extends`. В листинге 23.7 производный интерфейс `extended_pager` наследует методы базового интерфейса `pager`.

Листинг 23.7. Наследование интерфейсов

```
<?php
  interface pager
  {
    public function get_total();
    public function get_pnumber();
    public function get_page_link();
    public function get_parameters();
  }

  interface extended_pager extends pager
  {
    public function get_page();
  }
?>
```

Отношение интерфейсов `pager` и `extended_pager` демонстрируется на рис. 23.2. Как и для классов, не допускается множественное наследование интерфейсов: может быть лишь один базовый интерфейс, зато допускается множество производных.

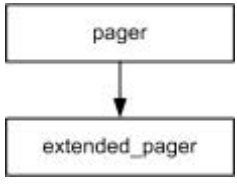


Рис. 23.2. Наследование производным интерфейсом `extended_pager` базового интерфейса `pager`

Класс, реализующий интерфейс `extended_pager`, должен перегружать не только метод `get_page()` производного класса `pager`, но и методы базового интерфейса pager: `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` (листинг 23.8).

Листинг 23.8. Реализация интерфейса `extended_pager`

```
<?php
  require_once("interface.pagers.php");

  class pager_class implements extended_pager
  {
     // ...
    public function get_total()
```

```
    {
      // ...
    }
    public function get_pnumber()
    {
      // ...
    }
    public function get_page_link()
    {
      // ...
    }
    public function get_parameters()
    {
      // ...
    }
    public function get_page()
    {
      // ...
    }
  }
?>
```

Отсутствие хотя бы одного метода из интерфейсов `pager` и `extended_pager` в классе `pager_class` приведет к ошибке.

23.6. Реализует ли объект интерфейс?

Выяснить, реализует ли объект интерфейс, можно при помощи оператора `instanceof`. Данный оператор применяется для определения принадлежности объекта классу. Для демонстрации возможностей оператора создадим два интерфейса: `first` и `second`, которые будут требовать от классов реализации одноименных методов (листинг 23.9).

Листинг 23.9. Интерфейсы `first` и `second`

```
<?php
  interface first
  {
    public function first();
  }
  interface second
  {
    public function second();
  }
?>
```

В листинге 23.10 создаются три класса: `fst` — для реализации интерфейса `first`, `snd` — интерфейса `second` и `cmn`, реализующий оба интерфейса.

Листинг 23.10. Классы `fst`, `snd` и `cmn`

```
<?php
  require_once("interface.first.php");
  require_once("interface.second.php");

  class fst implements first
  {
    public function first()
    {
      echo "Метод fst:first()<br>";
    }
  }
  class snd implements second
  {
    public function second()
    {
      echo "Метод snd:second()<br>";
    }
  }
  class cmn implements first, second
  {
    public function first()
    {
      echo "Метод cmn:first()<br>";
    }
    public function second()
    {
      echo "Метод cmn:second()<br>";
    }
  }
?>
```

В листинге 23.11 создается массив случайных объектов классов `fst`, `snd` и `cmn`. Далее при помощи оператора `instanceof` определяется, какие элементы массива реализуют интерфейс `second`, а какие нет.

Листинг 23.11. Создание массива случайных объектов

```
<?php
  // Подключаем классы
  require_once("interface.first.php");
```

```
// Формируем массив из случайных объектов
 for($i = 0; $i < 10; $i++)
 {
   switch(rand(1,3))
   {
     case 1:
       $arr[] = new fst();
       break;
     case 2:
       $arr[] = new snd();
       break;
     case 3:
       $arr[] = new cmn();
       break;
   }
 }
 // Определяем принадлежность элементов массива к классам
 foreach($arr as $obj)
 {
   if($obj instanceof second) echo "Объект класса ".get_class($obj)."
                                    реализует интерфейс second<br>";
   else echo "Объект класса ".get_class($obj)." не реализует интерфейс
              second<br>";
 }
?>
```

Следует обратить внимание, что имя класса в операторе `instanceof` не заключается в кавычки. Результатом работы скрипта из листинга 23.11 будут следующие строки:

```
Объект класса snd реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса cmn реализует интерфейс second
Объект класса fst не реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса cmn реализует интерфейс second
Объект класса fst не реализует интерфейс second
Объект класса snd реализует интерфейс second
Объект класса fst не реализует интерфейс second
```

Оператор `instanceof` возвращает значение `TRUE` не только по отношению к интерфейсу, который явно реализовывался при создании класса объекта, но и для всех базовых интерфейсов. В листинге 23.12 приводится пример интерфейса `third`, который является производным интерфейсом базового интерфейса `first`. Классы, реализующие интерфейс `third`, обязательно должны реализовывать два метода `first()` и `third()`.

Листинг 23.12. Оператор `instanceof` и производные интерфейсы

```
<?php
  require_once("interface.first.php");

  // Производный интерфейс
  interface third extends first
  {
    public function third();
  }

  // Класс, thd обязательно должен реализовывать
  // методы first() и third()
  class thd implements third
  {
    public function first()
    {
      echo "Метод thd:first()<br>";
    }
    public function third()
    {
      echo "Метод thd:third()<br>";
    }
  }

  $obj = new thd();

  if($obj instanceof first) echo "Объект реализует интерфейс first<br>";
  else echo "Объект не реализует интерфейс first<br>";

  if($obj instanceof third) echo "Объект реализует интерфейс third<br>";
  else echo "Объект не реализует интерфейс third<br>";
?>
```

Результатом работы скрипта из листинга 23.12 будут следующие строки:

```
Объект реализует интерфейс first
Объект реализует интерфейс third
```

Таким образом, оператор `instanceof` определяет принадлежность объекта не только производному интерфейсу, но и базовому.



Глава 24

Обработка ошибок и исключительных ситуаций

PHP предоставляет разработчикам два метода обработки ошибок: регистрация обработчика ошибок и механизм исключительных ситуаций. Регистрация обработчика ошибок является процедурным механизмом, в то время как исключительные ситуации более характерны для объектно-ориентированного подхода.

24.1. Обработка ошибок

Разработчику предоставляется широкий набор функций для обработки ошибок, список которых представлен в табл. 24.1.

Таблица 24.1. *Функции обработки ошибок*

Функция	Описание
`error_reporting([$level])`	Если указан необязательный параметр `$level`, функция устанавливает уровень чувствительности интерпретатора PHP. В качестве результата возвращает старый уровень
`set_error_handler($error_handler [, $error_types])`	Назначает обработчик ошибок `$error_handler` для уровня `$error_types`. Допускается многократный вызов функции `set_error_handler()`
`restore_error_handler()`	При многократном вызове функции `set_error_handler()` позволяет вернуться к предыдущему обработчику ошибок
`set_exception_handler ($exception_handler)`	Назначает обработчик по умолчанию `$exception_handler` для исключений
`restore_exception_handler()`	При многократном вызове функции `set_exception_handler()` позволяет вернуться к предыдущему обработчику ошибок
`debug_backtrace()`	Возвращает массив, элементы которого представляют точки входа, позволяющие отследить место вызова функции. Каждый элемент результирующего массива представляет собой массив, элементы которого содержат имя текущего файла, номер строки, текущую функцию и другую сопутствующую информацию

***Таблица 24.1** (окончание)*

Функция	Описание
`debug_print_backtrace()`	Функция аналогична `debug_backtrace()`, только вместо того, чтобы возвращать массив, она выводит его в окно браузера
`error_get_last()`	Возвращает массив с параметрами последней возникшей ошибки
`error_log($message [, $message_type [, $destination [, $extra_headers]]])`	Отправляет сообщение с текстом `$message` адресату `$destination`, который в зависимости от значения параметра `$message_type` может быть журнальным файлом сервера, адресом электронной почты. Если сообщение отправляется через электронную почту, через параметр `$extra_headers` можно передать дополнительные почтовые заголовки. Если сообщение успешно отправлено, функция возвращает `TRUE`, в противном случае возвращается `FALSE`
`trigger_error($error_msg [, $error_type])`	Генерирует ошибку с сообщением `$error_msg` (не более 1024 символов) и уровнем `$error_type`. Допускается использовать уровни `E_USER_ERROR`, `E_USER_WARNING` и `E_USER_NOTICE`. Попытка генерации ошибок с другими уровнями игнорируется
`user_error()`	Синоним функции `trigger_error()`

В последующих разделах каждая из функций табл. 24.1 рассматривается более подробно. Исключение составляют функции `set_exception_handler()` и `restore_exception_handler()`, подробное описание которых отложено до рассмотрения исключительных ситуаций *(см. разд. 24.2)*.

PHP имеет несколько уровней обработки ошибок, это связанно с тем, что ошибка в ядре, при разборе скрипта, его интерпретации, пользовательские ошибки относятся к разным логическим уровням, и их смешение не желательно. Такое разделение ошибок на уровни позволяет более гибко настраивать интерпретатор. В табл. 24.2 приводится список уровней и соответствующие им предопределенные константы, используемые для настройки чувствительности интерпретатора PHP к ошибкам, как в конфигурационном файле php.ini, так и из скриптов.

***Таблица 24.2.** Уровни обработки ошибок*

Константа	Значение	Описание
`E_ERROR`	1	Критическая ошибка исполнения, не совместимая с продолжением работы скрипта. Возникновение таких ошибок всегда сопровождается остановкой выполнения
`E_WARNING`	2	Предупреждения — некритическая ошибка исполнения скрипта. Возникновение таких ошибок не приводит к остановке выполнения

Таблица 24.2 *(окончание)*

Константа	Значение	Описание
E_PARSE	4	Ошибка синтаксического разбора скрипта
E_NOTICE	8	Замечание — своеобразный совет по улучшению скрипта. Выдается, в частности, если встречается неинициализированная переменная
E_CORE_ERROR	16	Критическая ошибка, возникшая при инициализации ядра PHP
E_CORE_WARNING	32	Некритическая ошибка (предупреждение), возникшая при инициализации ядра PHP
E_COMPILE_ERROR	64	Критическая ошибка, возникшая во время компиляции PHP
E_COMPILE_WARNING	128	Некритическая ошибка (предупреждение), возникшая во время компиляции PHP
E_USER_ERROR	256	Критическая ошибка пользовательского уровня, генерируемая вызовом функции trigger_error()
E_USER_WARNING	512	Некритическая ошибка (предупреждение) пользовательского уровня, генерируемая вызовом функции trigger_error()
E_USER_NOTICE	1024	Замечание пользовательского уровня, генерируемое вызовом функции trigger_error()
E_STRICT	2048	Замечание, направленное на повышение переносимости PHP-приложений между различными операционными системами
E_RECOVERABLE_ERROR	4096	Критическая ошибка, которая может быть обработана при помощи пользовательского обработчика, назначенного функцией set_error_handler(). Если обработчик не назначен, такой вид ошибок получает статус E_ERROR
E_ALL	8191	Данный уровень включает все предыдущие ошибки. До PHP 6, этот уровень не включал E_STRICT

В *разд. 3.8* подробно описывалась настройка директивы error_reporting в конфигурационном файле php.ini. При помощи этой директивы можно указать, сообщения о каких видах ошибок должны выводиться в окно браузера. В листинге 24.1 приводится фрагмент конфигурационного файла php.ini, в котором PHP предписывается выводить все сообщения (E_ALL), кроме замечаний (E_NOTICE).

Замечания

Вместо констант из табл. 24.2 можно использовать их числовые значения, однако это не рекомендуется делать, т. к. они могут изменяться от версии к версии.

Листинг 24.1. Фрагмент конфигурационного файла php.ini

```
<?php
  ...
  error_reporting  =  E_ALL & ~E_NOTICE
  ...
?>
```

Для комбинации нескольких констант, можно использовать поразрядные операторы `|`, `~`, `!`, `^` и `&`, правила работы с которыми подробно освещаются в *разд. 5.4*.

Директива `error_reporting` устанавливает чувствительность PHP на уровне сервера. Установить собственный уровень для скрипта или Web-приложения можно при помощи одноименной функции, которая имеет следующий синтаксис:

```
error_reporting([$level])
```

Если указан необязательный параметр `$level`, функция устанавливает уровень чувствительности интерпретатора PHP. В качестве результата возвращает старый уровень (в виде числового значения).

В листинге 24.2 приводится пример использования функции `error_reporting()` для установки локального уровня чувствительности интерпретатора PHP.

Листинг 24.2. Использование функции `error_reporting()`

```
<?php
  error_reporting (E_ALL & ~E_NOTICE)
?>
```

Если возникает ошибка и в конфигурационном файле php.ini или в самом скрипте включено отображение сообщений об ошибке, в окно браузера выводится сообщение об ошибке, файл и строка ее возникновения. Пусть файл index.php содержит ошибку, например вызов неизвестной функции (листинг 24.3).

Листинг 24.3. Файл index.php

```
<?php
  // Вызываем неизвестную функцию
  error1();
?>
```

Вызов неизвестной функции приводит к возникновению ошибки уровня `E_ERROR`, в результате чего в окно браузера выводится сообщение, представленное на рис. 24.1.

Помимо ошибок, генерируемых интерпретатором PHP, разработчики могут генерировать собственные ошибки при помощи функции `trigger_error()`, которая имеет следующий синтаксис:

```
trigger_error($error_msg [, $error_type])
```

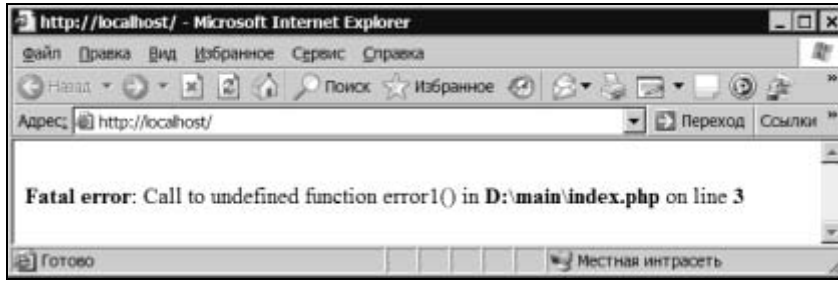


Рис. 24.1. Вывод сообщения о критической ошибке

Первый параметр `$error_msg` содержит текст сообщения об ошибке, второй необязательный параметр `$error_type` может задавать уровень ошибки (если параметр не задан, устанавливается `E_USER_NOTICE`). Функция `trigger_error()` позволяет сгенерировать только ошибки пользовательского уровня, т. е. `E_USER_ERROR`, `E_USER_WARNING` и `E_USER_NOTICE`. Если задан один из этих уровней, функция возвращает `FALSE`, в противном случае возвращается `TRUE`.

В листинге 24.4 приводится пример использования функции `trigger_error()`. В функции `print_age()`, пользовательская ошибка `E_USER_ERROR` генерируется, если аргумент `$age` является отрицательным.

Листинг 24.4. Использование функции `trigger_error()`

```
<?php
  // Печать возраста
  function print_age($age)
  {
    $age = intval($age);
    if ($age < 0)
    {
      trigger_error("Функция print_age(): ".
                    "возраст не может быть".
                    " отрицательным", E_USER_ERROR);
    }
    echo "Возраст составляет: $age";
  }

  // Вызов функции с отрицательным аргументом
  print_age(-10);
?>
```

Попытка выполнения скрипта из листинга 24.4 приведет к срабатыванию функции `trigger_error()` (рис. 24.2).

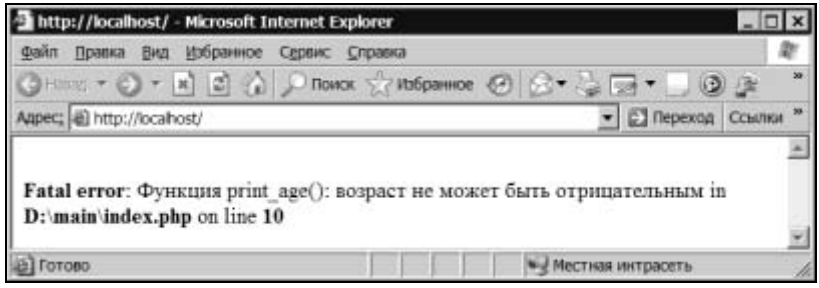


Рис. 24.2. Вывод сообщения о пользовательской критической ошибке

За вывод сообщения в окно браузера отвечает предопределенный обработчик ошибок, встроенный в интерпретатор PHP. Однако при помощи функции `set_error_handler()` можно назначить собственный обработчик и перегрузить реакцию на возникновение ошибки. Функция имеет следующий синтаксис:

```
set_error_handler($error_handler [, $error_types])
```

В качестве первого параметра `$error_handler` функция принимает имя функции-обработчика. Через второй необязательный параметр `$error_types` можно указывать уровень ошибок, которому назначается обработчик. Последний параметр может принимать не все значения, уровни `E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR`, `E_COMPILE_WARNING` и большинство ошибок уровня `E_STRICT` не могут обрабатываться при помощи пользовательских обработчиков.

Функция-обработчик, имя которой передается в качестве первого параметра функции `set_error_handler()`, имеет следующий синтаксис:

```
handler($errno, $errstr [, $errfile [, $errline [, $errcontext]]])
```

Параметры функции-обработчика заполняются интерпретатором PHP. Они имеют следующие значения:

- `$errno` — уровень ошибки, например, `E_USER_ERROR`, `E_USER_WARNING` и `E_USER_NOTICE` и т. п.;
- `$errstr` — сообщение об ошибке;
- `$errfile` — путь к файлу, в котором возникла ошибка;
- `$errline` — номер строки в файле, где произошла ошибка;
- `$errcontext` — массив с переменными, доступными в области видимости возникновения ошибки;

В листинге 24.5 определяется пользовательская функция — обработчик ошибок `error_handler()`, которая назначается в качестве текущего обработчика при помощи функции `set_error_handler()`.

Листинг 24.5. Назначение пользовательского обработчика ошибок

```
<?php
  // Функция-обработчик ошибок
  function error_handler($errno, $errstr, $errfile, $errline)
```

```
{
  switch ($errno)
  {
    case E_USER_ERROR:
      // Критическая ошибка
      echo "<b>Критическая ошибка пользовательского уровня</b> ".
           "[$errno] $errstr<br> Ошибка произошла в файле ".
           "$errfile в строке $errline<br>";
      // Завершаем работу скрипта
      exit();
      break;

    case E_USER_WARNING:
      // Некритическая ошибка
      echo "<b>Предупреждение пользовательского уровня</b> ".
           "[$errno] $errstr<br>";
      break;

    case E_USER_NOTICE:
      // Замечание
      echo "<b>Замечание пользовательского уровня</b> ".
           "[$errno] $errstr<br>";
      break;

    default:
      // Все остальные типы ошибок
      echo "Ошибка неизвестного типа: [$errno] $errstr<br>";
      break;
  }

  // Возвращаем TRUE, чтобы предотвратить выполнение
  // встроенного обработчика PHP
  return TRUE;
}

// Функция деления чисел
function division($dividend, $divisor)
{
  // Если делимое и делитель не являются числами,
  // генерируем замечание
  if (!is_numeric($dividend))
  {
    trigger_error("Функция division(), делимое не является числом",
                  E_USER_NOTICE);
```

```
    }
    if (!is_numeric($divisor))
    {
      trigger_error("Функция division(), делитель не является числом",
                    E_USER_NOTICE);
    }

    // Если делитель является целым числом, генерируем предупреждение
    if (is_integer($divisor))
    {
      trigger_error("Функция division(), делитель является целым ".
                    "числом, возможно отбрасывание дробной части",
                    E_USER_WARNING);
    }

    // Если делитель равен нулю, генерируем критическую ошибку
    if ($divisor == 0)
    {
      trigger_error("Функция division(), делитель равен 0", E_USER_ERROR);
      return NULL;
    }

    return $dividend / $divisor;
  }

  // Устанавливаем в качестве обработчика ошибок
  // функцию error_handler()
  set_error_handler("error_handler");

  // Генерация предупреждения
  echo division(10, 2)."<br>";

  // Генерация замечания
  echo division("", 2)."<br>";

  // Генерация критической ошибки
  echo division(10, 0)."<br>";
?>
```

Функция `division()`, принимающая в качестве первого аргумента делимое `$dividend`, а в качестве второго аргумента `$divisor` делитель, возвращает частное. При этом, если делитель принимает значение 0, функция генерирует критическую ошибку `E_USER_ERROR`, если делитель является целым числом — предупреждение `E_USER_WARNING`, если хотя бы один из аргументов не является числом — замечание

E_USER_NOTICE. На рис. 24.3 приводится результат выполнения скрипта из листинга 24.5. Как видно из листинга 24.5 и рис. 24.3, при использовании собственного обработчика ошибок можно добиться вывода сообщений полностью на русском языке.

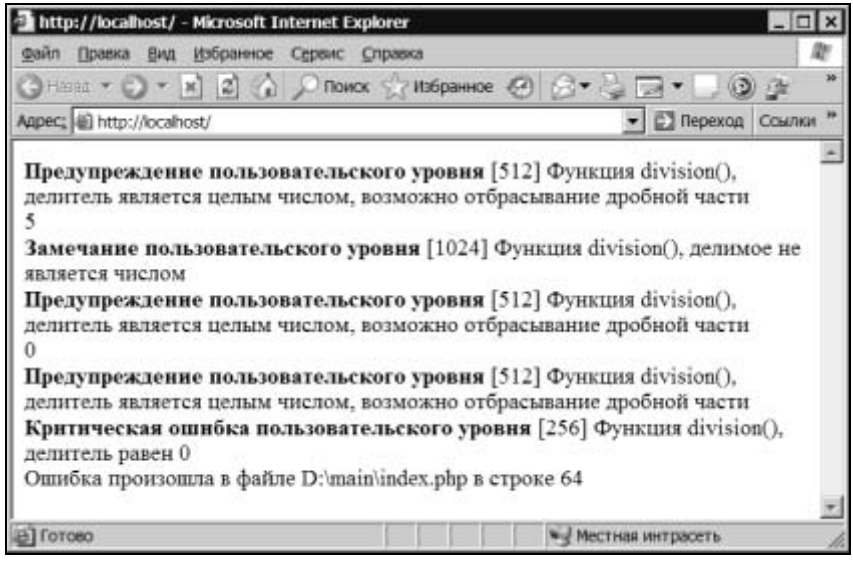


Рис. 24.3. Назначение собственного обработчика ошибок

Функция set_error_handler() возвращает название предыдущего обработчика ошибок, для встроенного обработчика в качестве имени зарезервирована пустая строка. Повторный вызов функции set_error_handler() не приводит к сбросу предыдущего обработчика, он запоминается в стеке, и к нему всегда можно вернуться, вызвав функцию restore_exception_handler() (листинг 24.6).

Листинг 24.6. Использование функции restore_exception_handler()

```
<?php
  // Устанавливаем первый обработчик ошибок
  $error = set_error_handler("error_handler"); // Пустая строка

    // Устанавливаем второй обработчик ошибок
    $error_next = set_error_handler("error_handler"); // error_handler

    // Откатываемся к первому обработчику ошибок
    restore_exception_handler();

  // Откатываемся к исходному обработчику ошибок
  restore_exception_handler();
?>
```

При генерации сообщений об ошибках выводится имя файла и номер строки, в которой произошла ошибка, к сожалению, ничего не сообщается о том, как интерпретатор достиг данной точки программы. Для решения этой проблемы предназначены функции `debug_backtrace()` и `debug_print_backtrace()`.

Функция `debug_backtrace()` возвращает массив, элементы которого представляют точки входа, позволяющие отследить место вызова функции. Каждый элемент результирующего массива представляет точку входа и реализован в виде массива, элементы которого описываются в табл. 24.3.

***Таблица 24.3.** Элементы результирующего массива функции `debug_backtrace()`*

Элемент массива	Описание
`function`	Название текущей функции
`line`	Номер текущей строки
`file`	Название текущего файла
`class`	Название текущего класса
`object`	Текущий объект
`type`	Тип вызова: динамический при помощи последовательности `->` или статический при помощи последовательности `::`
`args`	Если ошибка возникла внутри функции, массив `$args` содержит список аргументов функции

Пусть файл index.php (листинг 24.7) содержит вызов файла include.php (листинг 24.8), который реализует и вызывает функцию `test()`.

Листинг 24.7. Файл index.php

```
<?php
  require_once("include.php");
?>
```

Листинг 24.8. Файл include.php

```
<?php
  function test($par)
  {
    // Получаем путь до текущей точки
    $arr = debug_backtrace();
    // Выводим дамп с путем к текущей точке
    echo "<pre>";
    print_r($arr);
```

```
    echo "</pre>";
  }

  test(-10);
?>
```

Как видно из листинга 24.8, функция `test()` вызывает функцию `debug_backtrace()`, возвращающую следующий дамп:

```
Array
(
    [0] => Array
        (
            [file] => D:\main\include.php
            [line] => 12
            [function] => test
            [args] => Array
                (
                    [0] => -10
                )
        )
    [1] => Array
        (
            [file] => D:\main\index.php
            [line] => 2
            [args] => Array
                (
                    [0] => D:\main\include.php
                )
            [function] => require_once
        )
)
```

Если вызов функции `debug_backtrace()` заменить вызовом функции `debug_print_backtrace()`, результат может выглядеть следующим образом:

```
#0  test(-10) called at [D:\main\include.php:8]
#1  require_once(D:\main\include.php) called at [D:\main\index.php:2]
```

Еще одной функцией, заслуживающей внимания, является функция `error_log()`, которая позволяет документировать возникающие ошибки и имеет следующий синтаксис:

```
error_log($message [, $message_type [,
          $destination [, $extra_headers]]])
```

Функция отправляет сообщение с текстом `$message` адресату `$destination`, который в зависимости от значения параметра `$message_type` может быть журнальным файлом

сервера, адресом электронной почты. Допустимые значения параметра `$message_type` представлены в табл. 24.4.

Если сообщение отправляется посредством электронной почты, через параметр `$extra_headers` можно передать дополнительные почтовые заголовки. Если сообщение успешно отправлено, функция возвращает `TRUE`, в противном случае возвращается `FALSE`.

***Таблица 24.4.** Значения параметра `$message_type` функции `error_log()`*

Значение	Описание
0	Отправлять сообщение об ошибке в системный журнал или файл, который определяется директивой `error_log` конфигурационного файла php.ini
1	Отправить сообщение по электронной почте на адрес, указанный в параметре `$destination`
2	Отправить сообщение через отладочный канал, по IP-адресу или хосту, указанному в параметре `$destination`. В настоящий момент эта возможность не доступна
3	Добавить сообщение в файл, указанный в параметре `$destination`. Следует иметь в виду, что о переводе строки разработчику следует заботиться самостоятельно

Преобразуем обработчик из листинга 24.5 таким образом, чтобы все сообщения об ошибках помещались в файл error.log, а сообщения о критических ошибках дублировались по электронной почте **admin@site.ru**. Кроме этого, каждое сообщение об ошибках снабдим меткой времени, сигнализирующей о времени возникновения ошибки (листинг 24.9).

Листинг 24.9. Использование функции `error_log()`

```
<?php
  // Функция-обработчик ошибок
  function error_handler($errno, $errstr, $errfile, $errline)
  {
    switch ($errno)
    {
      case E_USER_ERROR:
        // Критическая ошибка
        $str = "<b>Критическая ошибка пользовательского уровня</b> ".
             "[$errno] $errstr<br> Ошибка произошла в файле ".
             "$errfile в строке $errline<br>\r\n";
        // Выводим сообщение об ошибке
        echo $str;
```

```
        // Сообщаем о критической ошибке администратору по почте
        error_log(date("d.m.Y H:i").$str, 1, "admin@site.ru");
        // Помещаем сообщение об ошибке в файл error.log
        error_log(date("d.m.Y H:i").$str, 3, "error.log");
        // Завершаем работу скрипта
        exit();
        break;

      case E_USER_WARNING:
        // Некритическая ошибка
        $str = "<b>Предупреждение пользовательского уровня</b> ".
               "[$errno] $errstr<br>\r\n";
        // Выводим сообщение об ошибке
        echo $str;
        // Помещаем сообщение об ошибке в файл error.log
        error_log(date("d.m.Y H:i").$str, 3, "error.log");
        break;

      case E_USER_NOTICE:
        // Замечание
        $str = "<b>Замечание пользовательского уровня</b> ".
               "[$errno] $errstr<br>\r\n";
        // Помещаем сообщение об ошибке в файл error.log
        error_log(date("d.m.Y H:i").$str, 3, "error.log");
        break;

      default:
        // Все остальные типы ошибок
        $str = "Ошибка неизвестного типа: [$errno] $errstr<br>\r\n";
        // Помещаем сообщение об ошибке в файл error.log
        error_log(date("d.m.Y H:i").$str, 3, "error.log");
        break;
    }

    // Возвращаем TRUE, чтобы предотвратить выполнение
    // встроенного обработчика PHP
    return TRUE;
  }
  // Далее по листингу 24.5
  ...
?>
```

24.2. Обработка ошибок и исключения

Исключения не являются непременным атрибутом объектно-ориентированного подхода, однако сопровождают каждый объектно-ориентированный язык де-факто. Дело в том, что объектно-ориентированный подход серьезно изменяет структуру кода. Разработчик отчасти сам становится автором нового языка программирования, разрабатывая классы в рамках предметной области. Структурные изменения кода требуют новых способов обработки ошибок (нештатных ситуаций).

Разработка класса происходит на абстрактном уровне: при этом создается не конкретная область памяти, а лишь определяется поведение объектов. Класс выступает инструментом, который, как и язык программирования, может применяться в совершенно разных областях и приложениях. Обработка нештатных ситуаций, ошибок как кода, так и ввода данных может быть различной для разных приложений: где-то достаточно вывести сообщение при помощи конструкции `echo`; где-то сообщение следует оформить в виде HTML-страницы с дизайном, согласованным с остальными страницами приложения; где-то сообщение об ошибке должно быть помещено в журнал (в файл или базу данных). Предусмотреть заранее формат ошибки невозможно, и любой формат будет сужать область применения класса и возможность его повторного использования.

Выходом из ситуации является разделение кода класса и кода обработки ошибок, что достигается при помощи специального механизма — исключений. Разработчик класса может сгенерировать исключение, а пользователь класса может его обработать по своему усмотрению.

Разработчики могут проектировать собственные исключения, которые являются классами, при этом генерация исключений сводится к передаче объекта исключения из точки возникновения нештатной ситуации в обработчик исключений.

Удобство применения исключительных ситуаций в объектно-ориентированном программировании вовсе не означает, что следует пренебрегать процедурными средствами обработки ошибок. В частности, в PHP имеется развитая система отслеживания и контроля ошибок, использование которой в ряде случаев удобнее и эффективнее исключений.

24.2.1. Синтаксис исключений

Для реализации механизма исключений в PHP введены следующие ключевые слова: `try` (контролировать), `throw` (генерировать) и `catch` (обрабатывать).

> ***ЗАМЕЧАНИЕ***
>
> Механизм исключений не обязательно должен быть привязан к объектно-ориентированной системе, допускается использование исключений применительно к структурному коду.

Ключевое слово `try` позволяет выделить в любом месте скрипта так называемый *контролируемый блок*, за которым следует один или несколько блоков обработки исключений, реализуемых с помощью ключевого слова `catch` (листинг 24.10).

Листинг 24.10. Создание контролируемого блока

```
<?php
  try
  {
    // Операторы
    ...
    // Генерация исключений
    throw Выражение_генерации_исключения;
    ...
    // Операторы
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
  }
?>
```

Обработчик (или обработчики) всегда располагаются после контролируемого оператором `try` блока кода. Среди операторов контролируемого блока могут быть любые операторы и объявления PHP. Если в теле контролируемого блока исключение генерируется при помощи ключевого слова `throw`, то интерпретатор PHP переходит в `catch`-обработчик. В листинге 24.11 представлен скрипт, в котором по случайному закону либо генерируется, либо не генерируется исключение.

Листинг 24.11. Генерация исключения по случайному закону

```
<?php
  try
  {
    // Генерируем исключение по случайному закону
    if (rand(0,1))
    {
      // Генерация исключений
      throw new Exception();
    }
  }
  catch(Exception $exp)
  {
    // Фраза выводится, если было сгенерировано исключение
    exit("Произошла исключительная ситуация");
  }
  // Фраза выводится, если исключение не генерировалось
  echo "Штатная работа скрипта";
?>
```

В зависимости от того, возвращает функция `rand()` 0 или 1, выводится либо фраза "Произошла исключительная ситуация", либо фраза "Штатная работа скрипта". Следует обратить внимание, что если исключение не генерируется, то код в `catch`-блоках не выполняется.

В качестве исключения выступает объект класса `Exception`, который создается при помощи ключевого `new` непосредственно при вызове оператора `throw`. Однако объект можно подготовить заранее, как это продемонстрировано в листинге 24.12.

Листинг 24.12. Предварительное создание объекта исключения

```
<?php
  try
  {
    if (rand(0,1))
    {
      // Подготовка исключения
      $obj = new Exception();
      // Генерация исключений
      throw $obj;
    }
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    exit("Произошла исключительная ситуация");
  }
  echo "Штатная работа скрипта";
?>
```

При генерации исключения ключевое слово `throw` принимает объект класса `Exception` или производного класса.

> ***ЗАМЕЧАНИЕ***
>
> Создание собственных производных классов, наследующих от класса `Exception`, рассматривается в *разд. 24.2.8*.

24.2.2. Интерфейс класса *Exception*

Для того чтобы эффективно использовать класс `Exception`, следует познакомиться с его интерфейсом. В табл. 24.5 представлены члены класса `Exception`, объявленные со спецификатором доступа `protected`, доступ к которым можно получить при помощи методов класса, а также из производных классов.

ЗАМЕЧАНИЕ

Класс `Exception` относится к так называемым предопределенным классам, т. е. классам, которые реализованы в PHP-интерпретаторе и не требуют реализации со стороны программиста. Другие предопределенные классы более подробно рассматриваются в *главе 26*.

Таблица 24.5. *Защищенные члены класса* `Exception`

Член	Описание
`$this->message`	Текстовое сообщение, описывающее исключительную ситуацию
`$this->code`	Числовой код, присвоенный данному типу исключительных ситуаций
`$this->file`	Имя файла, в котором произошла исключительная ситуация
`$this->line`	Номер строки файла `$this->file`, в которой произошла исключительная ситуация

Так как члены класса `Exception` объявлены со спецификатором `protected`, доступ к ним осуществляется при помощи методов класса, описание которых представлено в табл. 24.6.

Таблица 24.6. *Методы класса* `Exception`

Метод	Описание
`public function __construct( $message = null, $code = 0)`	Конструктор класса, инициализирующий члены `$message` и `$code`, оба параметра необязательны
`final function getMessage()`	Метод, возвращающий текстовое сообщение члену `$this->message`
`final function getCode()`	Метод, возвращающий числовой код `$this->code`, характеризующий исключительную ситуацию
`final function getFile()`	Метод, возвращающий имя файла `$this->file`, в котором произошла исключительная ситуация
`final function getLine()`	Метод, возвращающий номер строки `$this->line`, в которой произошла исключительная ситуация
`final function getTrace()`	Стек обработки исключительной ситуации в виде массива
`final function getTraceAsString()`	Стек обработки исключительной ситуации в виде строки
`function __toString()`	Перегрузка метода `__toString()`, возвращающего строку при использовании объекта в строковом контексте

Класс `Exception` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 24.13.

Листинг 24.13. Гипотетическая реализация класса `Exception`

```
<?php
  class Exception
  {
    // Текстовое сообщение, описывающее исключительную ситуацию
    protected $message = 'Unknown exception';
    // Числовой код, назначенный данному типу исключительных ситуаций
    protected $code = 0;
    // Имя файла, в котором произошла исключительная ситуация
    protected $file;
    // Номер строки, в которой произошла исключительная ситуация
    protected $line;

    // Конструктор класса, инициализирующий члены
    // $message и $code - оба параметра реализованы как необязательные
    public function __construct($message = null, $code = 0);

    // Метод, возвращающий текстовое сообщение $this->message
    public final function getMessage();
    // Метод, возвращающий числовой код $this->code,
    // характеризующий исключительную ситуацию
    public final function getCode();
    // Метод, возвращающий имя файла $this->file,
    // в котором произошла исключительная ситуация
    public final function getFile();
    // Метод, возвращающий номер строки $this->line,
    // в которой произошла исключительная ситуация
    public final function getLine();
    // Стек обработки исключительной ситуации в виде массива
    public final function getTrace();
    // Стек обработки исключительной ситуации в виде строки
    public final function getTraceAsString();

    // Перегрузка метода __toString(), возвращающего строку
    // при использовании объекта в строковом контексте
    public function __toString();
  }
?>
```

Как видно из табл. 24.5 и листинга 24.13, конструктор класса `Exception` позволяет инициализировать защищенные члены `$message` и `$code`, что бывает не лишним, если в контролируемом блоке генерируется несколько исключений (листинг 24.14).

Листинг 24.14. Передача сообщения и кода из точки генерации исключения

```
<?php
  try
  {
    $code = rand(0,1);
    if (!$code)
    {
      // Генерация исключений
      throw new Exception("Первая точка входа", $code);
    }
    else
    {
      // Генерация исключений
      throw new Exception("Вторая точка входа", $code);
    }
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
  }
?>
```

В зависимости от того, возвращает функция генерации случайного значения `rand()` число 0 или 1, скрипт из листинга 24.14 может выводить в окно браузера либо последовательность строк для первого оператора `throw`:

```
Исключение 0 : Первая точка входа
в файле D:\main\oop\08\index.php
в строке 8
```

либо для второго:

```
Исключение 1 : Вторая точка входа
в файле D:\main\oop\08\index.php
в строке 13
```

Таким образом, можно однозначно определить, в каком контексте было сгенерировано исключение, даже если контролируемый блок `try` содержит несколько вызовов оператора `throw`.

24.2.3. Генерация исключений в функциях

В предыдущих разделах механизм исключений демонстрировался в простейших ситуациях, когда для обработки внештатных ситуаций более уместны обычные логические операторы `if` и `switch`. Оценить полезность данного механизма можно на примере сложного скрипта, вызывающего множество функций, обработка внештатных ситуаций в которых должна производиться в одной точке.

Разработаем функцию `print_user()`, которая извлекает из базы данных список пользователей и выводит их в окно браузера (листинг 24.15).

Листинг 24.15. Функция `print_user()`

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Определяем константы для кодов исключения
  define("SQL_ERROR", 100);
  define("NONE_RECORDS", 101);

  function print_user()
  {
    $query = "SELECT * FROM object_user";
    $usr = mysql_query($query);
    // Генерируем исключение в случае ошибки выполнения SQL-запроса
    if (!$usr) throw new Exception(mysql_error(), SQL_ERROR);
    // Генерируем исключение, если в таблице нет ни одной записи
    if (!mysql_num_rows($usr))
    {
      throw new Exception("Отсутствуют записи в таблице", NONE_RECORDS);
    }

    // Выводим список пользователей
    while($user = mysql_fetch_array($usr))
    {
      echo $user['name']."<br>";
    }
  }
?>
```

Функция генерирует два исключения: первое — при возникновении ошибки в SQL-запросе, второе — если в таблице `object_user` нет ни одной записи. В случае возникновения ошибки в SQL-запросе через объект класса `Exception` передается текстовое

описание возникшей проблемы, получаемое посредством функции `mysql_error()`, а также константа `SQL_ERROR`, значение которой определяется в том же файле. Если обнаруживается, что таблица не содержит ни одной записи, через объект класса `Exception` в обработчик передается текстовое сообщение "Отсутствуют записи в таблице", а также константа `NONE_RECORDS`.

> ***ЗАМЕЧАНИЕ***
>
> Для числовых кодов предпочтительнее вводить константы: это позволяет значительно увеличить читабельность и масштабируемость кода. Для изменения значения константы (например, чтобы исключить конфликт значений), достаточно исправить код в одной точке. С другой стороны, если используются так называемые "магические числа", т. е. непосредственные значения чисел вместо констант, придется произвести десятки, а то и сотни исправлений, для чего, вероятно, потребуется много времени, не говоря уже о том, что это может вызвать появление ошибок в коде.

В скрипте из листинга 24.15 соединение с СУБД MySQL устанавливается при помощи вспомогательного файла config.php, обработку внештатных ситуаций в котором можно также реализовать при помощи исключений (листинг 24.16).

Листинг 24.16. Установка соединения с СУБД MySQL

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "oop";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Определяем константы для кодов исключения
  define("NOT_CONNECTION", 0);
  define("NOT_DATABASE", 1);

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    throw new Exception("Невозможно установить соединение с сервером MySQL ",
                        NOT_CONNECTION);
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) )
  {
```

```
    throw new Exception("Ошибка выбора базы данных", NOT_DATABASE);
  }

  // Устанавливаем кодировку соединения; следует выбрать ту кодировку,
  // в которой данные будут отправляться серверу MySQL
  @mysql_query("SET NAMES 'cp1251'");
?>
```

Как видно из листинга 24.16, вместо того чтобы непосредственно выводить сообщения в окно браузера в случае возникновения ошибочных ситуаций, генерируются исключения. Для кодов исключения также вводятся константы `NOT_CONNECTION` и `NOT_DATABASE`, для ошибки отсутствия соединения и ошибки выбора базы данных, соответственно.

Генерация исключений позволяет "отложить" обработку ошибочных ситуаций и разделить логику приложения и логику обработки ошибок. В листинге 24.17 представлен возможный вызов функции `print_user()`.

Листинг 24.17. Вызов функции `print_user()`

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
    // Вызываем функцию
    print_user();
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
  }
?>
```

Важно отметить, что все совершенно разноплановые ошибки, которые возникают в функции `print_user()`, а также файле function.print_user.php и всех вложенных в него файлах, обрабатываются не только в одном месте, но и одинаково. Например, если разработчик указал неправильное имя базы, то скрипт из листинга 24.17 может вывести следующие строки:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
```

После того как создание Web-приложения завершено, разработчик получает возможность изменить формат вывода сообщения об ошибке, исправив лишь один `catch`-блок (листинг 24.18).

Листинг 24.18. Изменение формата вывода сообщения об ошибке

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
    // Вызываем функцию
    print_user();
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    $url = "error.php?id={$exp->getCode()}";
    echo "<HTML><HEAD>
           <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$url'>
          </HEAD></HTML>";
  }
?>
```

В листинге 24.18 осуществляется переадресация на страницу error.php с передачей кода состояния исключения через GET-параметр `id`. Страницу error.php можно оформить в стиле Web-приложения, уменьшить количество технической информации и добавить извинения.

Листинг 24.19. Файл error.php

```
<?php
  echo "К сожалению, в Web-приложении произошла ошибка номер
        $_GET[id]. Пожалуйста, сообщите об этом администрации
        сайта по адресу site@site.ru.";
?>
```

24.2.4. Стек обработки исключительной ситуации

В предыдущем разделе демонстрировалось, как исключение, сгенерированное внутри функции или вложенного файла, может обрабатываться единственным обработчиком. При помощи функций `getFile()` и `getLine()` были получены имя файла и номер строки, в которых сгенерировано исключение. Однако файл при помощи инструкций `include`, `include_once`, `require` и `require_once` может включаться во множество

других файлов проекта, поэтому локализация и воспроизведение ошибки не всегда возможны. В этой ситуации оказываются полезными функции `getTrace()` и `getTraceAsString()`, которые возвращают стек передачи исключения из файлов и функций в обработчик. Функция `getTrace()` возвращает стек в виде массива, а функция `getTraceAsString()` — в виде строки.

Перепишем вызов функции `print_user()` из листинга 24.17 так, чтобы в обработчике при помощи метода `getTraceAsString()` возвращался стек передачи исключительной ситуации (листинг 24.20).

Листинг 24.20. Использование функции `getTraceAsString()`

```
<?php
  try
  {
    // Подключаем реализацию функции print_user()
    require_once("function.print_user.php");
    // Вызываем функцию
    print_user();
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

В случае указания неправильного имени базы данных `catch`-обработчик из листинга 24.20 может вывести следующий отчет:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
#0 D:\main\oop\08\function.print_user.php(3): require_once()
#1 D:\main\oop\08\index.php(5): require_once('D:\main\oop\08\...')
#2 {main}
```

Если исключение произойдет на более высоком уровне вложенности, то количество пунктов в массиве, возвращаемом методом `getTraceAsString()`, возрастает. В случае

использования функции `getTrace()` стек передачи исключения возвращается в виде многомерного массива:

```
Исключение 1 : Ошибка выбора базы данных
в файле D:\main\oop\08\config.php
в строке 24
Array
(
    [0] => Array
        (
            [file] => D:\main\oop\08\function.print_user.php
            [line] => 3
            [function] => require_once
        )
    [1] => Array
        (
            [file] => D:\main\oop\08\index.php
            [line] => 5
            [args] => Array
                (
                    [0] => D:\main\oop\08\function.print_user.php
                )
            [function] => require_once
        )
)
```

24.2.5. Генерация исключений в классах

Обработка внештатных ситуаций и ошибок при помощи исключений приобретает еще большее значение в случае классов. Если ошибка возникает внутри объекта, зачастую выводить сообщение в окно браузера невозможно: это может нарушить дизайн приложения, для этого придется дублировать систему ошибок приложения и т. п. Единственный правильный выход в подобной ситуации — переложить ответственность за обработку внештатных ситуаций внутри объекта на плечи внешнего разработчика.

В качестве примера рассмотрим класс `employee` (сотрудник), который реализует специальные методы `__get()` и `__set()` для обращений к закрытым членам. Напомним его упрощенную реализацию (листинг 24.21).

Листинг 24.21. Упрощенная реализация класса `employee`

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
```

```
      $this->surname    = $surname;
      $this->name       = $name;
      $this->patronymic = $patronymic;
      $this->age        = $age;
    }
    private function __get($index)
    {
      return $this->$index;
    }
    private function __set($index, $value)
    {
      if (isset($this->$index))
      {
          $this->$index = $value;
      }
    }

    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

Класс содержит четыре закрытых члена: `$surname` (фамилия), `$name` (имя), `$patronymic` (отчество) и `$age` (возраст). Значения закрытых членов устанавливаются при помощи конструктора класса и могут быть изменены при помощи метода `__set()`, который при этом проверяет существование запрашиваемого члена. Если член с таким именем существует — производится изменение его значения, если не существует — состояние объекта остается неизменным. Такая проверка необходима, т. к. обращение к несуществующему члену класса автоматически приводит к его созданию.

Обращение к несуществующему члену в рамках метода `__get()` не так критично, поскольку метод не изменяет состояние объекта.

Однако обращение к несуществующему члену как в методе `__set()` практически наверняка вызовет ошибку на стороне внешнего разработчика, поэтому крайне полезно сгенерировать исключение, которое сигнализировало бы внешнему разработчику об ошибке с его стороны (листинг 24.22).

Листинг 24.22. Генерация исключения при обращении к несуществующему члену

```
<?php
  class employee
  {
```

```
    public function __construct($surname, $name, $patronymic, $age = 18)
    {
      $this->surname    = $surname;
      $this->name       = $name;
      $this->patronymic = $patronymic;
      $this->age        = $age;
    }
    private function __get($index)
    {
      if (isset($this->$index))
      {
        return $this->$index;
      }
      else new Exception("Член $index не существует");
    }
    private function __set($index, $value)
    {
      if (isset($this->$index))
      {
          $this->$index = $value;
      }
      else new Exception("Член $index не существует");
    }

    private $surname;
    private $name;
    private $patronymic;
    private $age;
  }
?>
```

Теперь достаточно поместить код, обращающийся к объектам класса `employee`, в контролируемый блок, чтобы предотвратить попытку обращения к несуществующим членам (листинг 24.23).

Листинг 24.23. Попытка обращения к несуществующему члену класса `employee`

```
<?php
  // Подключаем реализацию класса employee
  require_once("class.employee.php");

  try
  {
```

```
    // Объявляем объект класса employee
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    // Производим попытку изменения несуществующего члена класса $var,
    // что вызывает генерацию исключения
    $obj->var = 100;
    // Остальные операторы
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение: {$exp->getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

В листинге 24.23 производится попытка осуществить доступ к несуществующему члену `$obj`, в результате чего в методе `__set()` генерируется исключение. Перехват его в `catch`-блоке приводит к формированию отчета вида:

```
Исключение: Член var не существует
в файле D:\main\oop\08\class.employee.php
в строке 25
#0 D:\main\oop\08\index.php(12): employee->__set('var', 100)
#1 {main}
```

Важная особенность заключается в том, что оператор переходит в `catch`-блок сразу после возникновения исключительной ситуации, т. е. операторы, следующие за конструкцией `$obj->var`, выполнены не будут.

Как уже упоминалось, PHP-объекты существуют до конца работы скрипта, таким образом, объект `$obj` будет доступен в том числе и в `catch`-обработчике (листинг 24.24).

> ***Замечание***
>
> В большинстве объектно-ориентированных языков программирования, реализующих механизм исключительных ситуаций, к моменту выполнения `catch`-обработчика объект, как правило, оказывается уже разрушенным. Объект исключения `Exception` необходим, чтобы передать в обработчик информацию о случившемся.

Листинг 24.24. Объект доступен в `catch`-обработчике

```
<?php
  // Подключаем реализацию класса employee
  require_once("class.employee.php");
```

```
  try
  {
    // Объявляем объект класса employee
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    // Производим попытку изменения несуществующего члена класса $var,
    // что вызывает генерацию исключения
    $obj->var = 100;
  }
  catch(Exception $exp)
  {
    // Объект $obj все еще доступен
    echo "<pre>";
    print_r($obj);
    echo "</pre>";
  }
?>
```

Результатом работы скрипта из листинга 24.24 будет следующий дамп объекта:

```
employee Object
(
    [surname:private] => Корнеев
    [name:private] => Иван
    [patronymic:private] => Григорьевич
    [age:private] => 18
)
```

24.2.6. Генерация исключений в иерархиях классов

На первый взгляд, генерация исключений в базовых и производных классах не имеет никаких особенностей: вне зависимости от того, сгенерировано исключение в базовом или производном классе, обработчик имеет возможность его перехватить. Однако при отладке иерархии класса зачастую важно установить, где было сгенерировано исключение — в базовом или производном классе, т. к. классы могут располагаться в разных файлах.

Рассмотрим базовый класс `base`, который содержит единственный открытый метод `generate()`, задачей которого является генерация исключения (листинг 24.25).

Листинг 24.25. Базовый класс `base`

```
<?php
  class base
  {
    public function generate()
```

```
    {
      throw new Exception("Исключение базового класса");
    }
  }
?>
```

Перехват исключения, генерируемого методом `generate()`, может выглядеть так, как это представлено в листинге 24.26.

Листинг 24.26. Перехват исключения метода `generate()` базового класса

```
<?php
  require_once("class.base.php");

  try
  {
    $obj = new base();
    $obj->generate();
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

Результатом работы скрипта из листинга 24.26 будет следующий стек передачи исключения:

```
#0 D:\main\oop\08\index.php(7): base->generate()
#1 {main}
```

Пусть от базового класса `base` (который расположен в файле class.base.php) наследует производный класс `derived` (который, в свою очередь, располагается в файле class.derived.php) (листинг 24.27).

Листинг 24.27. Производный класс `derived`

```
<?php
  require_once("class.base.php");

  class derived extends base
  {
  }
?>
```

Класс `derived` не реализует никаких собственных методов, однако он наследует метод `generate()` базового класса `base` (листинг 24.28).

Листинг 24.28. Перехват исключения производного класса

```
<?php
  require_once("class.base.php");
  require_once("class.derived.php");

  try
  {
    $obj = new derived();
    $obj->generate();
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp->getTraceAsString();
    echo "</pre>";
  }
?>
```

Однако в стеке передачи исключения будет фигурировать не производный класс `derived`, а базовый класс `base`:

```
#0 D:\main\oop\08\index.php(8): base->generate()
#1 {main}
```

Это связано с тем, что метод `generate()` принадлежит базовому классу `base`. Однако если метод `generate()` перегружается в производном классе `derived`, пусть даже с вызовом метода производного класса, в стеке будет фигурировать производный класс `derived` (листинг 24.29).

Листинг 24.29. Перегрузка метода `generate()` в производном классе `derived`

```
<?php
  require_once("class.base.php");

  class derived extends base
  {
    public function generate()
    {
      parent::generate();
    }
  }
?>
```

Теперь вызов скрипта из листинга 24.28 будет приводить к следующему стеку передачи исключения:

```
#0 D:\main\oop\08\class.derived.php(8): base->generate()
#1 D:\main\oop\08\index.php(8): derived->generate()
#2 {main}
```

24.2.7. Использование объекта класса *Exception* в строковом контексте

Как видно из табл. 24.6, класс `Exception` реализует метод `__toString()`, который позволяет вызывать объект в строковом контексте. В листинге 24.30 представлен переработанный скрипт из листинга 24.23, демонстрирующий обработку исключительной ситуации при попытке обратиться к несуществующему члену класса. При этом в обработчике выводится только переданный ему объект класса `Exception`.

Листинг 24.30. Использование объекта класса `Exception` в строковом контексте

```
<?php
  require_once("class.employee.php");

  try
  {
    $obj = new employee("Корнеев", "Иван", "Григорьевич");
    $obj->var = 100;
  }
  catch(Exception $exp)
  {
    echo "<pre>";
    echo $exp;
    echo "</pre>";
  }
?>
```

Результатом работы скрипта будет автоматически сгенерированный методом `__toString()` отчет, содержащий всю доступную об исключительной ситуации информацию:

```
exception 'Exception' with message 'Член var не существует' in
D:\main\oop\08\class.employee.php:25
Stack trace:
#0 D:\main\oop\08\index.php(7): employee->__set('var', 100)
#1 {main}
```

24.2.8. Создание собственных исключений

Класс `Exception` может выступать в качестве базового класса для пользовательских классов исключений. Создадим новый производный класс исключения `ExceptionSQL`, который бы сохранял всю информацию, полученную об исключительной ситуации, в MySQL-таблицу `exception`, содержащую следующие поля:

- `id_exception` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`;
- `message` — текстовое поле для сообщения, описывающего исключительную ситуацию;
- `code` — целочисленное поле для хранения кода, назначенного данному типу исключительных ситуаций;
- `file` — текстовое поле для имени файла, в котором произошла исключительная ситуация;
- `line` — целочисленное поле, хранящее номер строки, в которой произошла исключительная ситуация;
- `trace` — стек передачи исключительной ситуации, возвращаемый методом `getTraceAsString()`;
- `putdate` — календарный столбец для хранения даты и времени возникновения исключительной ситуации.

В листинге 24.31 представлен оператор `CREATE TABLE`, позволяющий создать таблицу `exception`.

Листинг 24.31. Таблица `exception` для хранения исключений

```
CREATE TABLE exception (
  id_exception INT(11) NOT NULL AUTO_INCREMENT,
  message TEXT NOT NULL,
  code INT(11) NOT NULL,
  file TINYTEXT NOT NULL,
  line INT(11) NOT NULL,
  trace TEXT NOT NULL,
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_exception)
);
```

Теперь можно унаследовать от класса `Exception` производный класс `ExceptionSQL`, который в конструкторе будет помещать в таблицу `exception` новую запись (листинг 24.32).

Листинг 24.32. Создание собственного типа исключений

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  class ExceptionSQL extends Exception
  {
    // Первичный ключ записи в таблице exception,
    // соответствующий текущей исключительной ситуации
    protected $id_exception;
    // Конструктор класса, инициализирующий члены
    // $message и $code - оба параметра реализованы как необязательные
    public function __construct($message = null, $code = 0)
    {
      // Вызываем конструктор базового класса
      parent::__construct($message, $code);

      // Экранируем спецсимволы и приводим
      // числовые значения к целому типу
      $message = mysql_real_escape_string($this->message);
      $file    = mysql_real_escape_string($this->file);
      $trace   = mysql_real_escape_string($this->getTraceAsString());
      $code    = intval($this->code);
      $line    = intval($this->line);

      // Формируем и выполняем SQL-запрос на добавление
      // информации об исключительной ситуации в базу данных
      $query = "INSERT INTO exception
                VALUES (NULL,
                       '$message',
                        $code,
                       '$file',
                        $line,
                       '$trace',
                        NOW())";
      @mysql_query($query);

      // Извлекаем значение первичного ключа id_exception,
      // назначенного новой записи при помощи AUTO_INCREMENT
      $this->id_exception = mysql_insert_id();
    }
```

```
    // Возвращаем значение первичного ключа в таблице exception
    public function getID()
    {
      return $this->id_exception;
    }
  }
?>
```

Впрочем, поле `trace` вряд ли будет заполнено корректно, т. к. полный путь передачи исключения формируется лишь в `catch`-обработчике, а при создании объекта (когда вызывается конструктор) он остается незаполненным.

Помимо конструктора, класс `ExceptionSQL` содержит метод `getID()`, который позволяет получить значение закрытого члена `id_exception`, содержащего первичный ключ таблицы `exception` с информацией о текущей исключительной ситуации. В листинге 24.33 приводится пример обработки исключения `ExceptionSQL`. Следует обратить внимание на то, что в `catch`-обработчике для перехвата исключения указывается исключение типа `ExceptionSQL`.

Листинг 24.33. Обработка исключения `ExceptionSQL`

```
<?php
  // Подключаем реализацию класса
  require_once("class.ExceptionSQL.php");

  try
  {
    if (rand(0,1)) throw new ExceptionSQL("Случайное исключение");
  }
  catch(ExceptionSQL $exp)
  {
    // Получаем первичный ключ таблицы exception, соответствующий исключению
    $id_exception = $exp->getID();
    // Извлекаем информацию об исключительной ситуации
    $query = "SELECT * FROM exception
              WHERE id_exception = $id_exception";
    $ext = mysql_query($query);
    if (!$ext) exit("Ошибка извлечения исключения");
    $exception = mysql_fetch_array($ext);
    echo "<table border=1 cellpadding=5>";
    echo "<tr>
           <td>Сообщение:</td>
           <td>$exception[message]</td>
          </tr>";
```

```
    echo "<tr>
            <td>Код:</td>
            <td>$exception[code]</td>
          </tr>";
    echo "<tr>
            <td>Файл:</td>
            <td>$exception[file]</td>
          </tr>";
    echo "<tr>
            <td>Строка:</td>
            <td>$exception[line]</td>
          </tr>";
    echo "<tr>
            <td>Стек:</td>
            <td><pre>{$exception[trace]}</pre></td>
          </tr>";
    echo "<tr>
            <td>Время:</td>
            <td>$exception[putdate]</td>
          </tr>";
  }
?>
```

Результат работы скрипта из листинга 24.23 при условии, что функция `rand()` возвращает значение 1, представлен на рис. 24.4.

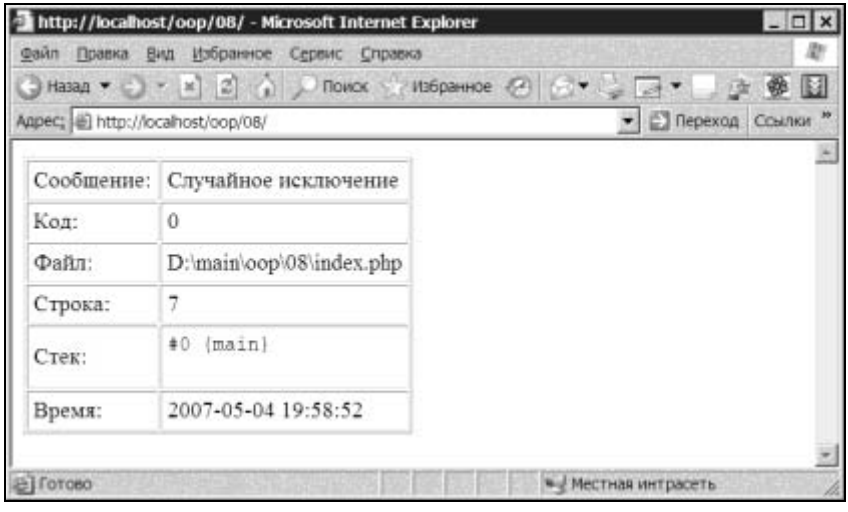


Рис. 24.4. Отчет скрипта из листинга 24.33 об исключительной ситуации

Помимо классической информации об исключительной ситуации (сообщение, код, имя файла, строка) появляется возможность получить точное время, когда произошло исключение. Можно модифицировать класс `ExceptionSQL` таким образом, чтобы сохранялась и другая ценная информация, например, IP-адрес посетителя (`$_SERVER['REMOTE_ADDR']`), адрес страницы, откуда посетитель пришел на данную страницу (`$_SERVER['HTTP_REFERER']`).

24.2.9. Создание новых типов исключений

Собственные классы исключений могут сами выступать в качестве базового класса для новых типов исключений. Например, от класса `ExceptionSQL` *(см. разд. 24.2.8)* можно унаследовать еще два класса:

- `ExceptionSQLNoneRecords` — исключительная ситуация, возникающая при отсутствии записей в таблице;
- `ExceptionSQLError` — исключительная ситуация, возникающая при возникновении ошибки обработки SQL-запроса.

Общая схема иерархии классов исключительных ситуаций представлена на рис. 24.5. Разумеется, в промышленных приложениях иерархия классов исключений может быть более сложной и разветвленной.

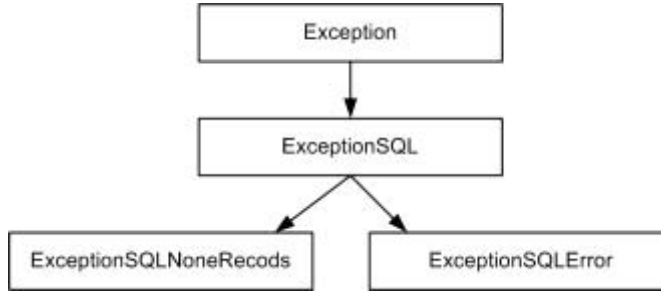


Рис. 24.5. Иерархия исключений

В листинге 24.34 представлена реализация класса `ExceptionSQLNoneRecords`. Класс перегружает конструктор, лишая его второго необязательного параметра `$code`, содержащего числовой код исключительной ситуации. Конструктору базового класса `ExceptionSQL` в качестве второго параметра передается константа 1001.

Листинг 24.34. Класс `ExceptionSQLNoneRecords`

```
<?php
  require_once("class.ExceptionSQL.php");

  class ExceptionSQLNoneRecords extends ExceptionSQL
  {
```

```
    public function __construct(
                          $message = "Запрашиваемые записи отсутствуют")
    {
      parent::__construct($message, 1001);
    }
  }
?>
```

В листинге 24.35 представлена реализация класса `ExceptionSQLError`, который, как и класс `ExceptionSQLNoneRecords`, перегружает конструктор, причем конструктор класса `ExceptionSQLError` не принимает ни одного параметра. Первый параметр базового класса `ExceptionSQL` формируется при помощи функции `mysql_error()`, а в качестве второго параметра ему передается константа 1002.

Листинг 24.35. Класс `ExceptionSQLError`

```
<?php
  require_once("class.ExceptionSQL.php");

  class ExceptionSQLError extends ExceptionSQL
  {
    public function __construct()
    {
      parent::__construct("Ошибка в SQL-запросе: ".mysql_error(), 1001);
    }
  }
?>
```

Для того чтобы обработать исключение, достаточно указать в `catch`-блоке его тип. Однако код в контролируемом блоке может генерировать большое количество разных типов исключений, для каждого их которых может потребоваться собственный `catch`-обработчик. В этом случае `catch`-обработчики располагаются друг под другом, как это продемонстрировано в листинге 24.36.

Листинг 24.36. Использование нескольких `catch`-обработчиков

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    $query = "SELECT * FROM exception";
    $ext = mysql_query($query);
```

```
    // Если произошла ошибка - генерируем исключение ExceptionSQLError()
    if (!$ext) throw new ExceptionSQLError();
    // Если в таблице exception отсутствуют записи - генерируем исключение
    if (!mysql_num_rows($ext)) throw new ExceptionSQLNoneRecords();
    while($exception = mysql_fetch_array($ext))
    {
      echo $exception['message']."<br>";
    }
  }
  catch(ExceptionSQLError $exp)
  {
    // Ошибка в SQL-запросе
    echo "Произошла ошибка при выполнении SQL-запроса<br>";
    echo "{$exp->getMessage()}<br>";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    // Отсутствуют записи
    echo "Таблица exception не содержит записей<br>";
  }
?>
```

Таким образом, для каждого из типов исключений можно предусмотреть собственный обработчик, количество которых не ограничено.

24.2.10. Перехват исключений производных классов

Исключения производных типов можно перехватывать при помощи исключений базового типа. В листинге 24.37 по случайному закону генерируется либо исключение `ExceptionSQLError`, либо исключение `ExceptionSQLNoneRecords`, однако для последнего отсутствует специализированный обработчик — его перехватывает обработчик `ExceptionSQL`.

Листинг 24.37. Перехват исключений производных классов

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if (rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
```

```
  }
  catch(ExceptionSQLError $exp)
  {
    // Перехватывается исключение ExceptionSQLError
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются исключения ExceptionSQL и ExceptionSQLNoneRecords
    echo "ExceptionSQL-исключение";
  }
?>
```

Если обработчик `ExceptionSQLError` отсутствует, оба типа исключений перехватываются обработчиком базового класса `ExceptionSQL` (листинг 24.38).

Листинг 24.38. Перехват исключений обработчиком базового класса

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if (rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются исключения ExceptionSQL, ExceptionSQLError и
    // ExceptionSQLNoneRecords
    echo "ExceptionSQL-исключение ".get_class($exp);
  }
?>
```

В листинге 24.38 тип исключения в `catch`-обработчике определяется при помощи функции `get_class()` *(см. разд. 21.8)*.

Важно помнить, что все обработчики производных классов должны быть расположены раньше обработчиков базовых классов, иначе исключения никогда не дойдут до специализированных обработчиков. В листинге 24.39 приводится скрипт, в котором никогда не срабатывает обработчик исключения `ExceptionSQLError`, т. к. исключение всегда перехватывается расположенным перед ним обработчиком исключений базового класса `ExceptionSQL`.

Листинг 24.39. Ошибочное расположение обработчиков

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if (rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются исключения ExceptionSQL, ExceptionSQLError и
    // ExceptionSQLNoneRecords
    echo "ExceptionSQL-исключение ".get_class($exp);
  }
  catch(ExceptionSQLError $exp)
  {
    // Этот блок никогда не будет исполнен, так как исключение
    // перехватывается предыдущим catch-обработчиком
    echo "ExceptionSQLError-исключение";
  }
?>
```

Замечание

Так как все исключения являются производными от базового класса `Exception`, то его использование позволяет перехватывать все исключения, не обработанные предыдущими обработчиками. Такая организация механизма исключительных ситуаций позволяет не вводить специальных ключевых слов вроде `__finally`, которые используются в Java.

24.2.11. Что происходит, когда исключения не перехватываются?

В сложных системах может сложиться ситуация, когда о некоторых возможных исключениях внешние разработчики, использующие класс, попросту не догадываются. В листинге 24.40 представлена простейшая ситуация генерации исключения без его обработки.

Листинг 24.40. Необработанное исключение

```
<?php
  throw new Exception();
?>
```

Результатом скрипта будут строки следующего вида:

```
Fatal error: Uncaught exception 'Exception' in D:\main\oop\08\index.php:2
Stack trace: #0 {main} thrown in D:\main\oop\08\index.php on line 2
```

Причем строки после `Fatal error: Uncaught` и до `thrown` формируются при помощи метода `__toString()`, в чем можно легко убедиться, перегрузив этот метод.

Листинг 24.41. Перегрузка метода `__toString()`

```
<?php
  class ExceptionSpecitial extends Exception
  {
    public function __toString()
    {
      return "Исключение ExceptionSpecitial оказалось не обработанным";
    }
  }

  throw new ExceptionSpecitial();
?>
```

Результатом работы скрипта из листинга 24.41 будут строки вида:

```
Fatal error: Uncaught Исключение ExceptionSpecitial оказалось не обработанным
thrown in D:\main\oop\08\index.php on line 10
```

Таким образом, изменить реакцию на необработанное исключение при помощи перегрузки метода `__toString()` не удастся, т. к. он предназначен исключительно для штатного использования. Тем более, разработчик класса вряд ли сможет предположить, какая реакция приложения на необработанное исключение будет удобна внешнему разработчику.

В связи с этим в PHP, начиная с версии 5.0, введена функция `set_exception_handler()`, которая позволяет переопределить обработчик для необработанных исключений. В качестве единственного параметра она принимает строку с именем функции обратного вызова, которое может быть произвольным, не совпадающим с именами уже существующих функций. Функция обратного вызова должна быть подготовлена внешним разработчиком и принимать единственный параметр — объект исключения. В листинге 24.42 приводится пример переопределения обработчика необработанных исключений.

Листинг 24.42. Использование функции `set_exception_handler()`

```
<?php
  function exception_handler($exp)
  {
    echo "Не обработано сообщение ".get_class($exp);
```

```
  }

  set_exception_handler("exception_handler");

  throw new Exception();

  echo "Это строка никогда не будет выведена";
?>
```

Результатом работы скрипта из листинга 24.42 будет предупреждение: "Не обработано сообщение Exception". Важно отметить, что при генерации исключения интерпретатор PHP покидает исполняемый модуль, и оставшаяся часть кода не выполняется.

24.2.12. Вложенные контролируемые блоки

Контролируемые блоки могут быть вложены друг в друга. Например, листинг 24.37 может быть переписан следующим образом (листинг 24.43).

Листинг 24.43. Вложенные контролируемые блоки

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    try
    {
      if (rand(0,1)) throw new ExceptionSQLError();
      else throw new ExceptionSQLNoneRecords();
    }
    catch(ExceptionSQLError $exp)
    {
      // Перехватывается исключение ExceptionSQLError
      echo "ExceptionSQLError-исключение";
    }
  }
  catch(ExceptionSQL $exp)
  {
    // Перехватываются исключения ExceptionSQL и ExceptionSQLNoneRecords
    echo "ExceptionSQL-исключение";
  }
?>
```

Исключения, которые не перехватываются внутренними контролирующими блоками, передаются во внешние контролирующие блоки. Если исключение не было обработано ни одним из обработчиков, то оно считается необработанным и передается функции обратного вызова, назначаемой функцией `set_exception_handler()` *(см. разд. 24.2.11)*.

Разумеется, вложенные контролируемые блоки редко используются в контексте, как это сделано в листинге 24.43, где более уместным было бы определить один контролируемый блок с несколькими `catch`-обработчиками. Вложенные контролируемые блоки используются в ситуациях, когда класс или функция часть исключений обрабатывают сами, а часть передают внешнему коду.

Пусть имеется файл generator.php, который по случайному закону генерирует одно из трех исключений: `ExceptionSQL`, `ExceptionSQLError` и `ExceptionSQLNoneRecords` (листинг 24.44).

Листинг 24.44. Файл generator.php

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  switch(rand(0,2))
  {
    case 0:
      throw new ExceptionSQL();
    case 1:
      throw new ExceptionSQLError();
    case 2:
      throw new ExceptionSQLNoneRecords();
  }
?>
```

Как видно из листинга 24.44, в `case`-блоках оператора `switch` не используются традиционные операторы `break`, позволяющие досрочно выйти из `switch`. Генерация исключения приводит к тому, что PHP-интерпретатор прекращает выполнение оставшейся части кода и покидает файл generator.php.

В листинге 24.45 представлена функция `catch_exception()`, которая обрабатывает два вида исключений: `ExceptionSQLError` и `ExceptionSQLNoneRecords`.

> ***ЗАМЕЧАНИЕ***
>
> Вместо отдельной функции `catch_exception()` может выступать метод класса.

Листинг 24.45. Функция обработки исключений

```
<?php
  function catch_exception()
  {
```

```
    try
    {
      require_once("generate.php");
    }
    catch(ExceptionSQLError $exp)
    {
      echo "ExceptionSQLError-исключение";
    }
    catch(ExceptionSQLNoneRecords $exp)
    {
      echo "ExceptionSQLNoneRecords-исключение";
    }
  }
?>
```

Следует отметить, что, если файл generate.php генерирует исключение `ExceptionSQL()`, функция `catch_exception()` не обрабатывает его самостоятельно, а передает во внешний код, который обладает такой возможностью (листинг 24.46).

Листинг 24.46. Перехват исключения `ExceptionSQL`

```
<?php
  require_once("function.catch_exception.php");

  try
  {
    catch_exception();
  }
  catch(ExceptionSQL $exp)
  {
    echo "ExceptionSQL-исключение";
  }
?>
```

Таким образом, можно организовывать каскады обработки исключений, вкладывая контролируемые блоки друг в друга. Разумеется, обработчики исключений производных классов должны располагаться на более глубоком уровне, чем обработчики исключений базовых классов.

24.2.13. Повторная генерация исключений

Исключение, перехваченное одним `catch`-обработчиком, может быть регенерировано для передачи его следующему по каскаду обработчику. Это позволяет нескольким обработчикам получить доступ к одному и тому же исключению. Для повторной генера-

ции исключения достаточно вызвать оператор `throw` в `catch`-блоке, передав ему значение объекта исключения.

В листинге 24.47 обработчик исключений базового класса `ExceptionSQL` предшествует обработчикам исключений производных типов `ExceptionSQLError` и `ExceptionSQLNoneRecords`.

Листинг 24.47. Повторная генерация исключений

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    try
    {
      if (rand(0,1)) throw new ExceptionSQLError();
      else throw new ExceptionSQLNoneRecords();
    }
    catch(ExceptionSQL $exp)
    {
      echo "ExceptionSQL-исключение ".get_class($exp)."<br>";
      // Передача исключения далее по каскаду
      throw $exp;
    }
  }
  catch(ExceptionSQLError $exp)
  {
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    echo "ExceptionNoneRecords-исключение";
  }
?>
```

В зависимости от сгенерированного исключения, скрипт из листинга 24.47 возвратит либо эти строки:

```
ExceptionSQL-исключение ExceptionSQLError
ExceptionSQLError-исключение
```

либо такие:

```
ExceptionSQL-исключение ExceptionSQLNoneRecords
ExceptionNoneRecords-исключение
```

Таким образом, какое бы из исключений производного класса не было бы сгенерировано, `catch`-обработчик базового класса в любом случае будет выполнен.

Важно отметить, что передать заново сгенерированное исключение можно только внешнему контролирующему блоку. Повторная генерация исключения в рамках одного контролирующего блока с несколькими `catch`-обработчиками не вызовет перехода к следующим `catch`-обработчикам. В листинг 24.48 повторно сгенерированное исключение не будет обработано ни одним из обработчиков.

Листинг 24.48. Повторно сгенерированное исключение останется необработанным

```
<?php
  require_once("class.ExceptionSQLError.php");
  require_once("class.ExceptionSQLNoneRecords.php");

  try
  {
    if (rand(0,1)) throw new ExceptionSQLError();
    else throw new ExceptionSQLNoneRecords();
  }
  catch(ExceptionSQL $exp)
  {
    echo "ExceptionSQL-исключение ".get_class($exp)."<br>";
    // Передача исключения далее по каскаду
    throw $exp;
  }
  catch(ExceptionSQLError $exp)
  {
    echo "ExceptionSQLError-исключение";
  }
  catch(ExceptionSQLNoneRecords $exp)
  {
    echo "ExceptionNoneRecords-исключение";
  }
?>
```

Результат работы скрипта из листинга 24.48 может выглядеть следующим образом:

```
ExceptionSQL-исключение ExceptionSQLError
Fatal error: Uncaught exception 'ExceptionSQLError' with message 'Ошибка в
SQL-запросе: ' in D:\main\oop\08\index.php:7 Stack trace: #0 {main} thrown in
D:\main\oop\08\index.php on line 7
```



Глава 25

Отражения

Механизм отражений предоставляет разработчику возможность исследования как пользовательских, так предопределенных классов, выясняя статус и состав отдельных членов, методов, классов и даже расширений PHP, а также объявлять объекты классов и выполнять над ними манипуляции. Кроме этого отражения позволяют автоматически генерировать документацию иерархии классов по схеме javadoc, применяемой в технологии Java.

25.1. Иерархия классов отражения

Механизм отражений включает иерархию классов, каждый из которых (кроме класса `Reflection`) реализует интерфейс `Reflector`. Интерфейс `Reflector` не требует реализации никаких методов и предназначен лишь для объединения классов отражения в одну группу таким образом, чтобы проверка объекта при помощи оператора `instanceof` на принадлежность интерфейсу `Reflector` возвращала значение `true`.

> ***Замечание***
>
> Класс механизма отражения относится к так называемым предопределенным классам, т. е. классам, которые реализованы в PHP-интерпретаторе и не требуют реализации со стороны программиста. Другие предопределенные классы более подробно рассматриваются в *главе 26*.

В табл. 25.1 представлен список классов механизма отражения и их краткое описание.

Таблица 25.1. *Список классов механизма отражения*

Класс	Описание
`Reflection`	Класс содержит вспомогательные статические методы для обслуживания объектов классов механизма отражения
`ReflectionException`	Класс для обработки исключительных ситуаций *(см. главу 24)*, возникающих при работе с отражениями
`ReflectionFunction`	Класс предназначен для представления (отражения) функции
`ReflectionParameter`	Класс предназначен для представления (отражения) параметра функции

Таблица 25.1 *(окончание)*

Класс	Описание
`ReflectionMethod`	Класс предназначен для представления (отражения) метода класса
`ReflectionClass`	Класс предназначен для представления (отражения) класса
`ReflectionObject`	Класс предназначен для представления (отражения) объекта
`ReflectionProperty`	Класс предназначен для представления (отражения) члена класса
`ReflectionExtension`	Класс предназначен для представления (отражения) расширения PHP

Некоторые классы иерархии являются базовыми, а некоторые — производными. Так, класс `ReflectionException` является производным класса `Exception`, класс `ReflectionMethod` наследует от `ReflectionFunction`, а `ReflectionObject` от `ReflectionClass` (рис. 25.1).

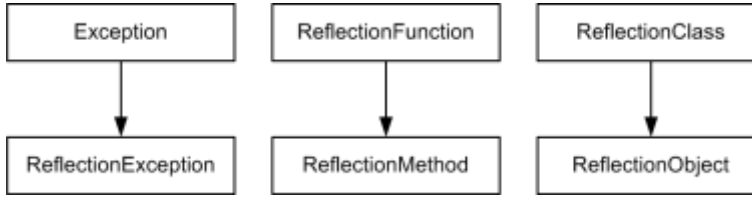


Рис. 25.1. Отношение наследования среди классов механизма отражения

25.2. Отражение функции. Класс *ReflectionFunction*

Объект класса `ReflectionFunction` содержит информацию о функции, имя которой задается в качестве параметра конструктора. В результате методы объекта могут вернуть о функции самую разнообразную информацию, начиная именем, заканчивая количеством и типом параметров, которые функция может принимать.

В листинге 25.1 представлена демонстрационная функция `hello()`, принимающая единственный параметр `$name`, который подставляется в приветственную строку, выводимую функцией.

Листинг 25.1. Демонстрационная функция `hello()`

```
<?php
  function hello($name)
  {
```

```
    echo "Hello, $name!";
  }
?>
```

Для того чтобы объявить отражение функции `hello()`, достаточно создать объект класса `ReflectionFunction`, инициированный строкой `"hello"` (листинг 25.2).

Листинг 25.2. Создание отражения функции `hello()`

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Выводим название функции
  echo "{$obj->getName()}<br>";
  // Выполняем функцию, передавая в качестве аргумента строку "world"
  $obj->invoke("world");
?>
```

Скрипт из листинга 25.2 создает объект отражения `$obj` для функции `hello()`. При помощи полученного объекта выводится название функции, а также производится ее выполнение. Результатом выполнения скрипта из листинга 9.2 будут следующие строки:

```
hello
Hello, world!
```

В табл. 25.2 представлен полный список методов класса `ReflectionFunction`, позволяющих получать самую разнообразную информацию о функции.

***Таблица 25.2.** Список методов класса `ReflectionFunction`*

Метод	Описание
`object __construct($name)`	Конструктор класса, принимающий в качестве единственного параметра `$name` имя функции
`string __toString()`	Вывод информации при использовании объекта в строковом контексте
`static string export($name, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о функции `$name` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `_toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionFunction::export('hello')`

***Таблица 25.2** (окончание)*

Метод	Описание
`string getName()`	Возвращает название метода
`bool isInternal()`	Возвращает `TRUE`, если функция является предопределенной (входит в ядро PHP), и `FALSE`, если функция является пользовательской
`bool isUserDefined()`	Возвращает `TRUE`, если функция является пользовательской, и `FALSE`, если функция является предопределенной (входит в ядро PHP)
`string getFileName()`	Возвращает полный путь к файлу, в котором определена функция
`int getStartLine()`	Возвращает номер строки в файле, соответствующий началу функции
`int getEndLine()`	Возвращает номер строки в файле, соответствующий концу функции
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед функцией
`array getStaticVariables()`	Возвращает ассоциативный массив всех статических переменных функции и соответствующие им значения (на текущий момент). Имена переменных служат ключами массива, а в качестве значений элементов выступают значения переменных
`mixed invoke($arg1, $arg2 ...)`	Выполняет функцию, передавая ей в качестве аргументов параметры `$arg1`, `$arg2` и т. д.
`mixed invokeArgs($arr)`	Выполняет функцию, однако в отличие от метода `invoke()` параметры передаются в виде массива `$arr`
`bool returnsReference()`	Возвращает `TRUE`, если функция возвращает результат по ссылке, и `FALSE` в противном случае
`ReflectionParameter[] getParameters()`	Возвращает массив объектов класса `ReflectionParameter` *(см. разд. 9.3)*, представляющих собой отражение параметров функции
`int getNumberOfParameters()`	Возвращает количество параметров функции
`int getNumberOfRequired-Parameters()`	Возвращает количество обязательных параметров функции

Класс `ReflectionFunction` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 25.3.

Листинг 25.3. Гипотетическая реализация класса `ReflectionFunction`

```
<?php
  class ReflectionFunction implements Reflector
  {
    final private function __clone();
    public function __construct($name);
    public function __toString();
    public function static export();
    public function getName();
    public function isInternal();
    public function isUserDefined();
    public function getFileName();
    public function getStartLine();
    public function getEndLine();
    public function getDocComment();
    public function getStaticVariables();
    public function invoke($args, ...);
    public function invokeArgs($arr);
    public function returnsReference();
    public function getParameters();
    public function getNumberOfParameters();
    public function getNumberOfRequiredParameters();
  }
?>
```

Одним из самых интересных методов является метод `getDocComment()`, который позволяет восстанавливать многострочный комментарий перед функцией, предоставляя разработчикам аналог javadoc из технологии Java для построения автоматической документации к системе. Модифицируем файл function.hello.php с функцией `hello()` так, как это представлено в листинге 25.4.

> ***Замечание***
>
> Важным условием распознавания комментария методом `getDocComment()` является начало комментария с последовательности `/**`.

Листинг 25.4. Многострочный комментарий перед функцией `hello()`

```
<?php
  /** Метод hello() принимает единственный параметр
      $name и выводит в окно браузера приветствие
      "Hello, $name!"
    */
```

```
  function hello($name)
  {
     echo "Hello, $name!";
  }
?>
```

Теперь обращаясь к методу `getDocComment()` объекта отражения функции `hello()`, можно получить содержимое комментария, предваряющего функцию в файле function.hello.php (листинг 25.5).

Листинг 25.5. Использование метода `getDocComment()` класса `ReflectionFunction`

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Выводим комментарий к функции
  echo "<pre>";
  echo $obj->getDocComment();
  echo "</pre>";
  // А также название файла и строки,
  // с которой функция начинается и которой заканчивается
  echo "Файл: {$obj->getFileName()}<br>";
  echo "Строки с {$obj->getStartLine()}
        по {$obj->getEndLine()}<br>";
?>
```

Результат выполнения скрипта из листинга 25.5 представлен на рис. 25.2.

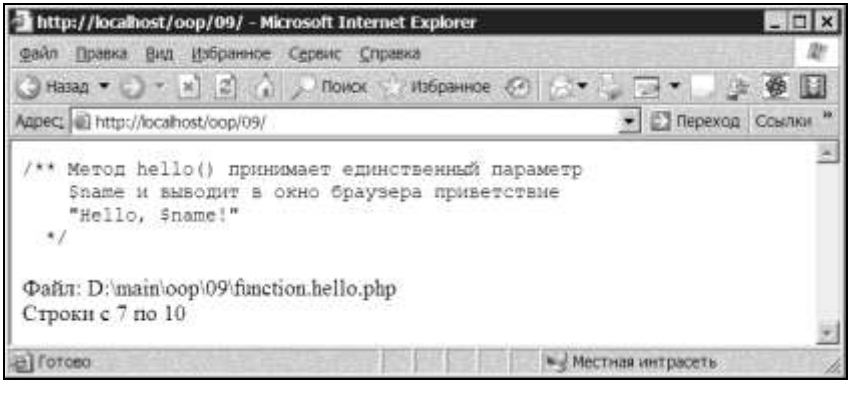


Рис. 25.2. Результат выполнения скрипта из листинга 25.5

25.3. Отражение параметра функции. Класс *ReflectionParameter*

Объект класса `ReflectionParameter` содержит информацию о параметре функции, имя которого задается в качестве параметра конструктора. В результате методы объекта могут вернуть о параметре разнообразную информацию. Полный список методов класса представлен в табл. 25.3.

Таблица 25.3. *Методы класса* `ReflectionParameter`

Метод	Описание
`object __construct($function, $parameter)`	Конструктор класса, создающий отражение для параметра `$paramter` функции `$function`. Можно задавать как название параметра `$parameter`, так и его порядковый номер (нумерация начинается с 0)
`string __toString()`	Вывод информации, при использовании объекта в строковом контексте
`static string export($function, $parameter, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о параметре `$parameter` функции `$function` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionParameter::export('hello', 0)`
`string getName()`	Возвращает название параметра
`ReflectionFunction getDeclaringFunction()`	Возвращает объект класса `ReflectionFunction` для функции, которой принадлежит текущий параметр
`ReflectionClass getDeclaringClass()`	Возвращает объект класса `ReflectionClass` *(см. разд. 25.4)*, которому принадлежит метод, которому в свою очередь принадлежит текущий параметр
`ReflectionClass getClass()`	Синоним метода `getDeclaringClass()`
`bool isArray()`	Возвращает `TRUE`, если параметр является массивом, и `FALSE` в противном случае
`bool allowsNull()`	Возвращает `TRUE`, если параметр может принимать значение `NULL` или является необязательным, и `FALSE` в противном случае
`bool isPassedByReference()`	Возвращает значение `TRUE`, если параметр передается по ссылке, и `FALSE` в противном случае
`bool isOptional()`	Возвращает `TRUE`, если параметр является необязательным, и `FALSE`, в противном случае

***Таблица 25.3** (окончание)*

Метод	Описание
`bool isDefaultValueAvailable()`	Возвращает `TRUE`, если параметр имеет значение по умолчанию, и `FALSE` в противном случае
`mixed getDefaultValue()`	Возвращает значение по умолчанию, если параметр его имеет

Класс `ReflectionParameter` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 25.6.

Листинг 25.6. Гипотетическая реализация класса `ReflectionParameter`

```
<?php
class ReflectionParameter implements Reflector
{
    final private function __clone();
    public function __construct($function, $parameter);
    public function __toString();
    public static function export($function, $parameter, $return);
    public function getName();
    public function getDeclaringFunction();
    public function getDeclaringClass();
    public function getClass();
    public function isArray();
    public function allowsNull();
    public function isPassedByReference();
    public function isOptional();
    public function isDefaultValueAvailable();
    public function getDefaultValue();
}
?>
```

В листинге 25.7 приводится пример создания отражения для единственного параметра функции `hello()` (см. листинг 25.4).

Листинг 25.7. Создание отражения для параметра функции `hello()`

```
<?php
  require_once("function.hello.php");

  $obj = new ReflectionParameter("hello", 0);
```

```
  // Выводим название параметра
  echo $obj->getName();
?>
```

Однако способ создания отражения параметра, представленный в листинге 25.7, практически не используется. В подавляющем большинстве случаев речь идет о массиве объектов класса `ReflectionParameter`, который возвращается методом `getParameters()` объекта класса `ReflectionFunction`. Такой массив, как правило, обходится при помощи цикла (листинг 25.8).

Листинг 25.8. Получение и обработка массива отражений параметров функции

```
<?php
  require_once("function.hello.php");

  // Получаем отражение функции
  $obj = new ReflectionFunction("hello");
  // Получаем массив объектов отражений параметров функции
  $arr = $obj->getParameters();
  // В цикле обрабатываем отражения параметров
  foreach($arr as $par)
  {
    // Выводим название очередного параметра
    echo $par->getName()."<br>";
  }
?>
```

25.4. Отражение класса. Класс *ReflectionClass*

Объект класса `ReflectionClass` содержит информацию о классе, имя которого задается в качестве параметра конструктора. В результате методы объекта могут вернуть о классе разнообразную информацию, кроме того ряд специальных методов позволяет получить доступ к массиву отражений методов и членов класса. Полный список методов класса представлен в табл. 25.4.

***Таблица 25.4.** Методы класса `ReflectionClass`*

Метод	Описание
`object __construct($name)`	Конструктор класса, позволяющий создавать отражение для класса с именем `$name`
`string __toString()`	Вывод информации при использовании объекта в строковом контексте

***Таблица 25.4** (продолжение)*

Метод	Описание
`static string export($class, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о классе `$class` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionClass::export('class_name')`
`string getName()`	Возвращает имя класса
`bool isInternal()`	Возвращает `TRUE`, если класс является предопределенным (входит в ядро PHP), и `FALSE`, если класс является пользовательским
`bool isUserDefined()`	Возвращает `TRUE`, если класс является пользовательским, и `FALSE`, если класс является предопределенным (входит в ядро PHP)
`bool isInstantiable()`	Возвращает `TRUE`, если класс не является абстрактным (возможно создание объекта данного класса), и `FALSE` в противном случае
`bool isAbstract()`	Возвращает `TRUE`, если класс является абстрактным (невозможно создание объекта данного класса), и `FALSE` в противном случае
`bool isInterface()`	Возвращает `TRUE`, если класс является интерфейсом
`bool isFinal()`	Возвращает `TRUE`, если класс не допускает наследования (создание производных классов), и `FALSE` в противном случае
`bool isIterateable()`	Возвращает `TRUE`, если класс может быть использован в качестве массива в цикле `foreach`, и `FALSE` в противном случае возвращается
`bool isInstance($object)`	Возвращает `TRUE`, если объект `$object` является экземпляром класса отражения, и `FALSE` в противном случае. Поведение данного метода совпадает с поведением оператора `instanceof`
`bool isSubclassOf($class)`	Возвращает `TRUE`, если класс с именем `$class` является базовым для текущего класса, и `FALSE` в противном случае
`bool hasConstant($name)`	Возвращает `TRUE`, если класс содержит константу с именем `$name`, и `FALSE` в противном случае
`bool hasMethod($name)`	Возвращает `TRUE`, если класс содержит метод с именем `$name`, и `FALSE` в противном случае
`bool hasProperty($name)`	Возвращает `TRUE`, если класс содержит член с именем `$name`, и `FALSE` в противном случае

***Таблица 25.4** (продолжение)*

Метод	Описание
`bool implementsInterface($name)`	Возвращает `TRUE`, если класс реализует интерфейс с именем `$name`, и `FALSE` в противном случае
`string getFileName()`	Возвращает полный путь к файлу, в котором определен класс
`int getStartLine()`	Возвращает номер строки в файле, соответствующий началу класса
`int getEndLine()`	Возвращает номер строки в файле, соответствующий концу класса
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед классом
`ReflectionMethod getConstructor()`	Возвращает отражение (объект класса `ReflectionMethod`) для конструктора класса
`ReflectionMethod getMethod($name)`	Возвращает отражение (объект класса `ReflectionMethod`) для метода с именем `$name`
`ReflectionMethod[] getMethods()`	Возвращает массив отражений для методов класса
`ReflectionProperty getProperty($name)`	Возвращает отражение (объект класса `ReflectionProperty`) для члена класса с именем `$name`
`ReflectionProperty[] getProperties()`	Возвращает массив отражений для членов класса
`array getConstants()`	Возвращает ассоциативный массив констант класса, в качестве ключей выступают названия констант, а в качестве значений элементов — значения констант
`mixed getConstant($name)`	Возвращает значение константы с именем `$name`
`ReflectionClass[] getInterfaces()`	Возвращает массив отражений для интерфейсов, которые реализует класс
`int getModifiers()`	Возвращает битовую маску с установленными битами, соответствующими разным атрибутам класса, например, 00100000 (32) — абстрактный (`abstract`) класс, 01000000 (64) — константный (`final`) класс, 10010000 (144) — интерфейс (`interface`)
`stdclass newInstance($arg1, $arg2, ...)`	Создает объект класса, используя в качестве параметров конструктора аргументы `$arg1`, `$arg2` и т. д.
`stdclass newInstanceArgs($arr)`	Создает объект класса, используя в качестве параметров конструктора элементы массива `$arr`
`ReflectionClass getParentClass()`	Возвращает отражение (объект класса `ReflectionClass`) базового класса

***Таблица 25.4** (окончание)*

Метод	Описание
`array getStaticProperties()`	Возвращает ассоциативный массив статических членов класса, в качестве ключей выступают названия членов, а в качестве значений элементов — значения статических членов
`mixed getStaticPropertyValue ($name [, $default])`	Возвращает значение статической переменной с именем `$name`
`void setStaticPropertyValue ($name, $value)`	Позволяет установить новое значение `$value` для статической переменной `$name`
`array getDefaultProperties()`	Возвращает значения членов класса по умолчанию
`ReflectionExtension getExtension()`	Возвращает отражение расширения PHP (объект класса `ReflectionExtension`), которому принадлежит текущий класс
`string getExtensionName()`	Возвращает имя расширения PHP, которому принадлежит текущий класс

Класс `ReflectionClass` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 25.9.

Листинг 25.9. Гипотетическая реализация класса `ReflectionClass`

```
<?php
  class ReflectionClass implements Reflector
  {
    final private function __clone();
    public function __construct($name);
    public function __toString();
    public static function export($class, $return);
    public function getName();
    public function isInternal();
    public function isUserDefined();
    public function isInstantiable();
    public function isAbstract();
    public function isInterface();
    public function isFinal();
    public function isInstance($object);
    public function isSubclassOf($class);
    public function hasConstant($name);
```

```
    public function hasMethod($name);
    public function hasProperty($name);
    public function getFileName();
    public function getStartLine();
    public function getEndLine();
    public function getDocComment();
    public function getConstructor();
    public function getMethod($name);
    public function getMethods();
    public function getProperty($name);
    public function getProperties();
    public function getConstants();
    public function getConstant($name);
    public function getInterfaces();
    public function getModifiers();
    public function newInstance($args1, ...);
    public function newInstanceArgs($arr);
    public function getParentClass();
    public function getStaticProperties();
    public function getStaticPropertyValue($name [, $default]);
    public function setStaticPropertyValue($name, $value);
    public function getDefaultProperties();
    public function isIterateable();
    public function implementsInterface($name);
    public function getExtension();
    public function getExtensionName();
  }
?>
```

Для демонстрации приемов работы с отражением класса создадим демонстрационный класс `example` (листинг 25.10).

Листинг 25.10. Демонстрационный класс `example`

```
<?php
  /* Тестовый класс, для демонстрации возможностей отражения класса. */
  class example
  {
    const NAME = "cls";

    private $first;
    public $second;
    private $third;
```

```
    static $count = 0;

    public function __construct($first, $second, $third)
    {
      $this->first  = $first;
      $this->second = $second;
      $this->third  = $third;
    }

    public function getFirst()
    {
      return $this->first;
    }

    public function helloWorld()
    {
      return "Hello, world";
    }
  }
?>
```

В листинге 25.11 приводится пример получения отражения класса `example`, при помощи метода `getConstructor()` которого получается отображение конструктора, метод `getParameters()` которого предоставляет массив отображений параметров. В результате скрипт в цикле выводит список параметров конструктора.

Листинг 25.11. Получение списка параметров конструктора

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем массив отображений параметров конструктора
  $arr = $obj->getConstructor()->getParameters();

  foreach($arr as $par)
  {
    echo $par->getName()."<br>";
  }
?>
```

Листинг 25.12 предоставляет сведения о файле, в котором класс определен, о его начальной и конечной строках.

Замечание

Многострочные комментарии, предваряющие методы класса, также можно получить при помощи соответствующего представления `ReflectionMethod`.

Листинг 25.12. Получение сведений о локализации класса

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Выводим комментарий к классу
  echo "<pre>";
  echo $obj->getDocComment();
  echo "</pre>";
  // А также название файла и строки,
  // с которой класс начинается и которой заканчивается
  echo "Файл: {$obj->getFileName()}<br>";
  echo "Строки с {$obj->getStartLine()}
        по {$obj->getEndLine()}<br>";
?>
```

При помощи метода `newInstance()` допускается неявное создание объекта класса. В листинге 25.13 демонстрируется создание объекта класса `example` и вызов его метода `getFirst()`.

Листинг 25.13. Неявное создание объекта класса `example`

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Неявное создание объекта класса example
  $o = $obj->newInstance(1, 2, 3);

  // Вызов метода объекта класса example
  echo $o->getFirst();
?>
```

При помощи набора методов `getStaticProperties()`, `setStaticPropertyValue()` и `getStaticPropertyValue()` можно управлять статическими членами класса (листинг 25.14).

Листинг 25.14. Управление статическим членом `count` класса `example`

```
<?php
  require_once("class.example.php");

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Выводим список статических членов класса
  echo "<pre>";
  print_r($obj->getStaticProperties());
  echo "</pre>";

  // Устанавливаем новое значение статическому члену класса count
  $obj->setStaticPropertyValue("count", 100);

  // Выводим значение статического члена класса count
  echo $obj->getStaticPropertyValue("count");

?>
```

25.5. Отражение объекта. Класс *ReflectionObject*

Класс `ReflectionClass`, рассмотренный в предыдущем разделе, позволяет создать отражение класса, однако в иерархии классов механизма отражения предусмотрен дополнительный класс `ReflectionObject`, позволяющий создать отражение объекта. Класс `ReflectionObject` является производным класса `ReflectionClass` и переопределят лишь конструктор и статический метод `export()`. Если бы класс `ReflectionObject` не был предопределенным, то его определение могло бы выглядеть так, как это представлено в листинге 25.15.

Листинг 25.15. Гипотетическая реализация класса `ReflectionObject`

```
<?php
  class ReflectionObject extends ReflectionClass
  {
    public function __construct($object);
    public static function export($object, $return);
  }
?>
```

Остальные методы класс `ReflectionObject` наследует от класса `ReflectionClass` (см. табл. 25.4).

25.6. Отражение метода класса. Класс *ReflectionMethod*

Класс `ReflectionMethod` позволяет создать отражение метода класса. Сам класс `ReflectionMethod` наследует от базового класса `ReflectionFunction` *(см. разд. 25.2)* большинство методов. Если бы класс `ReflectionMethod` не был предопределенным, то его определение могло бы выглядеть так, как это представлено в листинге 25.16.

Листинг 25.16. Гипотетическая реализация класса `ReflectionMethod`

```
<?php
  class ReflectionMethod extends ReflectionFunction
  {
    public function __construct($class, $name);
    public function __toString();
    public static function export($class, $name, $return);
    public function invoke($object, $args);
    public function invokeArgs($object, $arr);
    public function isFinal();
    public function isAbstract();
    public function isPublic();
    public function isPrivate();
    public function isProtected();
    public function isStatic();
    public function isConstructor();
    public function isDestructor();
    public function getModifiers();
    public function getDeclaringClass();
  }
?>
```

В табл. 25.5 представлено описание методов класса `ReflectionMethod`, которые класс реализует помимо наследуемых от класса `ReflectionFunction`.

***Таблица 25.5.** Методы класса `ReflectionMethod`*

Метод	Описание
`__construct($class, $name)`	Конструктор класса, создающий отражение метода `$name` класса или объекта `$class`
`string __toString()`	Вывод информации при использовании объекта в строковом контексте

Таблица 25.5 *(окончание)*

Метод	Описание
`static string export($class, $name, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о методе `$name` класса `$class` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionMethod::export('example', 'getFirst')`
`mixed invoke($object, $arg1, $arg2, ...)`	Выполняет метод объекта `$object` с параметрами `$arg1`, `$arg2` и т. д.
`mixed invokeArgs($object, $arr)`	Выполняет метод объекта `$object` с параметрами, которые задаются в виде массива `$arr`
`bool isFinal()`	Возвращает `TRUE`, если метод снабжен атрибутом `final`, и `FALSE` в противном случае
`bool isAbstract()`	Возвращает `TRUE`, если метод является абстрактным (снабжен атрибутом abstract), и `FALSE` в противном случае
`bool isPublic()`	Возвращает `TRUE`, если метод является открытым (снабжен атрибутом `public`), и `FALSE` в противном случае
`bool isPrivate()`	Возвращает `TRUE`, если метод является закрытым (снабжен атрибутом `private`), и `FALSE` в противном случае
`bool isProtected()`	Возвращает `TRUE`, если метод является защищенным (снабжен атрибутом `protected`), и `FALSE` в противном случае
`bool isStatic()`	Возвращает `TRUE`, если метод является статическим (снабжен атрибутом `static`), и `FALSE` в противном случае
`bool isConstructor()`	Возвращает `TRUE`, если метод является конструктором, и `FALSE` в противном случае
`bool isDestructor()`	Возвращает `TRUE`, если метод является деструктором, и `FALSE` в противном случае
`int getModifiers()`	Возвращает битовую маску с установленными битами, соответствующими разным атрибутам метода (`public`, `private`, `protected`, `final`, `static`)
`ReflectionClass getDeclaringClass()`	Возвращает отражение класса (объект класса `ReflectionClass`), которому принадлежит метод

В листинге 25.17 приводится пример создания отражения метода `helloWorld()` класса `example` (листинг 25.10) и его неявное выполнение.

Листинг 25.17. Создание отражения метода `helloWorld()` класса `example`

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);
  // Получаем отражение метода
  $obj = new ReflectionMethod($exp, "helloWorld");
  // Осуществляем неявное выполнение метода
  echo $obj->invoke($exp);
?>
```

Несмотря на то, что класс `ReflectionMethod` допускает создание отражения при помощи конструктора, чаще создают отражение метода при помощи отражения класса, используя либо метод `getMethod()`, либо метод `getMethods()` (табл. 25.4). В листинге 25.18 демонстрируется вызов метод `helloWorld()` класса `example`.

Листинг 25.18. Получение отражения метода, через отражение класса

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Осуществляем неявный вызов метода
  echo $obj->getMethod("helloWorld")->invoke($exp);
?>
```

25.7. Отражение члена класса. Класс *ReflectionProperty*

Класс `ReflectionProperty` позволяет создать отражение члена класса. Методы класса `ReflectionProperty` представлены в табл. 25.6.

Класс `ReflectionProperty` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 25.19.

Таблица 25.6. *Методы класса* `ReflectionProperty`

Метод	Описание
`__construct($class, $name)`	Конструктор класса, создающий отражение члена `$name` класса или объекта `$class`
`string __toString()`	Вывод информации, при использовании объекта в строковом контексте
`static string export($class, $name, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о члене `$name` класса `$class` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionProperty::export('example', 'first')`
`string getName()`	Возвращает имя члена
`bool isPublic()`	Возвращает `TRUE`, если член является открытым (снабжен атрибутом `public`), и `FALSE` в противном случае
`bool isPrivate()`	Возвращает `TRUE`, если член является закрытым (снабжен атрибутом `private`), и `FALSE` в противном случае
`bool isProtected()`	Возвращает `TRUE`, если член является защищенным (снабжен атрибутом `protected`), и `FALSE` в противном случае
`bool isStatic()`	Возвращает `TRUE`, если член является статическим (снабжен атрибутом `static`), и `FALSE` в противном случае
`bool isDefault()`	Возвращает `TRUE`, если член имеет значение по умолчанию, и `FALSE` в противном случае
`int getModifiers()`	Возвращает битовую маску с установленными битами, соответствующими разным атрибутам метода (`public`, `private`, `protected`, `final`, `static`)
`mixed getValue ($object)`	Возвращает значение, которое имеет член на текущий момент для объекта `$object`
`void setValue ($object, $value)`	Устанавливает новое значение `$value` для члена в объекте `$object`
`ReflectionClass getDeclaringClass()`	Возвращает отражение класса, которому принадлежит член
`string getDocComment()`	Возвращает содержимое многострочного комментария, расположенного непосредственно перед классом

Листинг 25.19. Гипотетическая реализация класса `ReflectionProperty`

```
<?php
  class ReflectionProperty implements Reflector
  {
    public function __construct($class, $name);
```

```
    public function __toString();
    public static function export($class, $name, $return);
    public function getName();
    public function isPublic();
    public function isPrivate();
    public function isProtected();
    public function isStatic();
    public function isDefault();
    public function getModifiers();
    public function getValue($object);
    public function setValue($object, $value);
    public function getDeclaringClass();
    public function getDocComment();
  }
?>
```

В листинге 25.20 приводится пример установки нового значения открытому члену `$second`.

Листинг 25.20. Неявная установка нового значения члена при помощи отражения

```
<?php
  require_once("class.example.php");

  // Создаем объект класса example
  $exp = new example(1, 2, 3);

  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем отражение члена $second
  $snd = $obj->getProperty("second");
  // Устанавливаем значение
  $snd->setValue($exp, 100);
  // Выводим новое значение
  echo $exp->second; // 100
?>
```

Важно отметить, что при помощи метода `setValue()` невозможно установить значение закрытой переменной — в этом случае генерируется исключение `ReflectionException` (рис. 25.3).

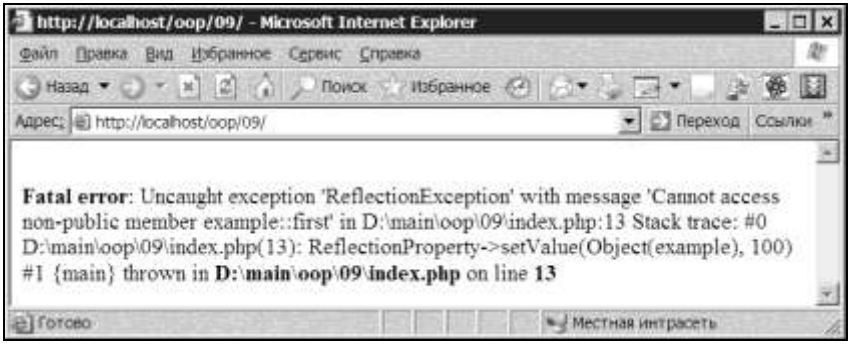


Рис. 25.3. При попытке доступа к закрытому члену генерируется исключение `ReflectionException`

25.8. Исключения механизма отражения

В предыдущем разделе была показана ситуация, при которой генерируется исключение `ReflectionException`. Исключение данного типа предназначено для обработки всех исключительных ситуаций механизма отражения. Класс `ReflectionException` наследуется от базового класса `Exception`, но не содержит никаких дополнительных методов. В листинге 25.21 приводится пример обработки исключительной ситуации при использовании механизма отражения.

Листинг 25.21. Обработка исключительной ситуации `ReflectionException`

```
<?php
  require_once("class.example.php");

  try
  {
    // Создаем объект класса example
    $exp = new example(1, 2, 3);

    // Получаем отражение класса
    $obj = new ReflectionClass("example");

    // Получаем отражение члена $first
    $fst = $obj->getProperty("first");
    // Пытаемся установить новое значение закрытому члену
    $fst->setValue($exp, 100);
    // Выводим новое значение
    echo $exp->getFirst();
  }
```

```
  catch(ReflectionException $exc)
  {
    echo "Исключение: {$exc->getMessage()}<br>";
    echo "в файле {$exc->getFile()}<br>";
    echo "в строке {$exc->getLine()}<br>";
  }
?>
```

В листинге 25.21 производится попытка установить новое значение закрытому члену `$first`, что приводит к генерации исключительной ситуации (рис. 25.4).



Рис. 25.4. Обработка исключительной ситуации

25.9. Отражение расширения. Класс *ReflectionExtension*

Отражение `ReflectionExtension` предназначено для получения информации о расширениях PHP.

> ***Замечание***
>
> Язык PHP имеет модульную структуру, имеется ядро PHP и ряд расширений, которые могут подключаться в конфигурационном файле php.ini при помощи директивы `extension`.

В табл. 25.7 представлен список методов класса `ReflectionExtension`.

Таблица 25.7. *Методы класса* `ReflectionExtension`

Метод	Описание
`__construct($name)`	Конструктор класса, создающий отражение для расширения с именем `$name`
`string __toString()`	Вывод информации при использовании объекта в строковом контексте

***Таблица 25.7** (окончание)*

Метод	Описание
`static string export($name, $return)`	По результату метод аналогичен функции `__toString()` — возвращает информацию о расширении `$name` в виде строки или выводит в окно браузера, если необязательный параметр `$retrun` равен `FALSE`. В отличие от метода `__toString()`, метод `export()` является статическим и допускает вызов без объявления объекта класса, например, `ReflectionExtension::export('standard')`
`string getName()`	Возвращает имя расширения
`string getVersion()`	Возвращает версию расширения
`ReflectionFunction[] getFunctions()`	Возвращает массив отражений для функций, входящих в состав расширения
`array getConstants()`	Возвращает ассоциативный массив с предопределенными константами, входящими в состав расширения
`array getINIEntries()`	Возвращает ассоциативный массив со значениями всех директив конфигурационного файла php.ini, которые влияют на работу расширения
`ReflectionClass[] getClasses()`	Возвращает массив отражений для классов, входящих в состав расширения
`array getClassNames()`	Возвращает массив с названиями классов, входящих в состав расширения

Класс `ReflectionExtension` является предопределенным и не требует объявления. Однако если бы класс реализовывался внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 25.22.

Листинг 25.22. Гипотетическая реализация класса `ReflectionExtension`

```
<?php
  class ReflectionExtension implements Reflector
  {
    public function __construct($name);
    public function __toString();
    public static function export($name, $return);
    public function getName();
    public function getVersion();
    public function getFunctions();
    public function getConstants();
    public function getINIEntries();
    public function getClasses();
    public function getClassNames();
  }
?>
```

Отражение `ReflectionExtension` может применяться не только для расширений, но и для ядра PHP, для этого конструктору класса следует передать имя "standard". В листинге 25.23 демонстрируется, как при помощи отражения расширения можно выяснить версию PHP.

Листинг 25.23. Получение версии PHP

```
<?php
  // Получаем отражение ядра PHP
  $obj = new ReflectionExtension("standard");
  // Выводим в окно браузера версию PHP
  echo $obj->getVersion();
?>
```

В листинге 25.24 приводится пример вывода директив конфигурационного файла php.ini и списка функций для расширения php_mysql, предназначенного для взаимодействия с СУБД MySQL.

> ***Замечание***
>
> Получить список загруженных в настоящий момент расширений PHP можно при помощи функции `get_loaded_extensions()`.

Листинг 25.24. Получение отражения расширения php_mysql

```
<?php
  // Получаем отражение расширения для работы с MySQL
  $obj = new ReflectionExtension("mysql");
  // Выводим директивы конфигурационного файла php.ini
  echo "<pre>";
  print_r($obj->getINIEntries());
  echo "</pre>";
  // Выводим список функций расширения
  $functions = $obj->getFunctions();
  foreach($functions as $funct)
  {
    echo $funct->getName()."<br>";
  }
?>
```

Результатом работы скрипта из листинга 25.24 будут следующие строки:

```
Array
(
    [mysql.allow_persistent] => 1
    [mysql.max_persistent] => -1
```

```
    [mysql.max_links] => -1
    [mysql.default_host] =>
    [mysql.default_user] =>
    [mysql.default_password] =>
    [mysql.default_port] =>
    [mysql.default_socket] =>
    [mysql.connect_timeout] => 60
    [mysql.trace_mode] =>
)
mysql_connect
mysql_pconnect
mysql_close
...
mysql_tablename
mysql_table_name
```

Важно отметить, что наряду с доступными функциями, предназначенными для использования внешним программистом, в списке приводятся внутренние функции, предназначенные для использования внутри библиотеки php_mysql.

25.10. Вспомогательный класс *Reflection*

Вспомогательный класс `Reflection` не реализует интерфейс `Reflector` и, следовательно, не является отражением. В состав класса входят лишь две статические функции, описание которых представлено в табл. 25.8.

Таблица 25.8. *Методы класса* `Reflection`

Метод	Описание
`public static array getModifierNames ($modifiers)`	Метод принимает битовую маску, которую возвращает метод `getModifiers()` классов отражений, и возвращает массив атрибутов в текстовом, а не числовом представлении
`public static export ($ref, $return)`	Метод принимает объект отражения `$refl` и вызывает его метод `__toString()`. Если необязательный параметр `$return` равен `TRUE`, метод возвращает строку, в противном случае результат выводится непосредственно в окно браузера

В листинге 25.25 демонстрируется использование метода `getModifierNames` для расшифровки битовой маски, получаемой для метода `getFirst()` класса `example` (листинг 25.10).

Листинг 25.25. Использование метода `getModifierNames()`

```
<?php
  require_once("class.example.php");
```

```
  // Получаем отражение класса
  $obj = new ReflectionClass("example");

  // Получаем битовую маску
  $mask = $obj->getMethod("getFirst")->getModifiers();

  // Расшифровываем содержимое битовой маски
  echo "<pre>";
  print_r(Reflection::getModifierNames($mask));
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 25.25 будет следующий дамп массива свойств:

```
Array
(
    [0] => public
)
```



Глава 26

Предопределенные классы

В PHP имеются готовые классы, которые не требуют объявления и могут использоваться непосредственно. Типичным примером предопределенного класса является класс исключения `Exception` или набор классов отражений. Предопределенные классы могут быть встроены в ядро или подключаться при помощи одного из расширений.

26.1. Класс *Dir*

Класс `Dir` является предопределенным классом, позволяющим получать доступ к каталогу. Объявление объекта класса `Dir` позволяет открыть каталог, а его методы — осуществить чтение содержимого каталога, заменяя набор функций `opendir()`, `rewinddir()`, `readdir()` и `closedir()`. Методы и члены класса `dir` перечислены в табл. 26.1.

Таблица 26.1. *Методы и члены класса* `Dir`

Элемент класса	Описание
`path`	Путь к каталогу
`handle`	Дескриптор открытого каталога
`read()`	Читает очередной элемент каталога, передвигая указатель каталога на одну позицию
`rewind()`	Сбрасывает указатель каталога в исходное состояние; применяется, когда в результате чтения каталога в цикле указатель устанавливается на конец каталога и его следует переместить в начало для повторного чтения
`close()`	Закрывает каталог

Объявление объекта класса `Dir` приводит к автоматическому открытию каталога, после чего становится доступно использование методов классов. Метод `read()` за один раз читает один элемент каталога, перемещая указатель каталога на одну позицию. Последовательный вызов метода `read()` позволяет обойти таким образом весь каталог до конца. В листинге 26.1 приводится скрипт, который выводит содержимое каталога oop.

Листинг 26.1. Чтение содержимого каталога

```
<?php
  // Открываем каталог
  $cat = dir("oop");
  // Читаем содержимое каталога
  while(($file = $cat->read()) !== false)
  {
    echo $file."<br />";
  }
  // Закрываем каталог
  $cat->close();
?>
```

Важно подчеркнуть, что дескриптор открытого каталога `$cat->handle` полностью совместим с классическими функциями для работы с каталогом. Скрипт из листинга 26.2 по результату полностью эквивалентен скрипту из листинга 26.1.

Листинг 26.2. Альтернативный способ чтения содержимого каталога

```
<?php
  // Открываем каталог
  $cat = dir("oop");
  // Читаем содержимое каталога
  while(($file = readdir($cat->handle)) !== false)
  {
    echo $file."<br />";
  }
  // Закрываем каталог
  closedir($cat->handle);
?>
```

Если в этом случае необходимо установить указатель каталога на его начало для повторного чтения, следует вызывать метод `rewind()`. В листинге 26.3 приводится скрипт, который подсчитывает количество файлов и подкаталогов, после чего осуществляет повторное чтение каталога для вывода списка его элементов.

Замечание

Из количества подкаталогов вычитается цифра 2, чтобы предотвратить учет двух скрытых служебных подкаталогов: `.` — текущий каталог и `..` — родительский каталог.

Листинг 26.3. Использование метода `rewind()`

```
<?php
  // Открываем каталог
  $dirname = "oop/";
  $cat = dir($dirname);

  // Устанавливаем счетчики файлов и подкаталогов в нулевое значение
  $file_count = 0;
  $dir_count = 0;

  // Подсчитываем количество файлов и подкаталогов
  while(($file = $cat->read()) !== false)
  {
    if (is_file($dirname.$file)) $file_count++;
    else $dir_count++;
  }
  // Не учитываем
  $dir_count = $dir_count - 2;
  // Выводим количество файлов и подкаталогов
  echo "Каталог $dirname содержит $file_count файлов
        и $dir_count подкаталогов<br />";

  // Устанавливаем указатель каталога в начало
  $cat->rewind();

  // Читаем содержимое каталога
  while(($file = $cat->read()) !== false)
  {
    if ($file != "." && $file != "..")
    {
      echo $file."<br />";
    }
  }
  // Закрываем каталог
  $cat->close();
?>
```

26.2. Классы времени

Основным рабочим классом является `DateTime`, который позволяет оперировать датой и временем. В табл. 26.2 и 26.3 представлены методы и константы класса `Datetime`.

Таблица 26.2. *Методы класса* `DateTime`

Метод	Описание
`__construct([$time [, $timezone]])`	Конструктор класса, принимающий два необязательных параметра. Первый `$time` определяет время (если параметр не передан, назначается текущее время), второй параметр `$timezone` представляет собой объект класса `DateTimeZone` и предназначен для задания часового пояса
`add($interval)`	Добавляет к текущей дате интервал `$interval`. Параметр `$interval` представляет собой объект класса `DateInterval`
`createFromFormat($format, $time [, $timezone])`	Создает объект `DateTime` из строки `$time`, формат которой задается строкой `$format`. Необязательный параметр `$timezone` представляет собой объект класса `DateTimeZone` и предназначен для задания часового пояса
`diff($datetime [, $absolute])`	Возвращает разницу между текущим объектом и объектом `$datetime`. Результат возвращается в виде объекта класса `DateInterval`. Если `$absolute` принимает значение `TRUE`, результат всегда имеет положительное значение, не зависимо от того, большая или меньшая дата вычитается из текущего объекта `DateTime`
`format($format)`	Возвращает время в виде строки, отформатированной в соответствии с `$format`
`getLastErrors()`	Возвращает массив предупреждений и ошибок, которые генерируются во время разбора строки со временем
`getOffset()`	Возвращает в секундах отклонение от часового пояса для летнего времени, т. е. для даты в летнем периоде возвращается 3600, для зимнего периода — 0
`getTimestamp ()`	Возвращает время в UNIXSTAMP-формате, т. е. количество секунд, прошедших с полуночи 1 января 1970 года
`getTimezone()`	Возвращает текущий часовой пояс в виде объекта `DateTimeZone`
`modify($modify)`	Изменяет значение даты и времени в соответствии со строкой `$modify`, синтаксис которой определяется форматом функции `strtotime()`
`setDate($year, $month, $day)`	Устанавливает дату, используя в качестве входных параметров: `$year` — год, `$month` — месяц и `$day` — день
`setISODate($year, $week [, $day])`	Устанавливает дату, используя в качестве входных параметров: `$year` — год, `$week` — номер недели и `$day` — количество дней от начала недели

***Таблица 26.2** (окончание)*

Метод	Описание
`setTime($hour, $minute [, $second])`	Устанавливает время, используя в качестве входных параметров: `$hour` — часы, `$minute` — минуты и `$second` — секунды
`setTimestamp($unixtimestamp)`	Устанавливает дату и время при помощи параметра `$unixtimestamp`, заданного в UNIXSTAMP-формате, т. е. количество секунд, прошедших с полуночи 1 января 1970 года
`setTimezone($timezone)`	Устанавливает часовой пояс при помощи объекта `$timezone` класса `DateTimeZone`
`sub($interval)`	Вычитает из текущей даты интервал `$interval`. Параметр `$interval` представляет собой объект класса `DateInterval`

В листинге 26.4 приводится пример создания объекта `$date` класса `DateTime` с текущей датой и вывод даты при помощи метода `format()` в окно браузера. Строка форматирования метода `format()` имеет тот же синтаксис, что и функция `date()` *(см. разд. 11.3)*.

Листинг 26.4. Использование объекта класса `DateTime`

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = new DateTime();
  echo $date->format("d-m-Y H:i:s"); // 13-07-2009 16:19:52
?>
```

По умолчанию создается объект `$date`, содержащий текущие дату и время. Для того чтобы задать произвольную дату, необходимо передать в качестве параметра конструктора строку в формате "YYYY-MM-DD HH:MM:SS" (листинг 26.5).

Листинг 26.5. Явное задание даты

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = new DateTime("2009-10-01 00:00:00");
  echo $date->format("d-m-Y H:i:s"); // 01-10-2009 00:00:00
?>
```

Следует отметить, что если часовой пояс не задан в конфигурационном файле php.ini или явно при помощи функции `date_default_timezone_set()`, при выполнении скрипта из листинга 24.4 генерируется исключительная ситуация (рис. 26.1).

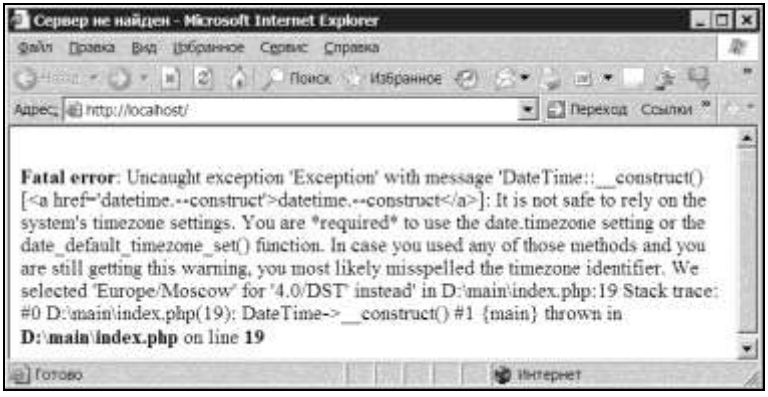


Рис. 26.1. Исключительная ситуация из-за невыставленного часового пояса

Избежать генерации исключительной ситуации можно также, передав конструктору класса `DateTime` второй необязательный объект `$timezone` класса `DateTimeZone`, позволяющий явно определить часовой пояс (листинг 26.6).

Листинг 26.6. Явная инициализация часового пояса

```
<?php
  $date = new DateTime("2009-10-01 00:00:00",
                       new DateTimeZone("Europe/Moscow"));
  echo $date->format("d-m-Y H:i:s"); // 13-07-2009 16:19:52
?>
```

Для удобства вывода даты в различных форматах, класс `DateTime` содержит несколько констант-строк с популярными форматами времени (табл. 26.3).

***Таблица 26.3.** Константы класса `DateTime`*

Константа	Формат
`DateTime::ATOM`	`"Y-m-d\TH:i:sP"`
`DateTime::COOKIE`	`l, d-M-y H:i:s T`
`DateTime::ISO8601`	`Y-m-d\TH:i:sO`
`DateTime::RFC822`	`D, d M y H:i:s O`
`DateTime::RFC850`	`l, d-M-y H:i:s T`
`DateTime::RFC1036`	`D, d M y H:i:s O`
`DateTime::RFC1123`	`D, d M Y H:i:s O`
`DateTime::RFC2822`	`D, d M Y H:i:s O`
`DateTime::RFC3339`	`Y-m-d\TH:i:sP`
`DateTime::RSS`	`D, d M Y H:i:s O`
`DateTime::W3C`	`Y-m-d\TH:i:sP`

В листинге 26.7 приводится пример использования константы `DateTime::COOKIE`, для формирования временной метки в формате, используемом в HTTP-заголовках `Set-Cookie`.

Листинг 26.7. Использование констант класса `DateTime`

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = new DateTime("2009-10-01 00:00:00");
  echo $date->format(DateTime::COOKIE);
?>
```

Помимо конструктора класса `DateTime`, получить объект можно при помощи статического метода `createFromFormat()`, который получает в качестве первого параметра строку форматирования (синтаксис функции `date()`), а в качестве второго параметра временную метку, удовлетворяющую строке форматирования (листинг 26.8).

Листинг 26.8. Использование метода `createFromFormat()`

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = DateTime::createFromFormat("Y-m-d H:i:s",
                                     "2008-10-01 00:00:00");
  echo $date->format(DateTime::ISO8601);
?>
```

Среди остальных методов класса `DateTime` выделяются методы `add()`, `sub()` и `diff()`, которые работают с временными интервалами. Все три метода работают с классом `DateInterval`, который моделирует разницу между двумя временными точками. Метод `add()` позволяет добавить интервал, `sub()` — вычесть, а `diff()` получить разницу между двумя временными точками.

Самый простой способ получить объект класса `DateInterval` — воспользоваться методом `diff()`, произведя вычитание одного объекта класса `DateTime` из другого (листинг 26.9).

Листинг 26.9. Использование метода `diff()`

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = new DateTime('2008-08-03 14:52:10');
  $nowdate = new DateTime();
  $interval = $nowdate->diff($date);
  // Выводим результаты
  echo $date->format("d-m-Y H:i:s")."<br />";
```

```
  echo $nowdate->format("d-m-Y H:i:s")."<br />";
  // Выводим разницу
  echo $interval->format("%Y-%m-%d %H:%S")."<br />";
  // Выводим дамп интервала
  echo "<pre>";
  print_r($interval);
  echo "</pre>";
?>
```

В листинге 24.9 вычисляется разница во времени между текущей датой и 8 августа 2008 года. Для этого создаются два объекта класса `DateTime`: `$date` и `$nowdate()`, при помощи метода `diff()` объекта `$nowdate()` из него вычитается объект `$date`. Результатом этой операции является объект `$interval` класса `DateInterval`. На рис. 26.2 приводится результат работы скрипта из листинга 24.9.

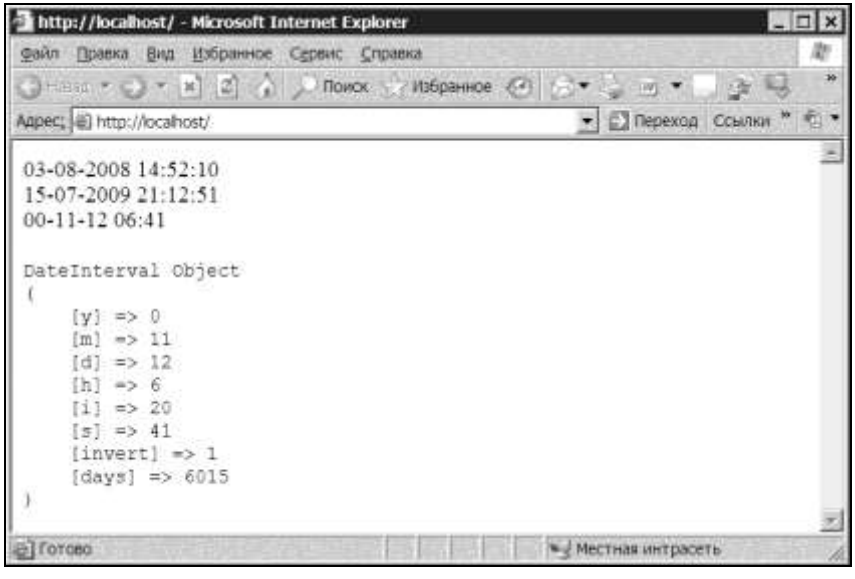


Рис. 26.2. Результат выполнения скрипта из листинга 26.9

Скрипт из листинга 26.9 помимо форматированных результатов выводит дамп объекта `DateInterval`, из которого видно, что объект содержит восемь открытых членов:

- `y` — годы;
- `m` — месяцы;
- `d` — дни;
- `h` — часы;
- `i` — минуты;
- `s` — секунды;

- `invert` — принимает 1, если интервал отрицательный, и 0, если интервал положительный;
- `days` — разница в днях.

Член `invert` влияет на результаты вычисления при помощи методов `add()` и `sub()`, осуществляющих сложение и вычитание даты `DateTime` и интервала `DateInterval`. Для того чтобы получить беззнаковый интервал, второму аргументу метода `diff()` следует передать значение `TRUE` (листинг 26.10).

Листинг 26.10. Использование второго параметра метода `diff()`

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $date = new DateTime('2008-08-03 14:52:10');
  $nowdate = new DateTime();
  $interval = $nowdate->diff($date, true);
  // Выводим дамп интервала
  echo "<pre>";
  print_r($interval);
  echo "</pre>";
?>
```

Результат работы скрипта из листинга 26.10 представлен на рис. 26.3. Как видно, член `invert` объекта класса `DateInterval` принимает значение 0.

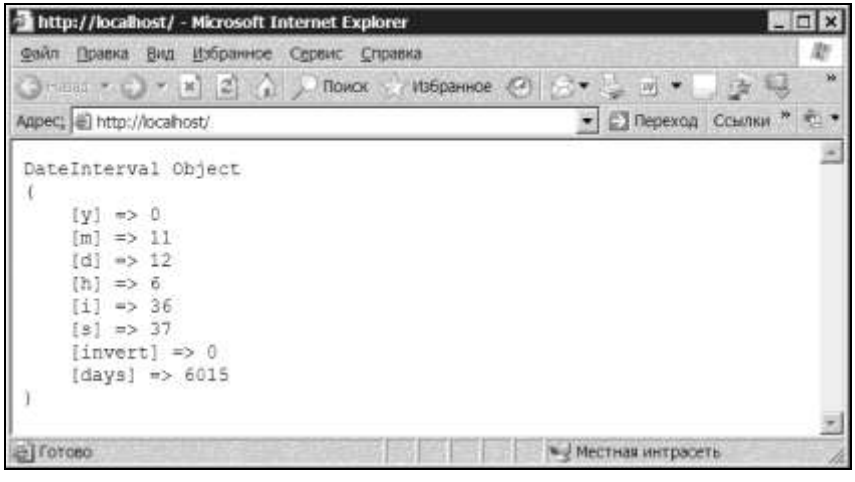


Рис. 26.3. Дамп объекта класса `DateInterval`

Для получения произвольного объекта класса `DateInterval` необходимо воспользоваться конструктором. Конструктор принимает строку специального формата, описа-

ние даты в которой начинается с символа P, а времени — с символа T. В табл. 26.4 приводятся коды для всех временных величин, используемых для создания интервала.

Таблица 26.4. *Коды инициализации, используемые конструктором* `DateInterval`

Код	Пример	Описание
Y	P3Y	Год
M	P3M	Месяц
D	P3D	День
H	T3H	Час
M	T3M	Минута
S	T3S	Секунда

В столбце "Пример" табл. 26.4 приводится запись для 3 единиц, например, P3Y означает три года, T3S — 3 секунды. Величины можно комбинировать, например, запись P3Y1M14D означает 3 года, 1 месяц и 14 дней, запись T12H19M2S означает 12 часов, 19 минут и 2 секунды. Дату и время можно объединять в строку — P3Y1M14DT12H19M2S. В листинге 26.11 создается интервал, который вычитается из текущей даты при помощи метода sub().

> ***Замечание***
>
> Порядок вхождения разных кодов в строку не имеет значения, можно перечислять сначала дни, потом месяцы, затем года, можно использовать обратный порядок.

Листинг 26.11. Создание интервала `DateInterval` при помощи конструктора

```
<?php
  @date_default_timezone_set("Europe/Moscow");
  $nowdate = new DateTime();
  $date = $nowdate->sub(new DateInterval("P3Y1M14DT12H19M2S"));
  echo $date->format("Y-m-d H:i:s");
?>
```

26.3. Библиотека SPL

Библиотека SPL — это сокращение от Standard PHP Library (Стандартная библиотека PHP). Программистов, знакомых с C++, придется разочаровать: несмотря на то, что данная библиотека претендует на лавры STL из стандартной библиотеки C++, до законченного варианта SPL еще очень далеко. Библиотека достаточно абстрактна, не содержит алгоритмов и, по сути, поддерживает только итераторы.

Замечание

Слово "стандартный" в компьютерной области имеет большое значение. Создано огромное количество библиотек, как платформонезависимых, так и ориентирующихся на конкретную аппаратную или операционную реализацию. Прилагательное "стандартный" означает, что любая реализация компилятора или интерпретатора обязана содержать "стандартную" библиотеку, а любой код, созданный с использованием исключительно стандартных средств, может быть откомпилирован под любой платформой, будь то Windows, один из вариантов Linux, Mac OS или любая другая операционная система. PHP не стандартизирован — его выпускает одна группа разработчиков, что обеспечивает ему высокую переносимость. Однако международной стандартизации язык не подвергался, поэтому лейбл "стандартный" является, скорее, маркетинговым ходом разработчика библиотеки.

Библиотека STL из стандартной библиотеки C++ имеет набор предопределенных контейнеров, итераторов, способных работать с контейнерами, и алгоритмов, оперирующих итераторами. Разработчикам остается лишь заполнять контейнеры и применять к ним алгоритмы поиска, сортировки, объединения, вычитания и т. п. Появление такой библиотеки вызвано тем, что C++ является достаточно суровым языком программирования — сложен и очень требователен к программистам. Например, отсутствуют массивы, как и строки в классическом смысле этого слова: имеются лишь участки памяти с условными границами. Об ассоциативных массивах и регулярных выражениях вообще речи не идет. Программист должен постоянно помнить о необходимости выделения и освобождения памяти. Язык является строго типизированным и, как следствие, поддерживает шаблоны функций и классов, что еще более его усложняет. Поэтому вскоре возникла идея о создании универсальной библиотеки, которая возьмет на себя рутинные, но часто встречающиеся операции.

Так как PHP является интерпретируемым языком программирования, он обладает как широким набором возможностей, так и рядом проблем — отслеживание памяти, строгая типизация, шаблоны, отсутствие ассоциативных массивов — вообще отсутствуют или решены на уровне языка программирования. К достоинствам языка можно также отнести огромный массив функций, а простота инициализации массивов и отсутствие необходимости выделения под них памяти сводит большинство задач к использованию массивов.

Все это приводит к тому, что библиотека SPL является несколько искусственным образованием. Если STL является стандартом де-факто в мире C++, то о библиотеке SPL в PHP практически никто не слышал.

Замечание

Данный раздел является введением в библиотеку SPL и не претендует на полноту (тем более, что библиотека находится в стадии разработки). Здесь будет описана лишь часть классов и интерфейсов, поддерживаемых библиотекой SPL. За детальным описанием библиотеки следует обратиться к официальной документации.

26.3.1. Итераторы

Итератор — это объект, позволяющий обходить коллекцию способом, не зависящим от внутреннего устройства коллекции. Независимо от того, какую коллекцию обходит итератор, он ведет себя одинаково и поддерживает одни и те же методы.

Это позволяет построить с использованием итераторов универсальные алгоритмы (например, сортировки). Далее для каждой новой коллекции потребуется только создать итераторы, алгоритм может быть к нему применен автоматически и его не потребуется создавать заново. Как уже упоминалось ранее, алгоритмы в SPL как таковые отсутствуют, а все операции с итераторами сводятся к тому, что появляется возможность использовать итератор в цикле `foreach`. Зачем это нужно?

Многие операции в PHP осуществляются по следующему алгоритму: открыть ресурс, обойти его в цикле, закрыть ресурс. Например, в листинге 26.12 приводится пример обхода в цикле `while` таблицы, полученной в результате `SELECT`-запроса к таблице `catalogs`.

Листинг 26.12. Обход в цикле результирующей таблицы

```
<?php
  ...
  // Получение результирующей таблицы
  $query = "SELECT * FROM catalogs"
  $cat = mysql_query();
  if (!$cat) exit(mysql_error());

  // Обход результирующей таблицы
  while($catalog = mysql_fetch_array($cat))
  {
    // Работа с массивом $catalog
  }
  ...
?>
```

В листинге 26.13 приводится пример обхода открытого каталога при помощи функций `opendir()` и `readdir()`.

Листинг 26.13. Обход каталога в цикле

```
<?php
  ...
  // Открытие каталога
  $dir = opendir('oop/');

  // Обход каталога
  while(($file = readdir($dh)) !== false)
  {
    // Работа с элементами каталога
  }
  ...
?>
```

В листинге 26.14 приводится пример построчного обхода файла, содержимое которого преобразуется в массив `$arr`.

Листинг 26.14. Построчный обход содержимого файла

```
<?php
  ...
  // Получение содержимого файла в виде массива
  $arr = file("text.txt");

  // Обход файла
  for($i = 0; $i < count($arr); $i++)
  {
     // Работа со строками файла
  }
  ...
?>
```

Каждый из примеров в листингах 26.12 и 26.13 демонстрирует обход коллекции (записей, имен файлов и подкаталогов, строк файлов), однако в каждом случае используется свой вариант цикла. Итераторы позволяют применять в каждом из этих случаев единый метод обхода коллекции при помощи цикла `foreach()` (листинг 26.15).

Листинг 26.15. Использование итератора

```
<?php
  ...
  // Получение коллекции
  $iterator = new InteratorClass();

  // Обход коллекции
  foreach($iterator as $element)
  {
    // Работа с элементом коллекции $element
  }
  ...
?>
```

Если листинг 26.15 реализует какой-либо алгоритм — сортировка, постраничный вывод и т. п., то для его применения к различным коллекциям достаточно объявить соответствующий коллекции итератор `$iterator`.

26.3.2. Интерфейс *Iterator*

Каждый итератор реализует интерфейс `Iterator`, который требует реализации следующих методов:

- `current()` — вернуть текущий элемент;
- `key()` — вернуть ключ текущего элемента;
- `next()` — переместить итератор к следующему элементу;
- `rewind()` — поместить итератор в начало коллекции;
- `valid()` — проверить текущий элемент после вызова методов `rewind()` и `next()`; метод возвращает `false`, если итератор находится за пределами коллекции (закончились элементы).

В листинге 26.16 приводится пример создания класса собственного итератора `IteratorDir`, реализующий интерфейс `Iterator`. Класс `IteratorDir` реализует итератор для доступа к элементам каталога.

Листинг 26.16. Класс `IteratorDir`

```
<?php
  class IteratorDir implements Iterator
  {
    // Путь к каталогу
    protected $dir;

    // Номер файла или подкаталога в каталоге
    protected $key;
    // Имя файла или подкаталога в каталоге
    protected $file;
    // Флаг, принимающий значение FALSE или TRUE в зависимости
    // от того, закончились файлы или нет
    protected $valid;

    // Конструктор
    public function __construct($path)
    {
      $this->dir = opendir($path);
    }

    // Деструктор
    public function __destruct()
    {
      closedir($this->dir);
    }
```

```
    // Установка итератора в начало
    public function rewind()
    {
      $this->key = 0;
      rewinddir($this->dir);
      $this->next();
    }

    // Чтение следующего элемента
    public function next()
    {
      $this->key++;
      $this->valid = false !== ($this->file = readdir($this->dir));
    }

    // Получение ключа
    public function key()
    {
      return $this->key;
    }

    // Получение значения
    public function current()
    {
      return $this->file;
    }

    // Возвращает флаг, закончились элементы в коллекции или нет
    public function valid()
    {
      return $this->valid;
    }
  }
?>
```

Теперь, для того чтобы обойти каталог в цикле, достаточно объявить объект класса `IterarorDir`, передав конструктору путь к нему в качестве параметра (листинг 26.17).

Листинг 26.17. Использование объекта класса `IteratorDir`

```
<?php
  require_once("class.iteratordir.php");

  $dir = new IteratorDir('oop');
```

```
  foreach($dir as $file)
  {
    echo $file."<br>";
  }
?>
```

Результатом работы скрипта будет список файлов и подкаталогов каталога oop.

26.3.3. Класс *DirectoryIterator*

Библиотека SPL содержит уже готовые классы, реализующие интерфейс `Iterator` (объекты которых могут использоваться в цикле `foreach`). Одним из таких классов является `DirectoryInterator`, предоставляющий доступ к содержимому каталога. В листинге 26.18 приводится пример использования итератора.

> ***ЗАМЕЧАНИЕ***
>
> По результату выполнения скрипты из листингов 26.17 и 26.18 эквивалентны.

Листинг 26.18. Использование класса `DirectoryIterator`

```
<?php
  $dir = new DirectoryIterator('oop');
  foreach($dir as $file)
  {
    echo $file."<br>";
  }
?>
```

Объект `$file` здесь выступает не как строка, а как объект, реализующий методы, представленные в табл. 26.5.

Таблица 26.5. *Методы класса* `DirectoryIterator`

Метод	Описание
`current()`	Возвращает ссылку на объект `$this`
`getATime()`	Возвращает время последнего доступа к файлу или подкаталогу (в секундах, прошедших с полуночи 1 января 1970 года)
`getCTime()`	Возвращает время последнего изменения файла или каталога (в секундах, прошедших с полуночи 1 января 1970 года)
`getFilename()`	Возвращает имя файла или подкаталога
`getGroup()`	Возвращает имя группы, в которую входит файл (только для UNIX-подобных операционных систем)

***Таблица 26.5** (окончание)*

Метод	Описание
`getInode()`	Возвращает номер узла в файловой системе, соответствующий файлу или подкаталогу (только для UNIX-подобных операционных систем)
`getMTime()`	Возвращает время последней модификации файла или каталога (в секундах, прошедших с полуночи первого января 1970 года)
`getOwner()`	Возвращает имя владельца файла (только для UNIX-подобных операционных систем)
`getPath()`	Возвращает имя каталога (без имени файла и подкаталога)
`getPathname()`	Возвращает путь к файлу, включая название каталога, а также название файла или подкаталога
`getPerms()`	Возвращает права доступа, назначенные файлу или подкаталогу (только для UNIX-подобных операционных систем)
`getSize()`	Возвращает объем файла (в байтах); для подкаталогов всегда возвращается значение 0
`getType()`	Возвращается тип текущего элемента каталога: `dir` для каталога и `file` — для файла
`isDir()`	Возвращает `TRUE`, если текущий элемент является каталогом, и `FALSE` в противном случае
`isExecutable()`	Возвращает `TRUE`, если текущий файл является исполняемым, а каталог допускает переход в него, и `FALSE` в противном случае
`isFile()`	Возвращает `TRUE`, если текущий элемент является файлом, и `FALSE` в противном случае
`isLink()`	Возвращает `TRUE`, если текущий элемент является ссылкой, и `FALSE` в противном случае
`isReadable()`	Возвращает `TRUE`, если файл или подкаталог доступны для чтения, и `FALSE` в противном случае
`isWritable()`	Возвращает `TRUE`, если файл доступен для записи, а каталог допускает создание внутри себя файлов, и `FALSE` в противном случае

Например, для того чтобы после имени файла вывести его размер, достаточно воспользоваться методом `getSize()` (листинг 26.19).

Листинг 26.19. Использование методов класса `DirectoryIterator`

```
<?php
  $dir = new DirectoryIterator('oop');
  foreach($dir as $file)
  {
```

```
    // Выводим только файлы
    if ($file->isFile())
    {
      // Имя файла и его размер
      echo $file." ".$file->getSize()."<br>";
    }
  }
?>
```

26.3.4. Класс *FilterIterator*

Элементы коллекции могут быть отфильтрованы при помощи итератора, производного от класса `FilterIterator`. В листинге 26.20 приводится пример создания фильтра `ExtensionFilter` для класса `DirectoryIterator`, который фильтрует все файлы, обладающие расширением php.

Листинг 26.20. Создание фильтра `ExtensionFilter`

```
<?php
  class ExtensionFilter extends FilterIterator
  {
    // Фильтруемое расширение
    private $ext;
    // Итератор DirectoryIterator
    private $it;

    // Конструктор
    public function __construct(DirectoryIterator $it, $ext)
    {
       parent::__construct($it);
       $this->it = $it;
       $this->ext = $ext;
    }

    // Метод, определяющий, удовлетворяет текущий элемент фильтру или нет
    public function accept()
    {
      if (!$this->it->isDir())
      {
         $ext = array_pop(explode('.', $this->current()));
         return $ext != $this->ext;
      }
      return true;
    }
  }
?>
```

Использование класса `ExtensionFilter` совместно с классом `DirectoryIterator` приведет к тому, что из результирующего списка файлов будут исключены все файлы с расширением php (листинг 26.21).

Листинг 26.21. Вывод списка файлов и каталогов за исключением PHP-файлов

```
<?php
  require_once("class.extension.filter.php");

  $filter = new ExtensionFilter(
               new DirectoryIterator('oop'), 'php');

  foreach($filter as $file)
  {
    echo $file."<br>";
  }
?>
```

26.3.5. Класс *LimitIterator*

Класс `LimitIterator` и его производные позволяют осуществить постраничный вывод. Конструктор класса принимает в качестве первого параметра итератор, в качестве второго параметра — начальную позицию (по умолчанию равную 0), а в качестве третьего — смещение от позиции. При этом итератор работает с участком коллекции, определяемым вторым и третьим параметрами. В листинге 26.22 приводится пример вывода первых пяти элементов каталога oop с исключением PHP-файлов.

Листинг 26.22. Использование класса `LimitIterator`

```
<?php
  require_once("class.extension.filter.php");

  $limit =  new LimitIterator(
              new ExtensionFilter(new DirectoryIterator('oop'), "php"),
              0, 5);

  foreach($limit as $file)
  {
    echo $file."<br>";
  }
?>
```

26.3.6. Рекурсивные итераторы

Рекурсивной называется функция, которая вызывает сама себя. Подобные конструкции часто используются для обхода древовидных структур. Например, каталоги могут быть вложены друг в друга, и для вывода содержимого каталога, включая все вложенные подкаталоги, может потребоваться рекурсивная функция. Типичный пример рекурсивной функции приводится в листинге 26.23.

Листинг 26.23. Рекурсивная функция для вывода содержимого каталога

```
<?php
  function recurse_dir($path)
  {
    static $depth = 0;

    $dir = opendir($path);
    while(($file = readdir($dir)) !== false)
    {
      if ($file == '.' || $file == '..' ) continue;
      echo str_repeat("-", $depth)." $file<br>";

      if (is_dir("$path/$file"))
      {
        $depth++;
        recurse_dir("$path/$file");
        $depth--;
      }
    }
    closedir($dir);
  }

  recurse_dir('oop');
?>
```

Скрипт из листинга 26.23 выводит список файлов и подкаталогов каталога oop, определяя степень вложенности при помощи статической переменной `$depth`. Чтение содержимого каталога выполняется в цикле; если текущий элемент является файлом — его название выводится в окно браузера, если подкаталогом — для него вызывается функция `recurse_dir()`. При спуске на один уровень значение переменной `$depth` увеличивается на единицу, при возвращении — уменьшается. Это позволяет выводить перед именем файла то количество символов -, которое соответствует уровню "залегания" файла.

Решение проблемы вывода содержимого вложенного каталога можно решить при помощи итераторов (листинг 26.24).

Листинг 26.24. Рекурсивный обход каталога при помощи итераторов

```
<?php
  $dir = new RecursiveIteratorIterator(
           new RecursiveDirectoryIterator('oop'),
         true);

  foreach ($dir as $file)
  {
    echo str_repeat("-", $dir->getDepth())." $file<br>";
  }
?>
```

Метод `getDepth()` итератора `RecursiveIteratorIterator` возвращает глубину вложения элемента.



Глава 27

Работа с СУБД MySQL

В настоящее время ни одно серьезное Web-приложение не может обойтись без работы с базой данных, обеспечивающей почти безграничные возможности манипулирования данными: сортировка, поиск, преобразование, редактирование и многое другое. При этом все низкоуровневые операции с файловой системой скрыты от программиста за несложными SQL-запросами. Работа с базой данных имеет как свои преимущества, так и недостатки. К достоинствам можно отнести значительное (иногда в 2—3 раза) снижение объема кода по сравнению с файловыми вариантами Web-приложений, что сокращает время разработки и упрощает процесс отладки. К недостаткам можно отнести зависимость приложения не только от работоспособности Web-сервера, но и от работоспособности сервера баз данных. Разумеется, вероятность того, что один из двух серверов может выйти из строя, выше, чем если бы работа Web-приложения зависела только от одного сервера. Кроме того, непосредственная работа с файловой системой осуществляется быстрее по сравнению со случаем, когда в качестве посредника для работы с ней выбирается СУБД. Так поисковая система Google, известная высокой скоростью обработки запросов, основана на модели плоских файлов и не использует базы данных. С другой стороны, PHP, как интерпретируемый язык, не может достичь производительности баз данных, реализованных на C.

В последнее время СУБД MySQL стала стандартом де-факто, благодаря своей надежности, производительности, низкой стоимости, простоте установки, настройки и обслуживания.

> ***Замечание***
>
> Установка и настройка сервера и клиентов MySQL рассматривается в *приложении 2*.

Поскольку MySQL будет использоваться в качестве СУБД на протяжении всей книги, эта глава целиком посвящена приемам работы с MySQL.

> ***Замечание***
>
> Данная глава является вводной в СУБД MySQL. Всестороннему рассмотрению предмета посвящены наши книги "Самоучитель MySQL 5", "MySQL 5" и "MySQL на примерах" (см. список литературы в конце книги).

27.1. Введение в СУБД и SQL

Системы управления базами данных (СУБД) предназначены для управления большими объемами данных. По сути, база данных — это те же файлы, в которых хранится информация. Сами по себе базы данных не представляли бы никакого интереса, если бы не было систем управления базами данных (СУБД). СУБД — это программный комплекс, который выполняет все низкоуровневые операции по работе с файлами базы данных, оставляя программисту оперировать логическими конструкциями при помощи языка программирования.

> ***Примечание***
>
> База данных — это файловое хранилище информации, и не более. Программные продукты типа MySQL, Oracle, Dbase, Informix, PostgreSQL и др. — это системы управления базами данных (СУБД). Базы данных везде одинаковы — это файлы с записанной в них информацией. Все указанные программные продукты отличаются друг от друга способом организации работы с файловой системой. Однако для краткости эти СУБД часто называют просто базами данных. Следует учитывать, что когда мы будем говорить "база данных" — речь будет идти именно о СУБД.

Язык программирования, с помощью которого пользователь общается с СУБД (или, как говорят, "осуществляет запросы к базе данных"), называется SQL (Structured Query Language, структурированный язык запросов).

Таким образом, для получения информации из базы данных необходимо направить ей запрос, созданный с использованием SQL, результатом выполнения которого будет результирующая таблица (рис. 27.1).

Несмотря на то, что SQL называется "языком запросов", в настоящее время этот язык представляет собой нечто большее, чем просто инструмент для создания запросов. С помощью SQL осуществляется реализация всех возможностей, которые представляются пользователям разработчиками СУБД, а именно:

- выборка данных (извлечение из базы данных содержащейся в ней информации);
- организация данных (определение структуры базы данных и установление отношений между ее элементами);
- обработка данных (добавление/изменение/удаление);
- управление доступом (ограничение возможностей ряда пользователей на доступ к некоторым категориям данных, защита данных от несанкционированного доступа);
- обеспечение целостности данных (защита базы данных от разрушения);
- управление состоянием СУБД.

SQL не является специализированным языком программирования, т. е. в отличие от языков высокого уровня (PHP, C++, Pascal и т. д.), с его помощью невозможно создать полноценную программу. Все запросы выполняются либо в специализированных программах, либо из прикладных программ при помощи специальных библиотек.

Несмотря на то, что язык запросов SQL строго стандартизирован, существует множество его диалектов: по сути, каждая база данных реализует собственный диалект со своими особенностями и ключевыми словами, недоступными в других базах данных.

Такая ситуация связана с тем, что стандарты SQL появились достаточно поздно, в то время как компании-поставщики баз данных существуют давно и обслуживают большое число клиентов, для которых требуется обеспечить обратную совместимость со старыми версиями программного обеспечения. Кроме того, рынок реляционных баз данных оперирует сотнями миллиардов долларов в год, все компании находятся в жесткой конкуренции и постоянно совершенствуют свои продукты. Поэтому, когда дело доходит до принятия стандартов, базы данных уже имеют реализацию той или иной особенности, и комиссии по стандартам в условиях жесткого давления приходится выбирать в качестве стандарта решение одной из конкурирующих фирм.

Теория реляционных баз данных была разработана доктором Коддом из компании IBM в 1970 году. Одной из задач реляционной модели была попытка упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы, на пересечении которых расположены данные.

Можно кратко сформулировать особенности реляционной базы данных.

- Данные хранятся в таблицах, состоящих из столбцов и строк.
- На пересечении каждого столбца и строки находится только одно значение.
- У каждого столбца есть свое имя, которое служит его названием, и все значения в одном столбце имеют один тип. Например, в столбце `id_catalog` все значения имеют целочисленный тип, а в столбце `name` — текстовый.
- Столбцы располагаются в определенном порядке, который задается при создании таблицы, в отличие от строк, которые располагаются в произвольном порядке. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.
- Запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объект запросов.



Рис. 27.1. Таблица реляционной базы данных

На рис. 27.1 приведен пример таблицы `catalogs` базы данных электронного магазина изделий компьютерных комплектующих, которые подразделяются на разделы. Каждая строка этой таблицы представляет собой один вид товарных позиций, для описания

которой используется поле `id_catalog` — уникальный номер раздела, `name` — название раздела. Столбцы определяют структуру таблицы, а строки — количество записей в таблице. Как правило, одна база данных содержит несколько таблиц, которые могут быть как связаны друг с другом, так и независимы друг от друга.

На рис. 27.2 приведена структура базы данных, состоящей из двух таблиц: `catalogs` и `products`. Таблица `catalogs` определяет количество и названия разделов, а таблица `products` содержит описание товарных позиций. Одной товарной позиции соответствует одна строка таблицы. Последнее поле таблицы `products` содержит значения из поля `id_catalog` таблицы `catalogs`. По этому значению можно однозначно определить, в каком разделе находится товарная позиция. Таким образом, таблицы оказываются связанными друг с другом. Эта связь условна, она может отсутствовать и в большинстве случаев проявляется только в результате специальных запросов, однако именно благодаря наличию связей такая форма организации информации получила название реляционной (связанной отношениями).

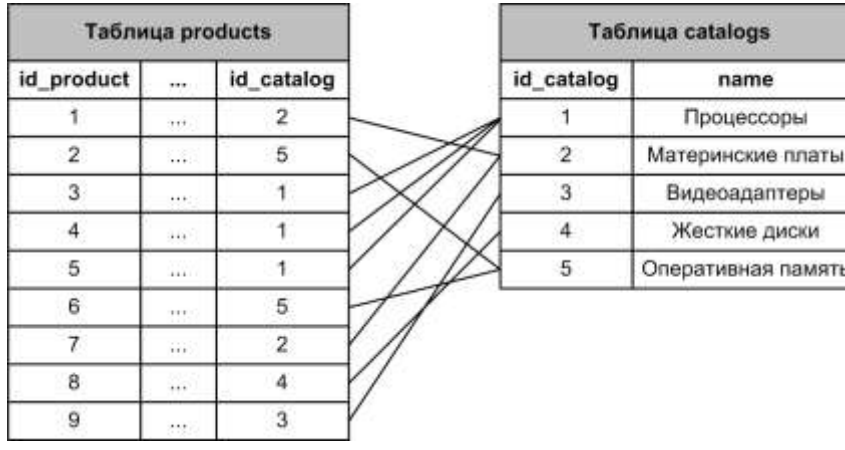


Таблица products

id_product	...	id_catalog
1	...	2
2	...	5
3	...	1
4	...	1
5	...	1
6	...	5
7	...	2
8	...	4
9	...	3

Таблица catalogs

id_catalog	name
1	Процессоры
2	Материнские платы
3	Видеоадаптеры
4	Жесткие диски
5	Оперативная память

Рис. 27.2. Связанные друг с другом таблицы

ПРИМЕЧАНИЕ

В математике таблица, все строки которой отличаются друг от друга, называется *отношением* (relation). Именно этому термину реляционные базы данных и обязаны своим названием.

Сила реляционных баз данных заключается не только в уникальной организации информации в виде таблиц. Запросы к таблицам базы данных также возвращают таблицы, которые называют *результирующими таблицами*. Даже если возвращается всего одно значение, его принято считать таблицей, состоящей из одного столбца и одной строки. То, что SQL-запрос возвращает таблицу, очень важно: это означает, что результаты запроса можно записать обратно в базу данных в виде таблицы, а результаты двух или более запросов, которые имеют одинаковую структуру, можно объединить в одну таблицу. И, наконец, это говорит о том, что результаты запроса сами могут стать предметом дальнейших запросов.

27.2. Первичные ключи

Строки в реляционной базе данных неупорядочены: в таблице нет "первой", "последней", "тридцать шестой" и "сорок третьей" строки. Возникает вопрос: каким же образом выбирать в таблице конкретную строку? Для этого в правильно спроектированной базе данных для каждой таблицы создается один или несколько столбцов, значения которых во всех строках различны. Такой столбец называется *первичным ключом* таблицы. Первичный ключ обычно сокращенно обозначают как PK (primary key). Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа, благодаря чему каждая строка таблицы обладает своим уникальным идентификатором. Так, на рис. 27.2 в качестве первичного ключа таблицы `products` выступает поле `id_product`, а в качестве первичного ключа таблицы `catalogs` — `id_catalog`.

По способу задания первичных ключей различают *логические* (естественные) ключи и *суррогатные* (искусственные).

Для логического задания первичного ключа необходимо выбрать в таблице столбец, который может однозначно установить уникальность записи. Примером такого ключа может служить поле "Номер паспорта", поскольку каждый такой номер является единственным в своем роде. Однако дата рождения уже не уникальна, поэтому соответствующее поле не может выступать в качестве первичного ключа.

Если подходящих столбцов для естественного задания первичного ключа не находится, пользуются суррогатным ключом. Суррогатный ключ представляет собой дополнительное поле в базе данных, предназначенное для обеспечения записей первичным ключом (именно такой подход принят на рис. 27.2).

> ***Примечание***
>
> Даже если в базе данных содержится естественный первичный ключ, лучше использовать суррогатные ключи, поскольку их применение позволяет абстрагировать первичный ключ от реальных данных. Это облегчает работу с таблицами, поскольку суррогатные ключи не связаны ни с какими фактическими данными таблицы.

Как уже упоминалось, в реляционных базах данных таблицы практически всегда логически связаны друг с другом. Одним из предназначений первичных ключей и является однозначная организация такой связи.

Рассмотрим, например, уже упоминавшуюся базу данных электронного магазина (рис. 27.3). Каждая из таблиц имеет свой первичный ключ, значение которого уникально в пределах таблицы. Еще раз подчеркнем, что таблица не обязана содержать первичные ключи, но это очень желательно, если данные связаны друг с другом. Столбец `id_catalog` таблицы `products` принимает значения из столбца `id_catalog` таблицы `catalogs`. Благодаря такой связи мы можем выстроить иерархию товарных позиций и определить, к какому разделу они относятся. Поле `id_catalog` таблицы `products` называют внешним или вторичным ключом. Внешний ключ сокращенно обозначают как FK (foreign key).

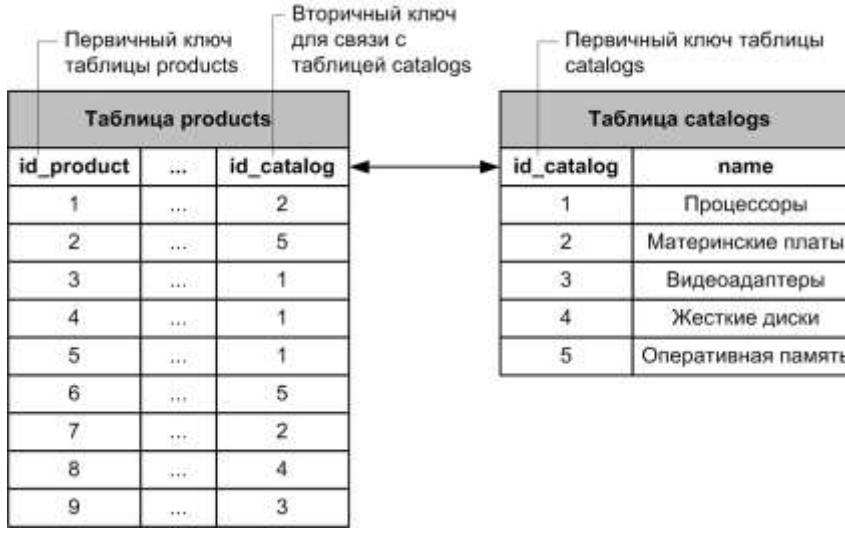


Рис. 27.3. Связь между таблицами `products` и `catalogs`

27.3. Создание и удаление базы данных

В СУБД MySQL создание базы данных сводится к созданию нового подкаталога в каталоге данных. По умолчанию для пользователей операционной системы Windows это каталог C:\mysql5\data, для пользователей RedHat-ориентированных дистрибутивов Linux — /var/lib/mysql. Далее, упоминая путь C:\mysql5\data, мы будем иметь в виду каталог данных, который, в общем случае, не обязательно должен располагаться в каталоге C:\mysql5\data. Создание базы данных средствами SQL осуществляется при помощи оператора `CREATE DATABASE`. В листинге 27.1 приведен пример создания базы данных `wet`.

Листинг 27.1. Создание базы данных `wet`

```
CREATE DATABASE wet;
```

> ***Замечание***
>
> В этом и последующих листингах жирным шрифтом отображается SQL-запрос, вводимый пользователем, а обычным — ответ сервера.

После выполнения запроса из листинга 27.1 можно обнаружить в каталоге C:\mysql5\data новый каталог wet.

> ***Замечание***
>
> Начиная с версии 4.1.1, в каталоге базы данных создается также файл db.opt, в котором указывается кодировка, используемая в базе данных по умолчанию `default-character-set`, и порядок сортировки символов `default-collation`.

Проконтролировать создание базы данных, а также узнать имена существующих баз данных можно при помощи оператора `SHOW DATABASES` (листинг 27.2).

Листинг 27.2. Использование оператора `SHOW DATABASES`

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
| wet                |
+--------------------+
```

Как видно из листинга 27.2, оператор `SHOW DATABASES` вернул имена четырех баз данных. Базы данных `information_schema` и `mysql` являются служебными и необходимы для поддержания сервера MySQL в работоспособном состоянии — в них хранится информация об учетных записях, региональных настройках и т. п.

> ***Замечание***
>
> База данных `mysql` является реальной базой данных, а `information_schema` — виртуальной, именно поэтому в каталоге данных нет подкаталога с соответствующим именем.

Можно создать в каталоге C:\mysql5\data\ новый каталог и выполнить после этого запрос `SHOW DATABASES` — созданная таким образом база данных будет отображена в списке баз данных. Удаление каталога приведет к исчезновению базы данных.

Удаление базы данных можно осуществить и штатными средствами при помощи оператора `DROP DATABASE`, за которым следует имя базы данных (листинг 27.3).

Листинг 27.3. Удаление базы данных `wet`

```
DROP DATABASE wet;
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

После выполнения оператора `DROP DATABASE` можно убедиться, что из каталога данных C:\mysql5\data\ был удален подкаталог wet.

Если производится попытка создания базы данных с уже существующим именем, возвращается ошибка (листинг 27.4).

Замечание

Так как имя базы данных — это имя каталога, то чувствительность к регистру определяется конкретной операционной системой. Так в операционной системе Windows имена `wet` и `Wet` будут обозначать одну и ту же базу данных, в то время как в UNIX-подобной операционной системе это будут две разные базы данных.

Листинг 27.4. Невозможно создать базу данных `wet`: база данных существует

```
CREATE DATABASE wet;
Query OK, 1 row affected (0.00 sec)
CREATE DATABASE wet;
ERROR 1007: Can't create database 'wet'; database exists
```

При создании базы данных можно указать кодировку, которая будет назначаться таблицам и столбцам по умолчанию. Для этого после имени базы данных следует указать конструкцию `DEFAULT CHARACTER SET` *`charset_name`*, где *`charset_name`* — имя кодировки, например, `cp1251`, которая обозначает русскую Windows-кодировку (листинг 27.5).

Листинг 27.5. Назначение кодировки по умолчанию

```
CREATE DATABASE wet DEFAULT CHARACTER SET cp1251;
```

Указание кодировки приводит к созданию в каталоге базы данных C:\mysql5\data\wet\ файла db.opt следующего содержания:

```
default-character-set=cp1251
default-collation=cp1251_general_ci
```

27.4. Выбор базы данных

Перед началом работы следует выбрать базу данных, к таблицам которой будет осуществляться обращение. Каждая клиентская программа решает эту задачу по-своему. Например, в консольном клиенте `mysql` выбрать новую базу данных можно при помощи команды `USE`. В листинге 27.6 при помощи команды `USE` в качестве текущей выбирается база данных `test`.

Листинг 27.6. Использование команды `USE`

```
USE test;
```

Если текущая база данных не выбрана, то обращение по умолчанию будет заканчиваться сообщением об ошибке "No database selected" ("База данных не выбрана") (листинг 27.7).

ЗАМЕЧАНИЕ

Оператор `SHOW TABLES` выводит список таблиц в текущей базе данных.

Листинг 27.7. База данных не выбрана

```
SHOW TABLES;
ERROR 1046 (3D000): No database selected
```

Можно не выбирать текущую базу данных, однако в этом случае во всех операторах придется ее указывать явно (листинг 27.8).

Листинг 27.8. Явное указание базы данных в операторе SHOW TABLES

```
SHOW TABLES FROM test;
```

При обращении к таблицам потребуется явное указание префикса базы данных, например, в листинге 27.9 из таблицы `User` базы данных `mysql` извлекаются поля `User` и `Host`.

Листинг 27.9. Явное указание базы данных в операторе SELECT

```
SELECT mysql.User.User, mysql.User.Host FROM mysql.User;
```

В результате явного обращения к базам данных SQL-запросы становятся достаточно громоздкими и менее гибкими, т. к. смена имени базы данных требует изменения SQL-запроса.

В программах, обращающихся к MySQL-серверу, для выбора базы данных используется функция `mysql_select_db()` (листинг 27.10).

ЗАМЕЧАНИЕ

Более подробно каждая из функций листинга 27.10 обсуждается в следующих разделах данной главы.

Листинг 27.10. Выбор базы данных при помощи функции

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
```

```
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
    exit("<p>В настоящий момент сервер базы данных не доступен,".
         "поэтому корректное отображение страницы невозможно.</p>");
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit("<p>В настоящий момент база данных не доступна,".
         "поэтому корректное отображение страницы невозможно.</p>");
  }

  // Настраиваем кодировку соединения
  @mysql_query("SET NAMES 'cp1251'");
?>
```

27.5. Типы данных

Как сообщалось ранее, таблицы состоят из столбцов и строк. Тип данных поля определяется столбцом. Список наиболее часто встречающихся типов приведен в табл. 27.1—27.3.

К *числовым типам* относятся целые числа и числа с плавающей точкой. Для чисел с плавающей точкой кроме максимальной ширины отображения можно также указывать число значащих цифр после точки, далее обозначаемое символом `P`.

> ***ЗАМЕЧАНИЕ***
>
> Здесь и далее необязательные элементы синтаксиса будут заключаться в квадратные скобки. Так, запись `TINYINT[(M)]` означает, что элемент может быть записан и как `TINYINT`, и как `TINYINT(M)`.

Таблица 27.1. *Числовые типы*

Тип	Объем занимаемой памяти	Диапазон допустимых значений
`TINYINT[(M)]`	1 байт	От –128 до 127 (от -2^7 до $2^7 - 1$); от 0 до 255 (от 0 до $2^8 - 1$)
`SMALLINT[(M)]`	2 байта	От –32 768 до 32 767 (от -2^{15} до $2^{15} - 1$); от 0 до 65 535 (от 0 до $2^{16} - 1$)
`MEDIUMINT[(M)]`	3 байта	От –8 388 608 до 8 388 607 (от -2^{23} до $2^{23} - 1$); от 0 до 16 777 215 (от 0 до $2^{24} - 1$)

***Таблица 27.1** (окончание)*

Тип	Объем занимаемой памяти	Диапазон допустимых значений
`INT[(M)]`, `INTEGER[(M)]`	4 байта	От –2 147 683 648 до 2 147 683 648 (от -2^{31} до $2^{31}-1$); от 0 до 4 294 967 295 (от 0 до $2^{32}-1$)
`BIGINT[(M)]`	8 байт	От -2^{63} до $2^{63}-1$; от 0 до 2^{64}
`BIT[(M)]`	(`M`+7)/8 байт	От 1 до 64 бита, в зависимости от значения `M`
`BOOL`, `BOOLEAN`	1 байт	0 или 1
`DECIMAL[(M[,D])]`, `DEC[(M[,D)]]`, `NUMERUC[(M[,D)]]`	`M` + 2 байта	Повышенная точность; зависит от параметров `M` и `D`
`FLOAT[(M,D)]`	4 байта	Минимальное значение $\pm 1.175494351 \times 10^{-39}$; максимальное значение $\pm 3.402823466 \times 10^{38}$
`DOUBLE[(M,D)]`, `REAL[(M,D)]`, `DOUBLE PRECISION[(M,D)]`	8 байт	Минимальное значение $\pm 2.2250738585072014 \times 10^{-308}$; максимальное значение $\pm 1.797693134862315 \times 10^{308}$

Для целых типов данных можно ограничить допустимый диапазон только положительными значениями, указав при объявлении типа ключевое слово `UNSIGNED`. В этом случае элементам данного столбца нельзя будет присвоить отрицательные значения, а допустимый диапазон положительных значений удвоится. Так, тип `TINYINT` может принимать значения от –128 до 127, а `TINYINT UNSIGNED` — от 0 до 255.

При объявлении целого типа может задаваться количество отводимых под число символов `M` (от 1 до 255). При использовании этого необязательного параметра выводимые значения, содержащие меньшее количество символов, будут дополняться пробелами слева. При этом, однако, не накладываются ограничения ни на диапазон величин, ни на количество разрядов. Если количество символов выводимого числа превышает `M`, под столбец будет выделено больше символов.

Диапазон вещественных чисел помимо максимального значения имеет также минимальное значение, которое характеризует точность данного типа. Параметр `M` в табл. 27.1 задает количество символов, отводимых для отображения всего числа, а `D` — для его дробной части.

Числовые типы данных с плавающей точкой также могут иметь параметр `UNSIGNED`. Как и в целочисленных типах, этот атрибут предотвращает хранение в отмеченном столбце отрицательных величин, но, в отличие от целочисленных типов, максимальный интервал для величин столбца остается прежним.

Замечание

Приближенные числовые данные могут задаваться в обычной форме, например, 45.67 и в так называемой научной нотации, например 5.456E–02 или 4.674E+04. Символ E означает, что стоящее перед ним число следует умножить на 10 в степени, указанной после E. Приведенные в предыдущих примерах числа можно записать как 0.05456 и 46740. Такой формат введен для удобства записи, т. к. числа в 300 степени неудобно представлять в обычном десятичном виде.

При выборе столбцов для формирования структуры таблицы необходимо обращать внимание на размер, занимаемый тем или иным типом данных: если значения, размещаемые в базе данных, никогда не будут выходить за пределы 100, не следует выбирать тип больше `TINYINT`. Если же в полях столбца предполагается хранить только целочисленные данные, то применение атрибута `UNSIGNED` позволит увеличить диапазон допустимых значений в два раза.

СУБД MySQL предлагает 5 типов данных для столбцов, хранящих календарные даты и время: `DATE`, `DATETIME`, `TIME`, `TIMESTAMP` и `YEAR` (табл. 27.2). Тип `DATE` предназначен для хранения даты, `TIME` — для времени суток, а `TIMESTAMP` — для представления и даты и времени суток. Тип `TIMESTAMP` предназначен для представления даты и времени суток в виде количества секунд, прошедших с 1 января 1970 года. Тип данных `YEAR` позволяет хранить только год.

Таблица 27.2. *Календарные типы данных*

Тип	Объем памяти	Диапазон
`DATE`	3 байта	От '1000-01-01' до '9999-12-31'
`TIME`	3 байта	От '–828:59:59' до '828:59:59'
`DATATIME`	8 байт	От '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
`TIMESTAMP [(M)]`	4 байта	От '1970-01-01 00:00:00' до '2038-12-31 59:59:59'
`YEAR [(M)]`	1 байт	От 1901 до 2155 для `YEAR(4)`; от 1970 до 2069 для `YEAR(2)`

Для значений, имеющих тип `DATE` и `DATETIME`, в качестве первого числа ожидается год либо в формате YYYY (например, '2005-10-15'), либо в формате YY (например, '05-10-15'). После года через дефис указывается месяц в формате MM (10), а затем число в формате DD (15).

В типах `TIME` и `DATETIME` время приводится в привычном формате hh:mm:ss, где hh — часы, mm — минуты, а ss — секунды. Дни, месяцы, часы, минуты и секунды можно записывать как с предваряющим нулем (01), так и без него (1). Все следующие записи идентичны:

```
'2005-04-06 02:04:08'
'2005-4-06 02:04:08'
```

```
'2005-4-6 02:04:08'
'2005-4-6 2:04:08'
'2005-4-6 2:4:08'
'2005-4-6 2:4:8'
```

В качестве разделителя между годами, месяцами, днями, часами, минутами и секундами может выступать любой символ, отличный от цифры. Так, следующие значения идентичны:

```
'05-12-31 11:30:45'
'05.12.31 11+30+45'
'05/12/31 11*30*45'
'05@12@31 11^30^45'
```

Дата и время суток могут также быть представлены в форматах 'YYYYMMDDhhmmss' и 'YYMMDDhhmmss'. Например, строки '20050523091528' и '050523091528' аналогичны '2005-05-23 09:15:28', однако строка '051122129015' уже не может рассматриваться как дата и время суток, т. к. ожидаемое для минут значение равно 90 и выходит за допустимый интервал. Вместо строк допустимы и целочисленные значения, например, 20050523091528 и 050523091528 рассматриваются как '2005-05-23 09:15:28'.

Начиная с версии MySQL 4.1.1, при указании времени суток допустимо указание после секунд через точку также микросекунд: 'hh:mm:ss.ffffff' (например, '10:25:14.000001'). Кроме расширенного формата можно использовать краткие форматы 'HH:MM' и 'HH': вместо не указанных величин будут подставлены нулевые значения.

Строковые типы данных, максимальный размер и требования к памяти приведены в табл. 27.3.

Таблица 27.3. *Строковые типы данных*

Тип	Объем занимаемой памяти, байт	Максимальный размер строки
`CHAR(M)`	`M`	`M` символов
`VARCHAR(M)`	`L`+1	`M` символов
`TINYBLOB`, `TINYTEXT`	`L`+1	2^8 – 1 символов
`BLOB`, `TEXT`	`L`+2	2^{16} – 1 символов
`MEDIUMBLOB`, `MEDIUMTEXT`	`L`+3	2^{24} – 1 символов
`LONGBLOB`, `LONGTEXT`	`L`+4	2^{32} – 1 символов
`ENUM('value1', 'value2',...)`	1 или 2	65 535 элементов
`SET('value1', 'value2',...)`	1, 2, 3, 4 или 8	64 элемента

Как видно из табл. 27.3, тип CHAR позволяет хранить строку фиксированной длины M — от 0 до 65 535. Его дополняет тип VARCHAR, позволяющий хранить переменные строки длиной L; при этом под значение ячейки отводится L+n байт, где n байт предназначены для хранения длины строки.

При выборе для столбца строкового типа данных следует принимать во внимание, что для хранения переменных строк VARCHAR требуется количество байтов, равное количеству символов в строке плюс один байт, в то время как тип CHAR(M), независимо от длины строки, использует для ее хранения все M байт. В то же время тип CHAR обрабатывается эффективнее переменных типов, т. к. всегда заранее известно, где заканчивается очередной блок данных (табл. 27.4).

***Таблица 27.4.** Сравнение типов CHAR и VARCHAR*

Значение	CHAR(4)		VARCHAR(4)	
	Представление	Занимаемый объем памяти, байт	Представление	Занимаемый объем памяти, байт
''	' '	4	''	1
'ab'	' ab'	4	'ab'	3
'abcd'	'abcd'	4	'abcd'	5
'abcdefgh'	'abcd'	4	'abcd'	5

В одной таблице не допускается одновременное создание столбцов типа CHAR и VARCHAR. В противном случае СУБД MySQL изменит тип столбцов согласно правилу: если в таблице присутствует хоть один столбец переменной длины, все столбцы типа CHAR приводятся к типу VARCHAR.

Типы BLOB и TEXT в СУБД MySQL во всем аналогичны и отличаются только в деталях. Например, при выполнении операций над столбцами типа TEXT кодировка учитывается, а для столбцов типа BLOB — нет.

Тип TEXT обычно используется для хранения больших объемов текста, в то время как BLOB — для больших двоичных объектов, таких как электронные документы, изображения, звуки и т. д.

К особым типам данных относятся ENUM и SET. Строки этих типов принимают значения из заранее заданного списка допустимых значений. Основное различие между ними заключается в том, что значение типа ENUM должно содержать точно один элемент из указанного множества, тогда как столбцы SET могут содержать любой набор элементов одновременно. Так, ячейки столбца, для которого объявлен тип данных ENUM('y','n'), могут принимать только два значения: либо 'y', либо 'n'. Ячейки столбца с типом данных SET с тем же набором допустимых значений могут принимать значения ('y','n'), ('y'), ('n') и пустое множество (), где пустое множество означает, что не выбран ни один из элементов.

Типы ENUM и SET можно назвать строковыми лишь отчасти, т. к. при объявлении они задаются списком строк, но во внутреннем представлении базы данных элементы множеств сохраняются в виде чисел.

Элементы типа ENUM нумеруются последовательно, начиная с 1. В зависимости от количества элементов в списке под хранение ячеек столбца может отводиться 1 байт (до 256 элементов) или 2 байта (от 257 до 65 536).

Элементы множества SET обрабатываются как биты, размер типа при этом также определяется количеством элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16), 3 байта (от 17 до 24), 4 байта (от 25 до 32) или 8 байт (от 33 до 64).

27.6. Создание и удаление таблиц

Оператор CREATE TABLE создает новую таблицу в выбранной базе данных и в простейшем случае имеет следующий синтаксис:

```
CREATE TABLE table_name [(create_definition, ...)]
```

Здесь *table_name* — имя создаваемой таблицы.

Создадим первую таблицу базы данных forum, которая называется authors и содержит различные данные о зарегистрированных посетителях форума: ник (name); пароль (passw); e-mail (email); Web-адрес сайта посетителя (url); номер ICQ (icq); сведения о посетителе (about); строку, содержащую путь к файлу фотографии посетителя (photo); время добавления запроса (putdate); последнее время посещения форума (last_time); статус посетителя (statususer) — является ли он модератором ('moderator'), администратором ('admin') или обычным посетителем ('user'). Кроме перечисленных полей в таблице имеется поле id_author, представляющее собой первичный ключ таблицы.

SQL-запрос, создающий таблицу authors, приведен в листинге 27.11. Обратите внимание, что поля, в которых хранятся короткие строки (имя пользователя, его пароль, e-mail, ICQ и т. п.), имеют тип данных TINYTEXT, который позволяет вводить не более 256 символов, тогда как под адрес домашней страницы и сообщение о себе выделяется поле с типом данных TEXT, которое может содержать уже 65 536 символов.

> ***ЗАМЕЧАНИЕ***
>
> Не следует экономить, поскольку разница между TEXT и TINYTEXT составляет всего один байт. Так как эти поля обладают плавающим размером, информация не будет занимать лишней памяти.

Листинг 27.11. Создание таблицы authors базы данных forum

```
CREATE TABLE authors (
   id_author INT(11) NOT NULL AUTO_INCREMENT,
   name TINYTEXT,
   passw TINYTEXT,
   email TINYTEXT,
   url TEXT,
```

```
  icq TINYTEXT,
  about TEXT,
  photo TINYTEXT,
  putdate DATETIME DEFAULT NULL,
  last_time DATETIME DEFAULT NULL,
  themes INT(10) DEFAULT NULL,
  statususer ENUM('user','moderator','admin') NOT NULL default 'user',
  PRIMARY KEY (id_author)
);
```

Выполнив SQL-команду `SHOW TABLES`, можно убедиться, что таблица `authors` успешно создана (листинг 27.12).

Листинг 27.12. Использование оператора `SHOW TABLES`

```
SHOW TABLES;
+-----------------+
| Tables_in_forum |
+-----------------+
| authors         |
+-----------------+
```

Аналогичным образом создадим еще несколько необходимых для работы форума таблиц. Следующей по порядку идет таблица `forums`, в которой содержатся данные о разделах форума.

> ***Примечание***
>
> Для удобства на форуме может быть создано несколько различных разделов. К примеру, на форуме, посвященном языкам программирования, чтобы не смешивать темы, относящиеся к различным языкам, имеет смысл создать следующие разделы: C++, PHP, Java и т. д.

В таблице `forums` присутствуют следующие поля: первичный ключ (`id_forum`); название раздела (`name`); правила форума (`rule`); краткое описание форума (`logo`); порядковый номер (`pos`) и флаг (`hide`), принимающий значение `'hide'`, если форум скрытый, и `'show'`, если он общедоступен.

SQL-запрос, создающий таблицу `forums`, приведен в листинге 27.13.

Листинг 27.13. Создание таблицы `forums`

```
CREATE TABLE forums (
  id_forum INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT,
  rule TEXT,
  logo TINYTEXT,
```

```
  pos INT(11) DEFAULT NULL,
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  PRIMARY KEY (id_forum)
);
```

Структура форума может быть следующей: имеется список разделов, переход по которому приводит посетителя к списку тем раздела. При переходе по теме посетитель приходит к обсуждению этой темы, состоящему из сообщений других посетителей. Теперь создадим таблицу `themes`, содержащую темы форума (листинг 27.14).

Листинг 27.14. Создание таблицы `themes`

```
CREATE TABLE themes (
  id_theme INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT,
  author TINYTEXT,
  id_author INT(11) DEFAULT NULL,
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  putdate DATETIME DEFAULT NULL,
  id_forum INT(11) default NULL,
  PRIMARY KEY (id_theme)
);
```

В таблице `themes` присутствуют следующие поля: первичный ключ (`id_theme`); название темы (`name`); автор темы (`author`); внешний ключ к таблице авторов (`id_author`); флаг (`hide`), принимающий значение `'hide'`, если тема скрыта, и `'show'`, если она отображается (это поле необходимо для модерирования); время добавления темы (`putdate`); внешний ключ к таблице форумов (`id_forum`), для того чтобы определить, к какому разделу форума относится данная тема.

Таблица `themes` нормализована только частично; она содержит два внешних ключа: `id_author` и `id_forum` для таблиц посетителей и списка форумов, в то же время в ней дублируется имя автора `author`, присутствующее также в таблице посетителей `authors` под именем `name`. Этот случай является примером умышленной денормализации, необходимой для избежания запроса таблицы авторов всякий раз при выводе списка тем и их авторов, что позволяет обеспечить приемлемую скорость работы форума.

Создадим последнюю таблицу `posts`, в которой будут храниться сообщения (листинг 27.15).

Листинг 27.15. Создание таблицы `posts`

```
CREATE TABLE posts (
  id_post INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT,
```

```
  url TEXT,
  file TINYTEXT,
  author TINYTEXT,
  id_author INT(11) DEFAULT NULL,
  hide ENUM('show','hide') NOT NULL DEFAULT 'show',
  putdate DATETIME DEFAULT NULL,
  parent_post INT(11) DEFAULT NULL,
  id_theme INT(11) DEFAULT NULL,
  PRIMARY KEY (id_post)
);
```

В таблице `posts` присутствуют следующие поля: первичный ключ (`id_post`); тело сообщения (`name`); необязательная ссылка на ресурс, которую автор сообщения может ввести при добавлении сообщения (`url`); путь к файлу, прикрепляемому к сообщению (`file`); имя автора (`author`); внешний ключ к таблице авторов (`id_author`); флаг (`hide`), принимающий значение `'hide'`, если сообщение скрытое, и `'show'`, если он отображается (это поле необходимо для модерирования); время добавления сообщения (`putdate`); сообщение, ответом на которое является данное сообщение (`parent_post`): если это первое сообщение в теме, то поле равно 0; внешний ключ к теме (`id_theme`), указывающий, к какой теме относится сообщение.

Убедимся, что все таблицы успешно созданы, выполнив команду `SHOW TABLES`. Результат показан в листинге 27.16.

Листинг 27.16. Список таблиц базы данных `forum`

```
SHOW TABLES;
+-----------------+
| Tables_in_forum |
+-----------------+
| authors         |
| forums          |
| posts           |
| themes          |
+-----------------+
```

Оператор `DESCRIBE` показывает структуру созданных таблиц и имеет следующий синтаксис:

```
DESCRIBE table_name
```

Здесь *`table_name`* — имя таблицы, структура которой запрашивается.

> ***ЗАМЕЧАНИЕ***
>
> Оператор `DESCRIBE` не входит в стандарт SQL и является внутренней командой СУБД MySQL.

Просмотреть структуру таблицы `forums` можно, выполнив SQL-запрос, приведенный в листинге 27.17.

Листинг 27.17. Структура таблицы `fourms`

```
mysql> DESCRIBE forums;
+----------+---------------------+------+-----+---------+---------------+
| Field    | Type                | Null | Key | Default | Extra         |
+----------+---------------------+------+-----+---------+---------------+
| id_forum | int(11)             | NO   | PRI | NULL    | auto_increment|
| name     | tinytext            | YES  |     | NULL    |               |
| rule     | text                | YES  |     | NULL    |               |
| logo     | tinytext            | YES  |     | NULL    |               |
| pos      | int(11)             | YES  |     | NULL    |               |
| hide     | enum('show','hide') | NO   |     | show    |               |
+----------+---------------------+------+-----+---------+---------------+
```

Изменить структуру таблицы позволяет оператор `ALTER TABLE`. С его помощью можно добавлять и удалять столбцы, создавать и уничтожать индексы, переименовывать столбцы и саму таблицу. Оператор имеет следующий синтаксис:

```
ALTER TABLE table_name alter_spec
```

Наиболее часто используемые значения параметра *`alter_spec`* приводятся в табл. 27.5.

***Таблица 27.5.** Основные преобразования, выполняемые оператором `ALTER TABLE`*

Синтаксис	Описание команды
`ADD create_definition [FIRST|AFTER column_name]`	Добавляет новый столбец. *`create_definition`* представляет собой название нового столбца и его тип. Конструкция `FIRST` добавляет новый столбец перед столбцом *`column_name`*; конструкция `AFTER` — после него. Если место добавления не указано, столбец добавляется в конец таблицы
`ADD INDEX [index_name] (index_col_name,...)`	Добавляет индекс *`index_name`* для столбца *`index_col_name`*. Если имя индекса `index_name` не указывается, ему присваивается имя, совпадающее с именем столбца *`index_col_name`*
`ADD PRIMARY KEY (index_col_name,...)`	Делает столбец *`index_col_name`* или группу столбцов первичным ключом таблицы
`CHANGE old_col_name new_col_name type`	Заменяет столбец с именем *`old_col_name`* на столбец с именем *`new_col_name`* и типом *`type`*
`DROP col_name`	Удаляет столбец с именем *`col_name`*
`DROP PRIMARY KEY`	Удаляет первичный ключ таблицы
`DROP INDEX index_name`	Удаляет индекс *`index_name`*

Добавим в таблицу `forums` новый столбец `test` (листинг 27.18), разместив его после столбца `name`.

Листинг 27.18. Добавление столбца в таблицу

```
ALTER TABLE forums ADD test INT(10) AFTER name;
DESCRIBE forums;
+----------+---------------------+------+-----+---------+---------------+
| Field    | Type                | Null | Key | Default | Extra         |
+----------+---------------------+------+-----+---------+---------------+
| id_forum | int(11)             | NO   | PRI | NULL    | auto_increment|
| name     | tinytext            | YES  |     | NULL    |               |
| test     | int(10)             | YES  |     | NULL    |               |
| rule     | text                | YES  |     | NULL    |               |
| logo     | tinytext            | YES  |     | NULL    |               |
| pos      | int(11)             | YES  |     | NULL    |               |
| hide     | enum('show','hide') | NO   |     | show    |               |
+----------+---------------------+------+-----+---------+---------------+
```

Переименуем созданный столбец `test` в текстовый столбец `new_test` (листинг 27.19).

Листинг 27.19. Переименование столбца

```
ALTER TABLE forums CHANGE test new_test TEXT;
DESCRIBE forums;
+----------+---------------------+------+-----+---------+---------------+
| Field    | Type                | Null | Key | Default | Extra         |
+----------+---------------------+------+-----+---------+---------------+
| id_forum | int(11)             | NO   | PRI | NULL    | auto_increment|
| name     | tinytext            | YES  |     | NULL    |               |
| new_test | text                | YES  |     | NULL    |               |
| rule     | text                | YES  |     | NULL    |               |
| logo     | tinytext            | YES  |     | NULL    |               |
| pos      | int(11)             | YES  |     | NULL    |               |
| hide     | enum('show','hide') | NO   |     | show    |               |
+----------+---------------------+------+-----+---------+---------------+
```

При изменении только типа столбца указание имени все равно необходимо, хотя в этом случае оно будет фактически повторяться (листинг 27.20).

Листинг 27.20. Изменение типа столбца

```
ALTER TABLE forums CHANGE new_test new_test INT(5) NOT NULL;
DESCRIBE forums;
+----------+---------------------+------+-----+---------+---------------+
| Field    | Type                | Null | Key | Default | Extra         |
+----------+---------------------+------+-----+---------+---------------+
| id_forum | int(11)             | NO   | PRI | NULL    | auto_increment|
| name     | tinytext            | YES  |     | NULL    |               |
| new_test | int(5)              | NO   |     |         |               |
| rule     | text                | YES  |     | NULL    |               |
| logo     | tinytext            | YES  |     | NULL    |               |
| pos      | int(11)             | YES  |     | NULL    |               |
| hide     | enum('show','hide') | NO   |     | show    |               |
+----------+---------------------+------+-----+---------+---------------+
```

Теперь удалим столбец `new_test` (листинг 27.21).

Листинг 27.21. Удаление столбца из таблицы

```
ALTER TABLE forums DROP new_test;
DESCRIBE forums;
+----------+---------------------+------+-----+---------+---------------+
| Field    | Type                | Null | Key | Default | Extra         |
+----------+---------------------+------+-----+---------+---------------+
| id_forum | int(11)             | NO   | PRI | NULL    | auto_increment|
| name     | tinytext            | YES  |     | NULL    |               |
| rule     | text                | YES  |     | NULL    |               |
| logo     | tinytext            | YES  |     | NULL    |               |
| pos      | int(11)             | YES  |     | NULL    |               |
| hide     | enum('show','hide') | NO   |     | show    |               |
+----------+---------------------+------+-----+---------+---------------+
```

Оператор `DROP TABLE` предназначен для удаления одной или нескольких таблиц:

```
DROP TABLE table_name [ , table_name,...]
```

Например, для удаления таблицы `forums` необходимо выполнить SQL-запрос, представленный в листинге 27.22.

Листинг 27.22. Удаление таблицы `forums`

```
mysql> DROP TABLE forums;
```

27.7. Вставка числовых значений в таблицу

Для вставки записи в таблицу предназначен оператор INSERT. Однострочный оператор INSERT может использоваться в нескольких формах. Упрощенный синтаксис первой формы выглядит следующим образом:

```
INSERT [IGNORE] [INTO] tbl [(col_name,...)] VALUES (expression,...)
```

> ***Замечание***
>
> При описании синтаксиса операторов ключевые слова, которые не являются обязательными и могут быть опущены, заключаются в квадратные скобки.

Создадим таблицу tbl, состоящую из двух числовых столбцов: id, который по умолчанию будет иметь значение 5, и id_cat, который будет принимать по умолчанию значение NULL (листинг 27.23).

Листинг 27.23. Таблица tbl

```
CREATE TABLE tbl (
  id INT(11) NOT NULL DEFAULT '5',
  id_cat INT(11) DEFAULT NULL
);
```

Существует несколько вариантов использования оператора INSERT, каждый из которых приводит к вставке новой записи (листинг 27.24).

Листинг 27.24. Вставка записей при помощи оператора INSERT

```
INSERT INTO tbl VALUES (10, 20);
INSERT INTO tbl (id_cat, id) VALUES (10, 20);
INSERT INTO tbl (id) VALUES (30);
INSERT INTO tbl () VALUES ();
```

Проверить результат вставки новых записей в таблицу можно при помощи оператора SELECT (листинг 27.25), синтаксис которого подробно разбирается далее в этой главе.

Листинг 27.25. Просмотр содержимого таблицы

```
SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
| 10 |     20 |
| 20 |     10 |
| 30 |   NULL |
|  5 |   NULL |
+----+--------+
```

Рассмотрим различные формы оператора INSERT из листинга 27.24 более подробно. Первая форма оператора INSERT из листинга 27.24 вставляет в таблицу tbl запись (10, 20), столбцы получают значения по порядку из круглых скобок, следующих за ключевым словом VALUES. Если значений в круглых скобках будет больше или меньше, чем столбцов в таблице, то сервер MySQL вернет ошибку "Column count doesn't match value count at row 1" ("Не совпадает количество значений и столбцов в запросе").

Порядок занесения значений в запись можно изменять. Для этого надо задать порядок следования столбцов в дополнительных круглых скобках после имени таблицы. В листинге 27.24 второй оператор INSERT меняет порядок занесения значений: первое значение получает второй столбец id_cat, а второе значение — первый столбец id.

Часть столбцов можно опускать из списка — в этом случае они получают значение по умолчанию. В листинге 27.24 в третьем операторе заполняется лишь поле id, при этом поле id_cat получает значение по умолчанию — NULL. Четвертый оператор вообще не содержит значений, в этом случае все столбцы таблицы tbl получат значения по умолчанию, которые определяет ключевое слово DEFAULT (листинг 27.23). Эффекта последнего оператора можно добиться, если использовать вместо значений ключевое слово DEFAULT. В листинге 27.26 оба оператора эквивалентны.

Листинг 27.26. Использование ключевого слова DEFAULT

```
INSERT INTO tbl () VALUES ();
INSERT INTO tbl (id, id_cat) VALUES (DEFAULT, DEFAULT);
```

Приведенная в листинге 2.24 форма оператора INSERT является стандартной и поддерживается всеми СУБД, которые реализуют стандарт SQL. Однако СУБД MySQL поддерживает альтернативный синтаксис оператора INSERT (листинг 27.27).

Листинг 27.27. Альтернативный синтаксис оператора INSERT

```
INSERT INTO tbl SET id = 40, id_cat = 50;
INSERT INTO tbl SET id = 50;
```

Как видно из листинга 27.27, значения присваиваются столбцам при помощи ключевого слова SET; не указанные в операторе столбцы принимают значение по умолчанию.

27.8. Вставка строковых значений в таблицу

Создадим таблицу catalogs, которая будет содержать два столбца: числовой столбец id_catalog и текстовый столбец name (листинг 27.28).

Листинг 27.28. Создание таблицы catalogs

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
);
```

Добавить новую запись в таблицу `catalogs` можно при помощи запроса, представленного в листинге 27.29.

Листинг 27.29. Добавление новой записи в таблицу `catalogs`

```
INSERT INTO catalogs VALUES (1, 'Процессоры');
SELECT * FROM catalogs;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 27.29, в таблицу `catalogs` добавилась новая запись с первичным ключом `id_catalogs`, равным единице, и значением поля `name` — "Процессоры". Строковые значения необходимо помещать в кавычки, в то время как числовые значения допускается использовать без них.

Вместо одиночных кавычек можно использовать двойные кавычки (листинг 27.30). Когда в текстовое поле необходимо вставить строку, содержащую двойные кавычки, можно использовать для обрамления строки одиночные кавычки и наоборот.

Листинг 27.30. Использование двойных кавычек

```
INSERT INTO catalogs VALUES (2, "Память");
SELECT * FROM catalogs;
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
|          2 | Память     |
+------------+------------+
```

В некоторых случаях (например, при использовании одинаковых кавычек в качестве обрамляющих и внутри вводимой строки) следует прибегнуть к экранированию внутренних кавычек, т. е. помещению символа \ перед кавычками, включенными в строку (листинг 27.31).

ЗАМЕЧАНИЕ

При помещении в базу данных текста, набранного пользователем, всегда следует экранировать кавычки, для того чтобы предотвратить возникновение ошибки и атаку SQL-инъекцией.

Листинг 27.31. Экранирование кавычек

```
INSERT INTO catalogs VALUES (3, "Память \"DDR\"");
SELECT * FROM catalogs;
+------------+--------------+
| id_catalog | name         |
+------------+--------------+
|          1 | Процессоры   |
|          2 | Память       |
|          3 | Память "DDR" |
+------------+--------------+
```

27.9. Вставка календарных значений

В общем случае вставка календарных значений мало чем отличается от вставки строк, однако этот тип данных не случайно выделяют в отдельный класс. Создадим таблицу `tbl`, которая будет содержать числовое поле `id` и два календарных поля: `putdate` типа `DATETIME` и `lastdate` типа `DATE` (листинг 27.32).

Листинг 27.32. Создание таблицы `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  putdate DATETIME NOT NULL,
  lastdate DATE NOT NULL
);
```

В листинге 27.33 приводится пример вставки записи в таблицу, содержащую столбцы календарного типа.

Листинг 27.33. Вставка календарных значений

```
INSERT INTO tbl VALUES (1, '2009-01-03 0:00:00', '2009-01-03');
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2009-01-03 00:00:00 | 2009-01-03 |
+----+---------------------+------------+
```

Если столбцу передается "лишняя" информация, то она отбрасывается. В листинге 27.34 в столбец типа `DATE` передаются часы, минуты и секунды, однако в столбец попадает только информация о дате.

Листинг 27.34. Отбрасывание "лишней" информации

```
INSERT INTO tbl VALUES (2, '2009-01-03 0:00:00', '2009-01-03 0:00:00');
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2009-01-03 00:00:00 | 2009-01-03 |
|  2 | 2009-01-03 00:00:00 | 2009-01-03 |
+----+---------------------+------------+
```

Зачастую календарные поля предназначены для того, чтобы пометить момент вставки записи в базу данных. Для получения текущего времени удобно воспользоваться встроенной функцией MySQL — `NOW()`. В листинге 27.35 при помощи функции `NOW()` в таблицу `tbl` вставляется запись с текущей временной меткой.

Листинг 27.35. Использование функции `NOW()`

```
INSERT INTO tbl VALUES (3, NOW(), NOW());
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2009-01-03 00:00:00 | 2009-01-03 |
|  2 | 2009-01-03 00:00:00 | 2009-01-03 |
|  3 | 2009-01-03 15:03:33 | 2009-01-03 |
+----+---------------------+------------+
```

Вычисление текущего времени в рамках одного SQL-запроса производится только один раз, сколько бы раз они не вызывались на протяжении данного запроса. Это приводит к тому, что временное значение в рамках всего запроса остается постоянным.

Для получения сдвига даты относительно текущей можно прибавлять и вычитать интервалы. Для этого используется ключевое слово `INTERVAL`, после которого следует временной интервал. В листинге 27.36 в поле `putdate` помещается дата, эквивалентная началу 2009 года за вычетом 3 недель, а в поле `lastdate` помещается дата, равная текущей плюс 3 месяца.

Листинг 27.36. Использование интервалов

```
INSERT INTO tbl
VALUES (4, '2009-01-01 0:00:00' - INTERVAL 3 WEEK,
           NOW() + INTERVAL 3 MONTH);
SELECT * FROM tbl;
```

```
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2009-01-03 00:00:00 | 2009-01-03 |
|  2 | 2009-01-03 00:00:00 | 2009-01-03 |
|  3 | 2009-01-03 15:03:33 | 2009-01-03 |
|  4 | 2008-12-11 00:00:00 | 2009-04-03 |
+----+---------------------+------------+
```

Как видно из листинга 27.36, интервалы могут быть различного типа. Полный список допустимых интервалов приводится в табл. 27.6.

Таблица 27.6. *Типы временных интервалов*

Тип	Описание	Формат ввода
MICROSECOND	Микросекунды	xxxxxx
SECOND	Секунды	ss
MINUTE	Минуты	mm
HOUR	Часы	hh
DAY	Дни	DD
WEEK	Недели	WW
MONTH	Месяцы	MM
QUARTER	Квартал	QQ
YEAR	Год	YY
SECOND_MICROSECOND	Секунды и микросекунды	'ss.xxxxxx'
MINUTE_MICROSECOND	Минуты, секунды и микро-секунды	'mm:ss.xxxxxx'
MINUTE_SECOND	Минуты и секунды	'mm:ss'
HOUR_MICROSECOND	Часы, минуты, секунды и микро-секунды	'hh:mm:ss.xxxxxx'
HOUR_SECOND	Часы, минуты и секунды	'hh:mm:ss'
HOUR_MINUTE	Часы и минуты	'hh:mm'
DAY_MICROSECOND	Дни, часы, минуты, секунды и микросекунды	'DD hh:mm:ss.xxxxxx'
DAY_SECOND	Дни, часы, минуты и секунды	'DD hh:mm:ss'
DAY_MINUTE	Дни, часы и минуты	'DD hh:mm'
DAY_HOUR	Дни и часы	'DD hh'
YEAR_MONTH	Года и месяцы	'YY-MM'

27.10. Вставка уникальных значений

Первичный ключ таблицы (PRIMARY KEY) или столбец, индексированный уникальным индексом (UNIQUE), не могут иметь повторяющихся значений. Вставка записи со значением, уже имеющимся в таблице, приводит к возникновению ошибки (листинг 27.37).

Листинг 27.37. Значения первичного ключа должны быть уникальными

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id)
);
INSERT INTO tbl VALUES (1, 'Видеоадаптеры');
INSERT INTO tbl VALUES (1, 'Видеоадаптеры');
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

Если необходимо, чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, следует добавить после оператора INSERT ключевое слово IGNORE (листинг 27.38).

Листинг 27.38. Использование ключевого слова IGNORE

```
INSERT IGNORE INTO tbl VALUES (1, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Видеоадаптеры |
+----+---------------+
```

Как видно из листинга 27.38, генерация ошибки не происходит, тем не менее, новая запись также не добавляется.

27.11. Механизм *AUTO_INCREMENT*

При добавлении новой записи с уникальными индексами выбор такого уникального значения может быть непростой задачей. Для того чтобы не осуществлять дополнительный запрос, направленный на выявление максимального значения первичного ключа для создания нового уникального значения, в MySQL введен механизм его автоматической генерации. Для этого достаточно снабдить первичный ключ атрибутом AUTO_INCREMENT, после чего при создании новой записи достаточно передать данному

столбцу в качестве значения NULL или 0 — поле автоматически получит значение, равное максимальному значению в столбце, плюс единица. В листинге 27.39 создается таблица tbl, состоящая из первичного ключа id и текстового поля name. Первичный ключ id снабжен атрибутом AUTO_INCREMENT.

Листинг 27.39. Действие механизма AUTO_INCREMENT

```
CREATE TABLE tbl (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id)
);
INSERT INTO tbl VALUES (NULL, 'Процессоры');
INSERT INTO tbl VALUES (NULL, 'Материнские платы');
INSERT INTO tbl VALUES (NULL, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+-------------------+
| id | name              |
+----+-------------------+
|  1 | Процессоры        |
|  2 | Материнские платы |
|  3 | Видеоадаптеры     |
+----+-------------------+
```

27.12. Многострочный оператор *INSERT*

Многострочный оператор INSERT совпадает по форме с однострочным оператором. В нем используется ключевое слово VALUES, после которого добавляется не один, а несколько списков *expression*. В листинге 27.40 добавляются сразу пять записей при помощи одного оператора INSERT.

> ***Замечание***
>
> Как и в однострочной версии оператора INSERT, допускается использование ключевого слова IGNORE для игнорирования записей, значения уникальных индексов которых совпадают с одним из уже имеющихся в таблице значений.

Листинг 27.40. Многострочный оператор INSERT

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
  PRIMARY KEY (id)
);
```

```
INSERT INTO catalogs VALUES (0, 'Процессоры'),
(0, 'Материнские платы'),
(0, 'Видеоадаптеры'),
(0, 'Жесткие диски'),
(0, 'Оперативная память');
```

Как и в случае однострочного варианта, допускается изменять порядок и состав списка добавляемых значений (листинг 27.41).

Листинг 27.41. Изменение состава столбцов в многострочном INSERT

```
INSERT INTO catalogs (name) VALUES ('Процессоры'),
('Материнские платы'),
('Видеоадаптеры'),
('Жесткие диски'),
('Оперативная память');
```

27.13. Удаление данных

Время от времени возникает задача удаления записей из базы данных, для которой предназначены следующие два оператора:

- DELETE — удаление всех или части записей из таблицы;
- TRUNCATE TABLE — удаление всех записей из таблицы.

Оператор DELETE имеет следующий синтаксис:

```
DELETE FROM tbl
WHERE where_definition
ORDER BY ...
LIMIT rows
```

Оператор удаляет из таблицы *tbl* записи, удовлетворяющие условию *where_definition*. В листинге 27.42 из таблицы catalogs удаляются записи, значение первичного ключа id которых больше двух.

Листинг 27.42. Удаление записей из таблицы catalogs

```
DELETE FROM catalogs WHERE id > 2;
SELECT * FROM catalogs;
+----+------------------+---------------------+
| id | name             | putdate             |
+----+------------------+---------------------+
|  1 | Процессоры       | 2005-01-19 21:40:30 |
|  2 | Материнские платы | 2005-02-19 10:30:30 |
+----+------------------+---------------------+
```

Если в операторе DELETE отсутствует условие WHERE, из таблицы удаляются все записи (листинг 27.43).

Листинг 27.43. Удаление всех записей таблицы catalogs

```
DELETE FROM catalogs;
SELECT * FROM catalogs;
Empty set (0.00 sec)
```

Применение ограничения LIMIT позволяет задать максимальное количество уничтожаемых записей. В листинге 27.44 удаляется не более 3 записей таблицы catalogs.

Листинг 27.44. Удаление записей из таблицы catalogs

```
DELETE FROM catalogs LIMIT 3;
```

Оператор TRUNCATE TABLE, в отличие от оператора DELETE, полностью очищает таблицу и не допускает условного удаления. То есть оператор TRUNCATE TABLE аналогичен оператору DELETE без условия WHERE и ограничения LIMIT. В отличие от оператора DELETE удаление происходит гораздо быстрее, т. к. при этом не выполняется перебор каждой записи. Пример использования оператора TRUNCATE TABLE приводится в листинге 27.45.

> ***Замечание***
>
> Оптимизатор запросов СУБД MySQL автоматически использует оператор TRUNCATE TABLE, если оператор DELETE не содержит WHERE-условия или конструкции LIMIT.

Листинг 27.45. Удаление всех записей таблицы products

```
TRUNCATE TABLE products;
```

27.14. Обновление записей

Операция обновления позволяет менять значения полей в уже существующих записях. Для обновления данных предназначены операторы UPDATE и REPLACE. Первый обновляет отдельные поля в уже существующих записях, тогда как оператор REPLACE больше похож на INSERT, за исключением того, что если старая запись в данной таблице имеет то же значение индекса UNIQUE или PRIMARY KEY, что и новая, то старая запись перед занесением новой записи будет удалена.

Оператор UPDATE имеет следующий синтаксис:

```
UPDATE [IGNORE] tbl
SET col1=expr1 [, col2=expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT rows]
```

Сразу после ключевого слова `UPDATE` в инструкции указывается таблица *`tbl`*, которая подвергается изменению. В предложении `SET` перечисляются столбцы, которые подвергаются обновлению, и устанавливаются их новые значения. Необязательное условие `WHERE` позволяет задать критерий отбора строк — обновлению будут подвергаться только те строки, которые удовлетворяют условию *`where_definition`*.

> ***ЗАМЕЧАНИЕ***
>
> Если в столбец с уникальными значениями (например, столбец с первичным ключом) осуществляется попытка вставки уже существующего значения, это обычно заканчивается сообщением об ошибке. Подавить генерацию ошибки этого типа позволяет использование ключевого слова `IGNORE` (обновление, которое бы привело к дублированию уникальных значений, не осуществляется и в этом случае).

Заменим название элемента каталога "Процессоры" в таблице `catalogs` на "Процессоры (Intel)" (листинг 27.46).

Листинг 27.46. Обновление таблицы `catalogs`

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
UPDATE catalogs SET name = 'Процессоры (Intel)'
WHERE name = 'Процессоры';
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Оператор `REPLACE` работает аналогично `INSERT`, за исключением того, что если старая запись в данной таблице имеет то же значение индекса `UNIQUE` или `PRIMARY KEY`, что и новая, то старая запись перед занесением новой будет удалена. Следует учитывать,

что если не используются индексы UNIQUE или PRIMARY KEY, то применение команды REPLACE не имеет смысла, т. к. ее действие при этом идентично команде INSERT.

Синтаксис оператора RENAME аналогичен синтаксису оператора INSERT:

```
REPLACE [INTO] tbl [(col_name,...)] VALUES (expression,...),(...),...
```

В таблицу *tbl* вставляются значения, определяемые в списке после ключевого слова VALUES. Задать порядок столбцов можно при помощи необязательного списка *col_name*, следующего за именем таблицы *tbl*. Как и в случае оператора INSERT, оператор REPLACE допускает многострочный формат. Пример использования оператора приведен в листинге 27.47, где в таблицу catalogs добавляются пять новых записей.

ЗАМЕЧАНИЕ

Многотабличный синтаксис для операторов REPLACE и INSERT не предусмотрен.

Листинг 27.47. Использование оператора REPLACE

```
SELECT * FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры (Intel) |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
REPLACE INTO catalogs VALUES
(4, 'Сетевые адаптеры'),
(5, 'Программное обеспечение'),
(6, 'Мониторы'),
(7, 'Периферия'),
(8, 'CD-RW/DVD');
SELECT * FROM catalogs ORDER BY id_catalog;
+------------+-------------------------+
| id_catalog | name                    |
+------------+-------------------------+
|          1 | Процессоры (Intel)      |
|          2 | Материнские платы       |
|          3 | Видеоадаптеры           |
|          4 | Сетевые адаптеры        |
|          5 | Программное обеспечение |
|          6 | Мониторы                |
|          7 | Периферия               |
|          8 | CD-RW/DVD               |
+------------+-------------------------+
```

27.15. Выборка данных

Рассмотрим запросы на выборку данных на примере таблицы `catalogs` (листинг 27.48). Таблица имеет два поля: первичный ключ `id_catalog` и название элемента каталога `name`.

Листинг 27.48. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL,
  name TINYTEXT NOT NULL
  PRIMARY KEY (id)
);
INSERT INTO catalogs VALUES (0, 'Процессоры'),
(0, 'Материнские платы'),
(0, 'Видеоадаптеры'),
(0, 'Жесткие диски'),
(0, 'Оперативная память');
```

Выбрать все записи таблицы `catalogs` можно при помощи запроса, представленного в листинге 27.49.

Листинг 27.49. Выборка данных при помощи оператора `SELECT`

```
SELECT id_catalog, name FROM catalogs;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Если требуется вывести все столбцы таблицы, необязательно перечислять их имена после ключевого слова `SELECT`, достаточно заменить этот список символом `*` (все столбцы) (листинг 27.50).

Листинг 27.50. Использование символа * (все столбцы)

```
SELECT * FROM catalogs;
```

```
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

К списку столбцов в операторе `SELECT` прибегают в том случае, если необходимо изменить порядок следования столбцов в результирующей таблице или выбрать только часть столбцов (листинг 27.51).

Листинг 27.51. Изменение числа и порядка выводимых столбцов

```
SELECT name, id_catalog FROM catalogs;
+--------------------+------------+
| name               | id_catalog |
+--------------------+------------+
| Видеоадаптеры      |          3 |
| Жесткие диски      |          4 |
| Материнские платы  |          2 |
| Оперативная память |          5 |
| Процессоры         |          1 |
```

27.16. Условная выборка

Ситуация, когда требуется изменить количество выводимых строк, встречается гораздо чаще, чем ситуация, когда требуется изменить число и порядок выводимых столбцов. Для ввода в SQL-запрос такого рода ограничений в операторе `SELECT` предназначено специальное ключевое слово `WHERE`, после которого следует логическое условие. Если запись удовлетворяет такому условию, она попадает в результат выборки, в противном случае такая запись отбрасывается.

В листинге 27.52 приводится пример запроса, извлекающего из таблицы `catalogs` записи, чей первичный ключ `id_catalog` больше (оператор `>`) 2.

Листинг 27.52. Использование конструкции `WHERE`

```
SELECT * FROM catalogs WHERE id_catalog > 2;
```

```
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

Оператор "больше" > возвращает TRUE (истину), если левый аргумент больше правого, и FALSE (ложь), если правый аргумент меньше левого или равен ему. Если логическое выражение возвращает TRUE для текущей записи, запись попадает в результирующую таблицу.

Помимо оператора > имеется еще несколько логических операторов, представленных в табл. 27.7.

Таблица 27.7. *Логические операторы*

Оператор	Описание
a > b	Возвращает TRUE, если аргумент a больше b, и FALSE — в противном случае
a < b	Возвращает TRUE, если аргумент a меньше b, и FALSE — в противном случае
a >= b	Возвращает TRUE, если аргумент a больше или равен аргументу b, и FALSE — в противном случае
a <= b	Возвращает TRUE, если аргумент a меньше или равен аргументу b, и FALSE — в противном случае
a = b	Возвращает TRUE, если аргумент a равен аргументу b, и FALSE — в противном случае
a <> b	Возвращает TRUE, если аргумент a не равен аргументу b, и FALSE — в противном случае
a != b	Аналогичен оператору <>
a <=> b	Оператор эквивалентности; по своему действию аналогичен оператору равенства =, однако допускает в качестве одного из аргументов NULL

Следует отметить, что логические операторы возвращают TRUE (истина) и FALSE (ложь) в стиле языка программирования C, т. е. без использования специальных констант и обозначений. За ложь принимается 0, а за истину — любое число, отличное от нуля. В этом легко убедиться, если вывести логическое выражение в результирующую таблицу (листинг 27.53).

Листинг 27.53. Вывод логических значений в результирующую таблицу

```
SELECT id_catalog, id_catalog > 2 FROM catalogs;
```

```
+------------+----------------+
| id_catalog | id_catalog > 2 |
+------------+----------------+
|          1 |              0 |
|          2 |              0 |
|          3 |              1 |
|          4 |              1 |
|          5 |              1 |
+------------+----------------+
```

Условие может быть составным и объединяться при помощи логических операторов. В листинге 27.54 используется составное условие: первичный ключ должен быть больше 2 и меньше или равен 4. Для объединения этих двух условий используется оператор `AND` (И).

> ***Замечание***
>
> Помимо оператора `AND` (И), для объединения логических выражений может использоваться оператор `OR` (ИЛИ).

Листинг 27.54. Использование составного логического условия

```
SELECT * FROM catalogs
WHERE id_catalog > 2 AND id_catalog <= 4;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

В табл. 27.8 и 27.9 представлены правила, по которым операторы `AND` и `OR` объединяют свои операнды.

Таблица 27.8. *Оператор* `AND`

Операнд/результат	Значения			
Первый операнд	TRUE	TRUE	FALSE	FALSE
Второй операнд	TRUE	FALSE	TRUE	FALSE
Результат	TRUE	FALSE	FALSE	FALSE

Как видно из табл. 27.8, оператор `AND` возвращает `TRUE` только в том случае, если оба его операнда принимают истинное значение (`TRUE`).

Таблица 27.9. *Оператор* `OR`

Операнд/результат	Значения			
Первый операнд	TRUE	TRUE	FALSE	FALSE
Второй операнд	TRUE	FALSE	TRUE	FALSE
Результат	TRUE	TRUE	TRUE	FALSE

Как видно из табл. 27.9, оператор `OR` возвращает `FALSE` только в том случае, если оба его операнда принимают ложное значение (`FALSE`).

Помимо бинарных логических операторов `AND` и `OR`, СУБД MySQL поддерживает унарный оператор отрицания `NOT`. Оператор возвращает истину для ложного аргумента и ложь для истинного аргумента. В листинге 27.55 демонстрируется использование оператора `NOT`.

Листинг 27.55. Использование оператора `NOT`

```
SELECT id_catalog, id_catalog > 2, NOT id_catalog > 2 FROM catalogs;
+------------+----------------+--------------------+
| id_catalog | id_catalog > 2 | NOT id_catalog > 2 |
+------------+----------------+--------------------+
|          1 |              0 |                  1 |
|          2 |              0 |                  1 |
|          3 |              1 |                  0 |
|          4 |              1 |                  0 |
|          5 |              1 |                  0 |
+------------+----------------+--------------------+
```

Помимо операторов `OR` и `AND`, язык SQL предоставляет еще один логический оператор: исключающее ИЛИ — `XOR`. Правила, по которым работает данный оператор, представлены в табл. 27.10.

Таблица 27.10. *Оператор* `XOR`

Операнд/результат	Значения			
Первый операнд	TRUE	TRUE	FALSE	FALSE
Второй операнд	TRUE	FALSE	TRUE	FALSE
Результат	FALSE	TRUE	TRUE	FALSE

Оператор `XOR` можно эмулировать при помощи остальных логических операторов по формуле: `(a AND (NOT b)) OR ((NOT a) and b)`.

Для выборки записей из определенного интервала используется оператор BETWEEN *min* AND *max*, возвращающий записи, значения которых лежат в диапазоне от *min* до *max* (листинг 27.56).

Листинг 27.56. Использование конструкции BETWEEN

```
SELECT * FROM catalogs WHERE id_catalog BETWEEN 3 AND 4;
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

Как видно из листинга 27.56, в результирующую таблицу возвращаются записи в диапазоне от 3 до 4.

Существует конструкция, противоположная конструкции BETWEEN — NOT BETWEEN, которая возвращает записи, не попадающие в интервал между *min* и *max* (листинг 27.57).

Листинг 27.57. Использование конструкции NOT BETWEEN

```
SELECT * FROM catalogs WHERE id_catalog NOT BETWEEN 3 AND 4;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Иногда требуется извлечь записи, удовлетворяющие не диапазону, а списку, например, записи с id_catalog из списка (1,2,5), как показано в листинге 27.58. Для этого предназначена конструкция IN.

Листинг 27.58. Использование оператора IN

```
SELECT * FROM catalogs WHERE id_catalog IN (1,2,5);
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          2 | Материнские платы  |
|          5 | Оперативная память |
|          1 | Процессоры         |
+------------+--------------------+
```

Конструкция `NOT IN` является противоположной оператору `IN` и возвращает 1 (истина), если проверяемое значение не входит в список, и 0 (ложь), если оно присутствует в списке (листинг 27.59).

Листинг 27.59. Использование конструкции `NOT IN`

```
SELECT * FROM catalogs WHERE id_catalog NOT IN (1,2,5);
+------------+---------------+
| id_catalog | name          |
+------------+---------------+
|          3 | Видеоадаптеры |
|          4 | Жесткие диски |
+------------+---------------+
```

В конструкции `WHERE` могут использоваться не только числовые столбцы. В листинге 27.60 из таблицы `catalogs` извлекается запись, соответствующая элементу каталога "Процессоры".

Листинг 27.60. Работа с текстовыми полями

```
SELECT * FROM catalogs WHERE name = 'процессоры';
+------------+------------+
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Зачастую условную выборку с участием строк удобнее производить не при помощи оператора равенства `=`, а посредством оператора `LIKE`, который позволяет использовать простейшие регулярные выражения. Оператор `LIKE` имеет следующий синтаксис:

expr `LIKE` *pat*

Оператор часто используется в конструкции `WHERE` и возвращает 1 (истину), если выражение *expr* соответствует выражению *pat*, и 0 — в противном случае. Главное преимущество оператора `LIKE` перед оператором равенства заключается в возможности использования специальных символов, приведенных в табл. 27.11.

***Таблица 27.11.** Специальные символы, используемые в операторе `LIKE`*

Символ	Описание
%	Соответствует любому количеству символов, даже их отсутствию
_	Соответствует ровно одному символу

При помощи специальных символов, представленных в табл. 27.11, можно задать различные шаблоны соответствия. В листинге 27.61 приводится пример выборки записей, которые содержат названия элементов каталога, заканчивающиеся на символ 'ы'.

Листинг 27.61. Извлечение из таблицы `catalogs` записей, названия которых заканчиваются на 'ы'

```
SELECT * FROM catalogs WHERE name LIKE '%ы';
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          3 | Видеоадаптеры     |
|          2 | Материнские платы |
|          1 | Процессоры        |
+------------+-------------------+
```

Оператор NOT LIKE, показанный в листинге 27.62, противоположен по действию оператору LIKE и имеет следующий синтаксис:

expr NOT LIKE *pat*

Оператор возвращает 0, если выражение *expr* соответствует выражению *pat*, и 1 — в противном случае. Таким образом, с его помощью можно извлечь записи, которые не удовлетворяют указанному условию.

Листинг 27.62. Использование оператора `NOT LIKE`

```
SELECT * FROM catalogs WHERE name NOT LIKE '%ы';
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

27.17. Псевдонимы столбцов

Имена вычисляемых столбцов, формируемые выражениями или функциями, часто достаточно длинны и не удобны для использования в прикладных программах, выполняющих доступ к элементам в результирующей таблице по имени столбца. В SELECT-запросе столбцу можно назначить новое имя при помощи оператора AS. В листинге 27.63 результату функции DATE_FORMAT() присваивается новый псевдоним printdate.

Листинг 27.63. Использование оператора AS

```
SELECT id_catalog, name, DATE_FORMAT(putdate, '%d.%m.%Y') AS printdate
FROM catalogs;
+------------+--------------------+------------+
| id_catalog | name               | printdate  |
+------------+--------------------+------------+
|          3 | Видеоадаптеры      | 10.01.2007 |
|          4 | Жесткие диски      | 05.01.2007 |
|          2 | Материнские платы  | 28.12.2006 |
|          5 | Оперативная память | 20.12.2006 |
|          1 | Процессоры         | 10.01.2007 |
+------------+--------------------+------------+
```

27.18. Сортировка записей

Как видно из предыдущих листингов данной главы, результат выборки представляет собой записи, которые располагаются в порядке, в котором они хранятся в базе данных. Однако часто требуется отсортировать значения по одному из столбцов. Это осуществляется при помощи конструкции `ORDER BY`, которая следует за выражением `SELECT`. После конструкции `ORDER BY` указывается столбец (или столбцы), по которому следует сортировать данные.

Как видно из листинга 27.64, первый запрос сортирует результат выборки по полю `id_catalog`, а второй — по полю `name`.

Листинг 27.64. Использование конструкции ORDER BY

```
SELECT id_catalog, name FROM catalogs ORDER BY id_catalog;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          1 | Процессоры         |
|          2 | Материнские платы  |
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
SELECT id_catalog, name FROM catalogs ORDER BY name;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          3 | Видеоадаптеры      |
|          4 | Жесткие диски      |
```

```
|           2 | Материнские платы  |
|           5 | Оперативная память |
|           1 | Процессоры         |
+-------------+--------------------+
```

По умолчанию сортировка производится в прямом порядке, однако, добавив после имени столбца ключевое слово `DESC`, можно добиться сортировки в обратном порядке (листинг 27.65).

Листинг 27.65. Обратная сортировка

```
SELECT id_catalog, name FROM catalogs ORDER BY id_catalog DESC;
+-------------+--------------------+
| id_catalog | name               |
+-------------+--------------------+
|           5 | Оперативная память |
|           4 | Жесткие диски      |
|           3 | Видеоадаптеры      |
|           2 | Материнские платы  |
|           1 | Процессоры         |
+-------------+--------------------+
```

Сортировку записей можно производить и по нескольким столбцам. Пусть имеется таблица `tbl`, состоящая из двух столбцов: `id_catalog` и `putdate`, содержащая записи с первичным ключом каталога `id_catalog` и датой обращения в поле `putdate`. В листинге 27.66 приводится дамп, позволяющий развернуть таблицу `tbl`.

Листинг 27.66. Таблица `tbl`

```
CREATE TABLE tbl (
  id_catalog int(11) NOT NULL,
  putdate datetime NOT NULL
);
INSERT INTO tbl VALUES (5, '2007-01-04 05:01:58');
INSERT INTO tbl VALUES (3, '2007-01-03 12:10:45');
INSERT INTO tbl VALUES (4, '2007-01-10 16:10:25');
INSERT INTO tbl VALUES (1, '2006-12-20 08:34:09');
INSERT INTO tbl VALUES (2, '2007-01-06 20:57:42');
INSERT INTO tbl VALUES (2, '2006-12-24 18:42:41');
INSERT INTO tbl VALUES (5, '2006-12-25 09:35:01');
INSERT INTO tbl VALUES (1, '2006-12-23 15:14:26');
INSERT INTO tbl VALUES (4, '2006-12-26 21:32:00');
INSERT INTO tbl VALUES (3, '2006-12-25 12:11:10');
```

Для того чтобы отсортировать таблицу `tbl` сначала по полю `id_catalog`, а затем по полю `putdate`, можно воспользоваться запросом, представленным в листинге 27.67.

Листинг 27.67. Сортировка таблицы по двум столбцам

```
SELECT * FROM tbl ORDER BY id_catalog, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          1 | 2006-12-23 15:14:26 |
|          1 | 2006-12-20 08:34:09 |
|          2 | 2007-01-06 20:57:42 |
|          2 | 2006-12-24 18:42:41 |
|          3 | 2007-01-03 12:10:45 |
|          3 | 2006-12-25 12:11:10 |
|          4 | 2007-01-10 16:10:25 |
|          4 | 2006-12-26 21:32:00 |
|          5 | 2007-01-04 05:01:58 |
|          5 | 2006-12-25 09:35:01 |
+------------+---------------------+
```

Как видно из листинга 27.67, записи в таблице `tbl` сначала сортируются по столбцу `id_catalog`, а совпадающие в рамках одного значения `id_catalog` записи сортируются по полю `putdate` в обратном порядке. Следует отметить, что ключевое слово `DESC` относится только к полю `putdate`. Для того чтобы отсортировать оба столбца в обратном порядке, потребуется снабдить ключевым словом как столбец `id_catalog`, так и `putdate` (листинг 27.68).

Листинг 27.68. Обратная сортировка по двум столбцам

```
SELECT * FROM tbl ORDER BY id_catalog DESC, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          5 | 2007-01-04 05:01:58 |
|          5 | 2006-12-25 09:35:01 |
|          4 | 2007-01-10 16:10:25 |
|          4 | 2006-12-26 21:32:00 |
|          3 | 2007-01-03 12:10:45 |
|          3 | 2006-12-25 12:11:10 |
|          2 | 2007-01-06 20:57:42 |
|          2 | 2006-12-24 18:42:41 |
|          1 | 2006-12-23 15:14:26 |
|          1 | 2006-12-20 08:34:09 |
+------------+---------------------+
```

Для прямой сортировки также существует ключевое слово `ASC` (в противовес ключевому слову `DESC`), но поскольку по умолчанию записи сортируются в прямом порядке, данное ключевое слово часто опускают.

27.19. Вывод записей в случайном порядке

Для вывода записей в случайном порядке используется конструкция `ORDER BY RAND()`. В листинге 27.69 демонстрируется вывод содержимого таблицы `catalogs` в случайном порядке.

Листинг 27.69. Вывод содержимого таблицы в случайном порядке

```
SELECT id_catalog, name FROM catalogs ORDER BY RAND();
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          4 | Жесткие диски      |
|          5 | Оперативная память |
|          2 | Материнские платы  |
|          1 | Процессоры         |
|          3 | Видеоадаптеры      |
+------------+--------------------+
```

Если требуется вывести лишь одну случайную запись, используется конструкция `LIMIT 1` (листинг 27.70).

Листинг 27.70. Вывод одной случайной записи

```
SELECT id_catalog, name FROM catalogs ORDER BY RAND() LIMIT 1;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          2 | Материнские платы |
+------------+-------------------+
```

27.20. Ограничение выборки

Результат выборки может содержать сотни и тысячи записей. Их вывод и обработка занимают значительное время и серьезно загружают сервер базы данных, поэтому информацию часто разбивают на страницы и предоставляют ее пользователю порциями. Извлечение только части запроса требует меньше времени и вычислений, кроме того, пользователю часто бывает достаточно просмотреть первые несколько записей.

Постраничная навигация используется при помощи ключевого слова `LIMIT`, за которым следует количество записей, выводимых за один раз. В листинге 27.71 извлекаются первые две записи таблицы `catalogs`, при этом одновременно осуществляется их обратная сортировка по полю `id_catalog`.

Листинг 27.71. Использование ключевого слова `LIMIT`

```
SELECT id_catalog, name FROM catalogs
ORDER BY id_catalog DESC
LIMIT 2;
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          5 | Оперативная память |
|          4 | Жесткие диски      |
+------------+--------------------+
```

Для того чтобы извлечь следующие две записи, используется ключевое слово `LIMIT` с двумя числами: первое указывает позицию, начиная с которой необходимо вернуть результат, а второе — количество извлекаемых записей (листинг 27.72).

Листинг 27.72. Извлечение записей, начиная со второй позиции

```
SELECT id_catalog, name FROM catalogs
ORDER BY id_catalog DESC
LIMIT 2, 2;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|          3 | Видеоадаптеры     |
|          2 | Материнские платы |
+------------+-------------------+
```

Для извлечения следующих двух записей необходимо использовать конструкцию `LIMIT 4, 2`.

27.21. Вывод уникальных значений

Очень часто встает задача выбора из таблицы уникальных значений. Воспользуемся таблицей `tbl` из листинга 27.68. Пусть требуется вывести все значения поля `id_catalog`; для этого можно воспользоваться запросом, представленным в листинге 27.73.

Листинг 27.73. Выборка значений поля `id_catalog` из таблицы `tbl`

```
SELECT id_catalog FROM tbl ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          1 |
|          2 |
|          2 |
|          3 |
|          3 |
|          4 |
|          4 |
|          5 |
|          5 |
+------------+
```

Как видно из листинга 27.73, результат не совсем удобен для восприятия. Было бы лучше, если бы запрос вернул уникальные значения столбца `id_catalog`. Для этого перед именем столбца можно использовать ключевое слово `DISTINCT`, которое предписывает MySQL извлекать только уникальные значения (листинг 27.74).

> ***Замечание***
>
> Ключевое слово `DISTINCT` имеет синоним — `DISTINCTROW`.

Листинг 27.74. Выборка уникальных значений

```
SELECT DISTINCT id_catalog FROM tbl ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

Как показано в листинге 27.74, результат запроса не содержит ни одного повторяющегося значения.

Для ключевого слова DISTINCT имеется противоположное слово ALL, которое предписывает извлечение всех значений столбца, в том числе и повторяющихся. Поскольку такое поведение установлено по умолчанию, ключевое слово ALL часто опускают.

Часто для извлечения уникальных записей прибегают также к конструкции GROUP BY, содержащей имя столбца, по которому группируется результат (листинг 27.75).

> ***Замечание***
>
> Конструкция GROUP BY располагается в SELECT-запросе перед конструкциями ORDER BY и LIMIT.

Листинг 27.75. Использование конструкции GROUP BY

```
SELECT id_catalog FROM tbl
GROUP BY id_catalog ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

27.22. Объединение таблиц

Как было сказано ранее, оператор SELECT возвращает результат в виде таблицы. Если формат результирующих таблиц (число, порядок следования и тип столбцов) совпадает, то возможно объединение результатов выполнения двух операторов SELECT в одну результирующую таблицу. Это достигается использованием оператора UNION. Пусть имеются два SELECT-запроса, представленные в листинге 27.76.

Листинг 27.76. Одиночные SELECT-запросы

```
SELECT id_catalog FROM catalogs;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

```
SELECT id + 5 FROM tbl;
+--------------+
| id_order + 5 |
+--------------+
|            6 |
|            7 |
|            8 |
|            9 |
|           10 |
+--------------+
```

Объединить результаты из этих двух таблиц можно, соединив два запроса SELECT при помощи ключевого слова UNION, как это продемонстрировано в листинге 27.77.

Листинг 27.77. Использование ключевого слова UNION

```
SELECT id_catalog FROM catalogs
UNION
SELECT id + 5 FROM tbl;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
|          6 |
|          7 |
|          8 |
|          9 |
|         10 |
+------------+
```

Во втором запросе SELECT, производящем выборку из таблицы tbl, к значению первичного ключа id добавляется значение 5, и, таким образом, все значения в результирующей таблице становятся уникальными. Однако если результирующая таблица содержит повторяющиеся строки, СУБД MySQL автоматически отбрасывает дубликаты. Изменить поведение по умолчанию можно при помощи ключевого слова ALL, которое добавляется после оператора UNION. Использование UNION ALL требует, чтобы возвращались все строки из обеих результирующих таблиц (листинг 27.78).

Листинг 27.78. Использование ключевого слова `UNION ALL`

```
SELECT id_catalog FROM catalogs
UNION ALL
SELECT id FROM tbl;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

27.23. Функции MySQL

СУБД MySQL предоставляет широкий набор встроенных функций, которые можно использовать для преобразования данных. В этом разделе будут рассмотрены наиболее часто используемые функции. За полным списком функций следует обратиться либо к официальной документации, либо к одной из наших книг, полностью посвященных MySQL (см. список литературы в конце книги).

Замечание

Отличительной чертой MySQL является то, что при использовании функций пробелы между именем функции и круглыми скобками недопустимы. Так, написание `NOW()` корректно, а `NOW ()` — нет. СУБД MySQL можно заставить игнорировать пробелы между именем функции и круглыми скобками, если запустить сервер с параметром `--sql-mode=IGNORE-SPACE` или поместить в секцию `[mysqld]` директиву `sql-mode=IGNORE-SPACE`.

27.23.1. Математические функции

Математические функции предназначены для выполнения разнообразных математических расчетов. В табл. 27.12 представлен список математических функций, доступных в СУБД MySQL.

Таблица 27.12. *Математические функции MySQL*

Функция	Описание
ABS(*X*)	Возвращает абсолютное значение аргумента *X*
ACOS(*X*)	Возвращает арккосинус числа *X* или NULL, если значение *X* выходит из диапазона от –1 до 1
ASIN(*X*)	Возвращает арксинус числа *X* или NULL, если значение X выходит из диапазона от –1 до 1
ATAN(*X*)	Возвращает арктангенс числа *X*
ATAN(*X*, *Y*), ATAN2(*X*, *Y*)	Возвращает арктангенс частного *X*/*Y*, функции аналогичны ATAN(*X*/*Y*)
CEILING(*X*), CEIL(*X*)	Функция принимает дробное число X и возвращает наименьшее целое, не меньше чем *X*
COALESCE(*val*, ...)	Возвращает первый элемент из списка, который не равен NULL
COS(*X*)	Возвращает косинус угла *X*, заданного в радианах
COT(*X*)	Возвращает котангенс угла *X*, заданного в радианах
CRC32(*str*)	Возвращает значение кода циклической проверки избыточности строки *str* — 32-битное значение в диапазоне от 0 до $2^{32} - 1$. Если в качестве аргумента функции передано значение NULL, функция возвращает NULL
DEGREES(X)	Возвращает значение угла X, преобразованное из радиан в градусы
GREATEST(*val*, ...)	Возвращает максимальное значение из списка
EXP(*X*)	Возвращает значение степени числа *X*: e^X, где *e* — основание натурального логарифма
FLOOR(*X*)	Принимает дробное число *X* и возвращает максимальное целое значение, не больше чем *X*
INTERVAL(*N*, *N1*, *N2*, ...)	Возвращает 0, если *N* < *N1*, и 1, если *N* < *N2*, и т. д., т. е. возвращается позиция, где происходит нарушение монотонного убывания значений списка. Все аргументы трактуются как целые числа
LEAST(*val*, ...)	Возвращает минимальное значение из списка
LOG(*X*) , LN(*X*)	Возвращает натуральный логарифм (с основанием *e*) числа *X*
LOG(*B*, *X*)	Возвращает логарифм числа *X* по основанию B
LOG2(*X*)	Возвращает логарифм числа *X* по основанию 2
LOG10(*X*)	Возвращает логарифм числа *X* по основанию 10
MOD(*M*, *N*), *M* % *N*, *M* MOD *N*	Возвращает остаток от деления *M* на *N*

Таблица 27.12 *(окончание)*

Функция	Описание
`PI()`	Возвращает значение числа π
`POWER(X, Y)`, `POW(X, Y)`	Возвращает значение числа `X`, возведенного в степень `Y`, — X^Y
`RADIANS(X)`	Возвращает значение угла `X`, преобразованное из градусов в радианы
`RAND()`	Возвращает случайное значение с плавающей точкой в диапазоне от 0.0 до 1.0. Каждый раз генерируется одна и также последовательность случайных чисел
`RAND(N)`	Возвращает случайное значение с плавающей точкой в диапазоне от 0.0 до 1.0. Генератор случайных чисел инициируется числом `N`
`ROUND(X)`	Возвращает округленное до ближайшего целого значение числа `X`
`ROUND(X, D)`	Возвращает значение числа `X`, округленное до `D` знаков после запятой
`SIGN(X)`	Позволяет определить знак числа `X` и возвращает –1, 0 или 1, если `X` отрицательно, равно 0 или положительно, соответственно
`SIN(X)`	Возвращает синус угла `X`, заданного в радианах
`SQPT(X)`	Возвращает квадратный корень числа `X`
`TAN(X)`	Возвращает тангенс угла `X`, заданного в радианах
`TRUNCATE(X, D)`	Возвращает число `X` с дробной частью, имеющей `D` знаков после запятой. Если количество знаков после запятой в числе `X` больше `D`, лишние разряды усекаются, если меньше, то в конец числа добавляются нули

Вычисление площади треугольников

Пусть имеется таблица `triangle`, которая состоит из трех полей, содержащих величины угла (`angle`) и длины двух прилежащих сторон *a* и *b* (рис. 27.4). В листинге 27.79 приведен дамп таблицы `triangle`.

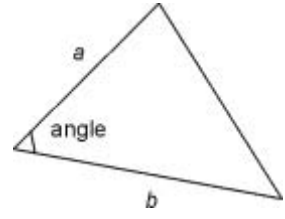


Рис. 27.4. Вычисление площади треугольника

Листинг 27.79. Дамп таблицы `triangle`

```
CREATE TABLE triangle (
  angle DOUBLE NOT NULL,
  A DOUBLE NOT NULL,
  B DOUBLE NOT NULL
);
INSERT INTO triangle VALUES (45, 1.414, 1),
                            (60, 2.707, 2.104),
                            (56, 2.088, 2.112),
                            (23, 5.014, 2.304),
                            (38, 3.482, 4.708);
SELECT * FROM triangle;
+-------+-------+-------+
| angle | A     | B     |
+-------+-------+-------+
|    45 | 1.414 |     1 |
|    60 | 2.707 | 2.104 |
|    56 | 2.088 | 2.112 |
|    23 | 5.014 | 2.304 |
|    38 | 3.482 | 4.708 |
+-------+-------+-------+
```

Вычислим площади треугольников, параметры которых содержатся в таблице `triangle` (листинг 27.79), по формуле:

$$S = a \times b \times \sin(angle).$$

SQL-запрос, вычисляющий площади треугольников по приведенной выше формуле, может выглядеть так, как это представлено в листинге 27.80.

Листинг 27.80. Вычисление площадей треугольников

```
SELECT A*B*SIN(RADIANS(angle))/2.0 AS S FROM triangle;
+-----------------+
| S               |
+-----------------+
| 0.4999244942989 |
| 2.4662359679828 |
| 1.8279681567786 |
| 2.2569130117385 |
| 5.0463480871756 |
+-----------------+
```

Как видно из листинга, прежде чем воспользоваться функцией синуса `SIN()`, необходимо конвертировать угол из градусов в радианы при помощи функции `RADIANS()`.

Округление результатов вычисления

Как видно из листинга 27.80, результат вычисления имеет 13 знаков после точки, что может быть не очень удобным для восприятия. В листинге 27.81 результат вычислений округляется до третьего знака после точки при помощи функции `ROUND()`.

Листинг 27.81. Округление результата вычисления до третьего знака после точки

```
SELECT ROUND(A*B*SIN(RADIANS(angle))/2.0, 3) AS S FROM triangle;
+-------+
| S     |
+-------+
| 0.500 |
| 2.466 |
| 1.828 |
| 2.257 |
| 5.046 |
+-------+
```

Функция `ROUND()` осуществляет математическое округление до ближайшего целого числа.

Функция `ROUND()` — не единственная функция управления дробными числами. Функция `CEILING(X)` принимает дробное число `X` и возвращает минимальное целое, не меньше чем `X`.

Функция `CEILING()` не производит округления, как может показаться на первый взгляд. Она возвращает первое целое число, которое встречает справа от значения аргумента (рис. 27.5).

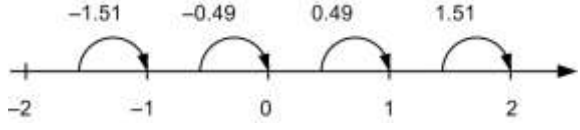


Рис. 27.5. Принцип действия функции `CEILING()`

В листинге 27.82 демонстрируется пример использования функции `CEILING()`.

Листинг 27.82. Использование функции `CEILING()`

```
SELECT CEILING(0.49), CEILING(1.51), CEIL(-0.49), CEIL(-1.51);
+---------------+---------------+-------------+-------------+
| CEILING(0.49) | CEILING(1.51) | CEIL(-0.49) | CEIL(-1.51) |
+---------------+---------------+-------------+-------------+
|             1 |             2 |           0 |          -1 |
+---------------+---------------+-------------+-------------+
```

Функция FLOOR(X) принимает дробное число X и возвращает максимальное целое значение, не больше чем X. Эта функция сходна по действию с функцией CEILING(X), но сдвиг происходит в обратную сторону. Пример использования функции FLOOR() представлен в листинге 27.83.

Листинг 27.83. Использование функции FLOOR()

```
SELECT FLOOR(0.49), FLOOR(1.51), FLOOR(-0.49), FLOOR(-1.51);
+-------------+-------------+--------------+--------------+
| FLOOR(0.49) | FLOOR(1.51) | FLOOR(-0.49) | FLOOR(-1.51) |
+-------------+-------------+--------------+--------------+
|           0 |           1 |           -1 |           -2 |
+-------------+-------------+--------------+--------------+
```

Последней функцией, управляющей дробными числами, является функция TRUNCATE(X,D), которая возвращает число X с дробной частью, имеющей D знаков после запятой. Если количество знаков после запятой в числе X больше D, лишние разряды усекаются, если меньше, то в конец числа добавляются нули (листинг 27.84).

Листинг 27.84. Использование функции TRUNCATE()

```
SELECT TRUNCATE(1.284,0), TRUNCATE(1.284,1), TRUNCATE(1.284,4);
+------------------+------------------+------------------+
| TRUNCATE(1.284,0) | TRUNCATE(1.284,1) | TRUNCATE(1.284,4) |
+------------------+------------------+------------------+
|                1 |              1.2 |           1.2840 |
+------------------+------------------+------------------+
```

27.23.2. Функции даты и времени

Группа функций даты и времени предназначена для форматирования и преобразования календарных значений. В табл. 27.13 представлен список календарных функций, доступных в СУБД MySQL.

ЗАМЕЧАНИЕ

Сложение и вычитание временных интервалов более подробно обсуждается в *разд. 27.9*.

***Таблица 27.13.** Функции даты и времени MySQL*

Функция	Описание
ADDDATE(*date*, INTERVAL *expr type*), DATE_ADD(*date*, INTERVAL *expr type*)	Возвращает время *date*, к которому прибавлен временной интервал, определяемый вторым параметром
ADDTIME(*expr1*, *expr2*)	Возвращает результат сложения двух календарных значений *expr1* и *expr2*

Таблица 27.13 *(продолжение)*

Функция	Описание
CONVERT_TZ(*dt*, *from_tz*, *to_tz*)	Переводит DATETIME-значение *dt* из часового пояса *from_tz* в *to_tz*. Если аргументы ошибочны, возвращается значение NULL
CURDATE(), CURRENT_DATE, CURRENT_DATE()	Возвращает текущую дату в формате 'YYYY-MM-DD' или YYYYDDMM в зависимости от того, вызывается функция в текстовом или числовом контексте
CURTIME(), CURRENT_TIME, CURRENT_TIME()	Возвращает текущее время суток в виде 'hh-mm-ss' или hhmmss в зависимости от того, вызывается функция в текстовом или числовом контексте
DATE(*datetime*)	Извлекает из значения даты и времени суток в формате DATETIME (например '2005-04-17 00:26:08') дату, отсекая часы, минуты и секунды — '2005-04-17'
DATEDIFF(*begin*, *end*)	Вычисляет разницу в днях между датами *begin* и *end*. Аргументы функции могут иметь тип DATE или DATETIME, однако при вычислении разницы используется только DATE-часть
DATE_FORMAT(*date*, *format*)	Форматирует время *date* в соответствии со строкой *format*
DAY(*date*), DAYOFMONTH(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в месяце (от 1 до 31)
DAYNAME(*date*)	Принимает в качестве аргумента дату *date* и возвращает день недели в виде полного английского названия
DAYOFWEEK(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня недели. Следует помнить, что в западных странах неделя начинается с воскресенья, для которого функция возвращает 1, для последнего дня недели, субботы, возвращается 7
DAYOFYEAR(*date*)	Принимает в качестве аргумента дату *date* и возвращает порядковый номер дня в году (от 1 до 366)
EXTRACT(*type* FROM *datetime*)	Принимает дату и время суток *datetime* и возвращает часть, определяемую параметром *type*
FROM_DAYS(*N*)	Принимает количество дней *N*, прошедших с нулевого года, и возвращает дату в формате 'YYYY-MM-DD'
FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*, *format*)	Принимает количество секунд, прошедших с полуночи 1 января 1970 года, и возвращает дату и время суток в виде строки 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss в зависимости от того, вызвана функция в строковом или числовом контексте

Таблица 27.13 *(продолжение)*

Функция	Описание
GET_FORMAT(*date* | *time* | *datetime*, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')	Возвращает строку форматирования, соответствующую одному из пяти стандартов времени
HOUR(*datetime*)	Возвращает значение часа (от 0 до 23) для времени *datetime*
LAST_DAY(*datetime*)	Принимает в качестве параметра значение даты *datetime* в кратком 'YYYY-MM-DD' или расширенном формате 'YYYY-MM-DD hh:mm:ss' и возвращает дату в кратком формате 'YYYY-MM-DD', день в которой выставлен на последний день текущего месяца
MAKEDATE(*year*, *dayofyear*)	Принимает в качестве параметров год *year*, номер дня в году *dayofyear* и возвращает дату в формате 'YYYY-MM-DD'
MAKETIME(*hour*, *minute*, *second*)	Принимает три параметра: часы *hour*, минуты *minute* и секунды *second*. В качестве результата функция возвращает время суток в формате 'hh:mm:ss'
MICROSECOND(*datetime*)	Извлекает из временного значения *datetime* микросекунды
MINUTE(*datetime*)	Возвращает значение минут (от 0 до 59) для времени суток *datetime*
MONTH(*datetime*)	Возвращает числовое значение месяца года (от 1 до 12) для даты *datetime*
MONTHNAME(*datetime*)	Возвращает строку с названием месяца для даты *datetime*
NOW(), CURRENT_TIME, CURRENT_TIME(), CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(), LOCALTIME(), LOCALTIME, LOCALTIMESTAMP(), LOCALTIMESTAMP, SYSDATE()	Возвращает текущие дату и время в виде строки в формате 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss, в зависимости от того, вызывается функция в строковом или числовом контексте
PERIOD_ADD(*period*, *N*)	Добавляет *N* месяцев к значению даты *period*. Аргумент *period* должен быть представлен в числовом формате YYYYMM или YYMM. Передача в качестве аргумента даты в любом другом формате приводит к непредсказуемому результату
PERIOD_DIFF(*period1*, *period2*)	Вычисляет разницу в месяцах между двумя датами *period1* и *period2*, которые представлены в числовом формате YYYYMM или YYMM

Таблица 27.13 *(продолжение)*

Функция	Описание
QUARTER(*datetime*)	Возвращает значение квартала года (от 1 до 4) для даты *datatime*, которая передается в формате 'YYYY-MM-DD' или 'YYYY-MM-DD hh:mm:ss'
SECOND(*time*)	Возвращает количество секунд для времени суток *time*, которое задается либо в виде строки 'hh:mm:ss', либо числа hhmmss
SEC_TO_TIME(*seconds*)	Принимает количество секунд *seconds*, прошедших от начала суток, и возвращает время в формате 'hh:mm:ss' или hhmmss, в зависимости от того, вызывается функция в строковом или числовом контексте
STR_TO_DATE(*str*, *format*)	Принимает дату и время суток в виде строки *str*, соответствующей формату *format*, и возвращает время в формате MySQL 'YYYY-MM-DD hh:mm:ss'
SUBDATE(*date*, INTERVAL *expr interval*), DATE_SUB(*date*, INTERVAL *expr interval*)	Возвращает дату *date*, из которой вычитается временной интервал, определяемый вторым параметром
SUBTIME(*datetime*, *time*)	Вычитает из временной величины *datetime* время суток *time* и возвращает результат
TIME(*datetime*)	Извлекает время суток 'hh:mm:ss' из даты и времени *datetime*
TIMEDIFF(*expr1*,*expr2*)	Возвращает разницу между временными значениями, заданными параметрами *expr1* и *expr2*
TIMESTAMP(*expr*)	Принимает в качестве аргумента дату и время суток в полном или кратком форматах и возвращает полный вариант в формате 'YYYY-MM-DD hh:mm:ss'
TIMESTAMPADD(*interval*, *int_expr*, *datetime_expr*)	Прибавляет к дате и времени суток *datetime_expr* в полном или кратком формате временной интервал *int_expr*, единицы измерения которого задаются параметром *interval*
TIMESTAMPDIFF(*interval*, *datetime_expr1*, *datetime_expr2*)	Возвращает разницу между двумя датами *datetime_expr1* или *datetime_expr2*, заданными в кратком 'YYYY-MM-DD' или полном 'YYYY-MM-DD hh:mm:ss' форматах. Единицы измерения, в которых функция возвращает результат, задаются параметром *interval*
TIME_FORMAT(*time*, *format*)	Форматирует время *time*, которое задается в виде 'hh:mm:ss' согласно строке форматирования *fomat*

Таблица 27.13 *(продолжение)*

Функция	Описание
TIME_TO_SEC(*time*)	Принимает в качестве аргумента время суток *time* в формате 'hh:mm:ss' и возвращает количество секунд, прошедших с начала суток
TO_DAYS(*date*)	Принимает дату *date* в полном 'YYYY-MM-DD hh:mm:ss' или кратком 'YYYY-MM-DD' формате и возвращает количество дней, прошедших с нулевого года
UNIX_TIMESTAMP(), UNIX_TIMESTAMP(*datetime*)	Вариант без параметра *datetime* возвращает количество секунд, прошедших с полуночи 1 января 1970 года. Кроме этого, функция может принимать необязательный параметр *datetime*, определяющий дату в кратком 'YYYY-MM-DD' или полном 'YYYY-MM-DD hh:mm:ss' форматах, возвращая в этом случае разницу в секундах между 1 января 1970 года и указанной в *datetime* датой
UTC_DATE()	Возвращает текущую дату в виде строки 'YYYY-MM-DD' или числа YYYYMMDD в зависимости от того, в каком контексте вызывается функция. В отличие от функций NOW() или CURDATE(), возвращающих локальную дату, в UTC_DATE(), время исчисляется по Гринвичу
UTC_TIME()	Возвращает текущее время суток в виде строки 'hh:mm:ss' или числа hhmmss в зависимости от того, в каком контексте вызывается функция. В отличие от функций NOW() или CURTIME(), возвращающих локальную дату, в UTC_DATE() время исчисляется по Гринвичу
UTC_TIMESTAMP()	Возвращает дату и время суток в виде строки 'YYYY-MM-DD hh:mm:ss' или в виде числа YYYYMMDDhhmmss в зависимости от того, в каком контексте вызывается функция. В отличие от функции NOW(), возвращающей локальную дату, в UTC_DATE() время исчисляется по Гринвичу
WEEK(*date*)	Возвращает номер недели (от 0 до 53) в году для даты *date*
WEEKDAY(*date*)	Принимает дату *date* в полном 'YYYY-MM-DD hh:mm:ss' или кратком 'YYYY-MM-DD' форматах и возвращает номер дня недели (0 для понедельника, 1 для вторника, ..., 6 для воскресенья)
WEEKOFYEAR(*datetime*)	Возвращает порядковый номер недели в году (от 1 до 53) для даты *datetime*
YEAR(*datetime*)	Возвращает год (от 1000 до 9999) для даты *datetime*

Таблица 27.13 *(окончание)*

Функция	Описание
YEARWEEK(*date*)	Возвращает число в формате YYYYWW, представляющее год и номер недели (от 0 до 53) в году, соответствующие дате *date*. Предполагается, что неделя начинается с воскресенья (как это принято на Западе)

Форматирование календарных значений

Функция DATE_FORMAT(*date*, *format*) форматирует время *date* в соответствии со строкой *format*. В строке *format* могут использоваться определители, представленные в табл. 27.14.

Таблица 27.14. *Определители строки форматирования функции* *DATE_FORMAT()*

Определитель	Описание
%a	Сокращенное наименование дня недели (Sun, ..., Sat)
%b	Сокращенное наименование месяца (Jan, ..., Dec)
%c	Месяц в числовой форме (1, ..., 12)
%D	День месяца с английским суффиксом (1st, 2nd, 3rd и т. д.)
%d	День месяца в числовой форме с ведущим нулем (01, ..., 31)
%e	День месяца в числовой форме (1, ..., 31)
%f	Микросекунды (000000, ..., 999999)
%H	Час с ведущим нулем (00, ..., 23)
%h	Час с ведущим нулем (01, ..., 12)
%I	Час с ведущим нулем (01, ..., 12)
%i	Минуты с ведущим нулем (00, ...,59)
%j	День года (001, ..., 366)
%k	Час с ведущим нулем (0, ..., 23)
%l	Час без ведущего нуля (1, ..., 12)
%M	Название месяца (January, ..., December)
%m	Месяц в числовой форме с ведущим нулем (01, ..., 12)
%p	AM или PM (для 12-часового формата)
%r	Время, 12-часовой формат (hh:mm:ss AM или hh:mm:ss PM)
%S	Секунды (00, ..., 59)
%s	Секунды (00, ..., 59)
%T	Время, 24-часовой формат (hh:mm:ss)

***Таблица 27.14** (окончание)*

Определитель	Описание
%U	Неделя (00, ..., 52), где воскресенье считается первым днем недели
%u	Неделя (00, ..., 52), где понедельник считается первым днем недели
%V	Неделя (01, ..., 53), где воскресенье считается первым днем недели. Используется с %X
%v	Неделя (01, ..., 53), где понедельник считается первым днем недели. Используется с %x
%W	Название дня недели (Sunday, ..., Saturday)
%w	День недели (0, ..., 6), где 0 — воскресенье, ..., 6 — суббота
%X	Год, в который началась неделя, где воскресенье считается первым днем недели, 4 разряда, используется с %V
%x	Год, в который началась неделя, где воскресенье считается первым днем недели, число, 4 разряда, используется с %v
%Y	Год, 4 разряда (YYYY)
%y	Год, 2 разряда (YY)
%%	Литерал %

Все другие символы, которые не указаны в табл. 27.14, выводятся без изменений.

Рассмотрим наиболее часто встречающуюся задачу преобразования календарного значения из MySQL-формата '2007-01-12 11:55:17' в привычный формат '12.01.2007 11:55'. Осуществить такое преобразование можно при помощи запроса, представленного в листинге 27.85.

Листинг 27.85. Преобразование даты при помощи функции DATE_FORMAT()

```
SELECT DATE_FORMAT(putdatetime,'%d.%m.%Y %H:%i') AS putdatetime FROM tbl;
+------------------+
| putdatetime      |
+------------------+
| 12.01.2007 11:55 |
+------------------+
```

Вычисление возраста человека

Пусть имеется таблица tbl, содержащая сведения о сотрудниках и состоящая из трех столбцов:

- id — первичный ключ таблицы;
- fio — фамилия, имя, отчество сотрудника;
- putdate — дата рождения.

В листинге 27.86 представлен запрос `CREATE TABLE`, создающий таблицу `tbl`.

Листинг 27.86. Создание таблицы `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  fio TINYTEXT NOT NULL,
  putdate DATE NOT NULL
);
INSERT INTO tbl VALUES (1, 'Тимирязев В.К.', '1972-12-02');
INSERT INTO tbl VALUES (2, 'Мальшев Ю.Г.', '1976-06-24');
INSERT INTO tbl VALUES (3, 'Абрамов К.Т.', '1968-10-24');
INSERT INTO tbl VALUES (4, 'Ганюшкин В.В.', '1986-12-07');
```

Необходимо вычислить возраст сотрудника на текущую дату. Один из вариантов состоит в преобразовании даты рождения и текущей даты в дни при помощи функции `TO_DAYS()` и делении на число 365.25 (листинг 27.87): в году 365 дней; дробное число 0.25 призвано компенсировать високосные года, которые случаются раз в четыре года.

Листинг 27.87. Вычисление возраста сотрудников

```
SELECT fio, (TO_DAYS(NOW()) - TO_DAYS(putdate))/365.25 AS putdate
FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. | 34.1109 |
| Мальшев Ю.Г.   | 30.5517 |
| Абрамов К.Т.   | 38.2177 |
| Ганюшкин В.В.  | 20.0986 |
+----------------+---------+
```

Для того чтобы избавиться от дробной части, можно воспользоваться функцией `FLOOR()` (листинг 27.88).

Листинг 27.88. Отбрасывание дробной части

```
SELECT fio, FLOOR((TO_DAYS(NOW()) - TO_DAYS(putdate))/365.25) AS putdate
FROM tbl;
+----------------+---------+
| fio            | putdate |
+----------------+---------+
| Тимирязев В.К. |      34 |
```

```
| Мальшев Ю.Г.    |      30 |
| Абрамов К.Т.    |      38 |
| Ганюшкин В.В.   |      20 |
+-----------------+---------+
```

Преобразование даты в UNIXSTAMP-формат

Пожалуй, самым распространенным является формат хранения времени UNIXSTAMP. Его популярность обусловлена удобством операций сложения и вычитания временных интервалов. UNIXSTAMP-формат — это количество секунд, прошедших с полуночи 1 января 1970 года. Для преобразования MySQL-даты в UNIXSTAMP-формат используется функция `UNIX_TIMESTAMP([datetime])`. Функция может принимать необязательный параметр *datetime*, определяющий дату в кратком `'YYYY-MM-DD'` или полном `'YYYY-MM-DD hh:mm:ss'` форматах, возвращая в этом случае разницу в секундах между 1 января 1970 года и указанной в *datetime* датой (листинг 27.89).

Листинг 27.89. Использование функции `UNIX_TIMESTAMP()`

```
SELECT UNIX_TIMESTAMP(),UNIX_TIMESTAMP('1997-10-04 22:23:00');
+------------------+---------------------------------------+
| UNIX_TIMESTAMP() | UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+------------------+---------------------------------------+
|       1113999060 |                             875989380 |
+------------------+---------------------------------------+
```

Функция `FROM_UNIXTIME(unix_timestamp)` решает обратную задачу: принимает число секунд, прошедших с полуночи 1 января 1970 года, и возвращает дату и время суток в виде строки `'YYYY-MM-DD hh:mm:ss'` или в виде числа `YYYYMMDDhhmmss` в зависимости от того, вызвана функция в строковом или числовом контексте (листинг 27.90).

Листинг 27.90. Использование функции `FROM_UNIXTIME()`

```
SELECT FROM_UNIXTIME(1113837695), FROM_UNIXTIME(0);
+--------------------------+---------------------+
| FROM_UNIXTIME(1113837695) | FROM_UNIXTIME(0)    |
+--------------------------+---------------------+
| 2005-04-18 19:21:35       | 1970-01-01 03:00:00 |
+--------------------------+---------------------+
```

Как видно из листинга 27.90, передача функции значения 0 в качестве аргумента приводит к выводу вместо полуночи 3 часов ночи. Это связано с тем, что время на машине, где расположен сервер MySQL, настроено для третьей временной зоны.

Функция `FROM_UNIXTIME(unix_timestamp, format)` может принимать второй параметр *format*, который представляет собой строку, содержащую определители

из табл. 27.14. Использование определителей позволяет отформатировать строку, например, так, как это представлено в листинге 27.91.

Листинг 27.91. Форматирование результата в функции FROM_UNIX()

```
SELECT FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается');
+-------------------------------------------------------------+
| FROM_UNIXTIME(1286481600,'В %Y году контракт заканчивается') |
+-------------------------------------------------------------+
| В 2010 году контракт заканчивается                          |
+-------------------------------------------------------------+
```

27.23.3. Строковые функции

Строковые функции позволяют осуществлять преобразование строк и строковых столбцов. В табл. 27.15 представлен список строковых функций, доступных в СУБД MySQL.

При использовании строковых функций следует помнить, что позиция в строке начинается с 1, а не с 0, как, например, в С-подобных языках программирования.

> ***Замечание***
>
> Если строковой функции передается в качестве одного из аргументов значение NULL, она также возвращает NULL.

Таблица 27.15. *Строковые функции MySQL*

Функция	Описание
ASCII(*str*)	Возвращает значение ASCII-кода первого символа строки *str*
BIN(*N*)	Принимает десятичное число *N* и возвращает его двоичное представление
BIT_LENGTH(*str*)	Принимает строку *str* и возвращает ее длину в битах
CHAR(*N1*,*N2*,...)	Принимает последовательность из ASCII-кодов и возвращает строку, созданную путем объединения соответствующих им символов
CHAR_LENGTH(*str*), CHARACTER_LENGTH(*str*)	Возвращает количество символов в строке *str*. В отличие от функции LENGTH() ведет подсчет не байтов, а символов, поэтому корректно работает с многобайтовыми кодировками
CHARSET(*str*)	Возвращает имя кодировки, в которой представлена строка, передаваемая функции в качестве аргумента
COLLATION(*str*)	Возвращает порядок сортировки, установленный для кодировки аргумента *str*
COMPRESS(*string_to_compress*)	Сжимает строку *string_to_compress* при помощи библиотеки zlib

Таблица 27.15 *(продолжение)*

Функция	Описание
CONCAT(*str1*, *str2*, ...)	Возвращает строку, созданную путем объединения всех аргументов, количество которых не ограничено
CONCAT_WS(*separator*, *str1*, *str2*, ...)	Объединяет аргументы *str1*, *str2* и т. д., помещая между ними разделитель *separator*
CONV(*N*, *from_base*, *to_base*)	Преобразует число *N* из одной системы счисления *from_base* в другую *to_base*. Параметры *from_base* и to_base могут принимать значения от 2 до 36
CONVERT(*expr*, *type*), CONVERT(*expr* USING *charset*)	Преобразует выражения *expr* в тип *type*. Вторая форма функции CONVERT() (с использованием USING) предназначена для преобразования текста из одной кодировки в другую
ELT(*N*, *str1*, *str2*, *str3*, ...)	Возвращает *N*-ю строку из списка аргументов *str1*, *str2*, *str3* и т. д.: для *N* = 1 возвращается *str1*, для *N* = 2 — *str2* и т. д. Если строка с номером *N* отсутствует или равна NULL, возвращается NULL
EXPORT_SET(*bits*, *on*, *off*[, *separator*[, *number_of_bits*]])	Возвращает число *bits* в двоичном представлении, 1 в котором заменяется на параметр *on*, а 0 — на параметр *off*. Необязательный параметр *separator* задает разделитель, которым по умолчанию является запятая. Параметр *number_of_bits* позволяет ограничить количество возвращаемых функцией символов (по умолчанию равно 64). Если данный параметр не задан, будут выведены все 64 символа, а недостающие символы будут установлены в 0 (*off*)
FIELD(*str*, *str1*, *str2*, ...)	Находит строку *str* в списке строк *str1*, *str2*, ... и возвращает номер строки в этом списке (нумерация начинается с 1)
FIND_IN_SET(*str*, *str_list*)	Ищет вхождение строки *str* в список *str_list* и возвращает номер строки в этом списке (нумерация начинается с 1). Если вхождение не найдено, возвращается 0. Параметр *str_list* представляет собой набор строк, разделенных запятыми
FORMAT(*X*, *D*)	Преобразует число *X* в строку с *D* знаками после запятой. Каждые три разряда разделяются запятой
HEX(*N_or_S*)	Возвращает значение аргумента *N_or_S* в виде шестнадцатеричного числа. Аргумент *N_or_S* может быть как числом, так и строкой. В первом случае возвращается обычное шестнадцатеричное число. Когда в качестве аргумента выступает строка, то функция переводит в шестнадцатеричное представление каждый символ строки и объединяет результат

Таблица 27.15 *(продолжение)*

Функция	Описание
INSERT(*str*, *pos*, *len*, *new_str*)	Возвращает строку *str*, в которой подстрока, начинающаяся с позиции *pos* и имеющая длину *len* символов, заменена подстрокой *new_str*. Функция возвращает строку *str* без изменений, если значение *pos* находится за пределами строки
INSTR(*str*, *substr*)	Возвращает позицию первого вхождения подстроки *substr* в строку *str*
LEFT(*str*, *len*)	Возвращает *len* крайних левых символов строки *str*. Если аргумент *len* превышает число символов в строке *str*, она возвращается без изменений. Если *len* равно NULL или меньше 0, возвращается пустая строка
LENGTH(*str*), OCTET_LENGTH(*str*)	Возвращает длину строки *str*. Данная функция может некорректно работать с многобайтными кодировками, т. к. фактически возвращает количество байтов в строке, а не символов
LOAD_FILE(*file_name*)	Читает файл *file_name* и возвращает его содержимое в виде строки. Файл должен быть расположен на сервере, а в параметре *file_name* необходимо указывать абсолютный путь к файлу. Если файл не существует или не может быть прочитан из-за того, что MySQL не обладает достаточными правами доступа, то возвращается NULL
LOCATE(*substr*, *str* [, *pos*]), POSITION(*substr* IN *str*)	Возвращает позицию первого вхождения подстроки *substr* в строку *str*. Если ни одно вхождение не найдено, возвращается 0. Если хотя бы один из аргументов функции равен NULL, функция возвращает NULL. При наличии необязательного аргумента *pos* поиск начинается с позиции, указанной в этом аргументе
LOWER(*str*), LCASE()	Возвращает строку *str*, в которой все символы записаны строчными символами
LPAD(*str*, *len*, *padstr*)	Возвращает строку *str*, дополненную слева строкой *padstr* до длины *len*
LTRIM(*str*)	Возвращает строку *str*, из которой удалены все начальные пробелы
MAKE_SET(*N*, *bit0_str*, *bit1_str*, ...)	Представляет число *N* в двоичной форме, создавая список для типа SET, заменяя 1 значениями из списка *bit0_str*, *bit1_str*, ..., при этом для первого разряда используется *bit0_str*, для второго — *bit1_str* и т. д.
MID(*str*, *pos* [, *len*]), SUBSTRING(*str*, *pos*, *len*)	Возвращает подстроку строки *str*, которая начинается с позиции *pos* и имеет длину *len* символов. Если третий параметр *len* не указывается, то подстрока возвращается, начиная с позиции *pos* и до конца строки *str*

Таблица 27.15 *(продолжение)*

Функция	Описание
OCT(*N*)	Возвращает число *N* в восьмеричной системе счисления
ORD(*str*)	Возвращает ASCII-код крайнего левого символа строки *str*. Аналогична функции ASCII(), но в отличие от последней корректно работает с многобайтными кодировками
QUOTE(*str*)	Экранирует строку *str* с тем, чтобы получить корректное значение для SQL-выражения. Строка заключается в одинарные кавычки, и каждое вхождение одинарной кавычки ('), обратного слеша (\), значения ASCII-символов nul и <Ctrl>+<Z> экранируются обратным слешем
REPEAT(*str*, *count*)	Возвращает строку, полученную из *count* повторений строки *str*. Если аргумент *count* имеет отрицательное значение или ноль, возвращается пустая строка
REPLACE(*str*, *from_str*, *to_str*)	Возвращает строку *str*, в которой все подстроки *from_str* заменены *to_str*
REVERSE(*str*)	Возвращает строку *str*, записанную в обратном порядке
RIGHT(*str*, *len*)	Возвращает *len* крайних правых символов строки *str* или всю строку *str*, если ее длина короче *len*. Функция возвращает пустую строку, если аргумент *len* равен NULL или меньше 1. Если строка *str* равна NULL, также возвращает NULL
RPAD(*str*, *len*, *padstr*)	Возвращает строку *str*, дополненную справа строкой *padstr* до длины *len*. Если строка *str* содержит более *len* символов, она усекается до *len*
RTRIM(*str*)	Возвращает строку *str*, из которой удалены все конечные пробелы
SOUNDEX(*str*)	Возвращает произношение строки *str* по алгоритму, описанному Д. Кнутом. Все неалфавитные символы в строке игнорируются, а национальные символы рассматриваются как гласные
SPACE(*N*)	Возвращает строку, состоящую из *N* пробелов, или пустую строку, если *N* имеет отрицательное значение
SUBSTRING_INDEX(*str*, *delim*, *N*)	Возвращает подстроку строки *str*. Если параметр *N* имеет положительное значение, то функция SUBSTRING_INDEX() находит *N*-ое вхождение (отсчет слева) подстроки *delim* в строку *str* и возвращает всю часть строки, расположенную слева от подстроки *delim*. Если *N* имеет отрицательное значение, то находится *N*-ое вхождение (отсчет справа) подстроки *delim* в строку *str* и возвращается часть строки, расположенная справа от подстроки *delim*

Таблица 27.15 *(окончание)*

Функция	Описание
TRIM([[BOTH | LEADING | TRAILING] [*remstr*] FROM] *str*)	Удаляет из строки *str* расположенные в начале (в конце) символы, указанные в строке *remstr*. Если указано ключевое слово LEADING, удаляются символы, расположенные в начале, если TRAILING — в конце, если BOTH — и в начале, и в конце. Если ни одно из ключевых слов не задано, то по умолчанию устанавливается BOTH. Если строка *remstr* не задана, в качестве удаляемых символов выступают пробелы
UNCOMPRESS(*string_to_uncompress*)	Разархивирует строку *string_to_uncompress*, сжатую при помощи функции COMPRESS(). Если в качестве аргумента функции передается обычная, не сжатая строка, возвращается NULL
UNCOMPRESSED_LENGTH(*compressed_string*)	Возвращает длину строки *compressed_string*, сжатой функцией COMPRESS() до применения данной функции, т. е. при помощи функции UNCOMPRESSED_LENGTH() можно выяснить, какой длины будет строка после разархивирования ее функцией UNCOMPRESS()
UNHEX(*str*)	Является обратной функции HEX() и интерпретирует каждую пару символов строки *str* как шестнадцатеричный код, который необходимо преобразовать в символ
UPPER(*str*), UCASE(*str*)	Переводит все символы строки *str* в верхний регистр

Изменение кодировки строки

Функция CONVERT(*expr* USING *charset*) преобразует текст из одной кодировки в другую (листинг 27.92).

Листинг 27.92. Преобразование кодировки

```
SELECT CONVERT('База данных MySQL' USING koi8r);
+----------------------------------------+
| CONVERT('База данных MySQL' USING koi8r) |
+----------------------------------------+
| аЮГЮ ДЮММШУ MySQL                      |
+----------------------------------------+
```

Иногда требуется не преобразовать строку из одной кодировки в другую при помощи функции CONVERT(), а указать ее кодировку: это осуществляется при помощи так называемого *представителя* — имени кодировки, начинающейся с символа подчеркивания (листинг 27.93).

Листинг 27.93. Альтернативное преобразование кодировки

```
SELECT _koi8r'База данных MySQL';
+------------------+
| аЮГЮ ДЮММШУ MySQL |
+------------------+
| аЮГЮ ДЮММШУ MySQL |
+------------------+
```

Имя кодировки, которое предваряется знаком подчеркивания, помещается перед преобразуемой строкой; результатом является строка в требуемой кодировке.

Первые несколько символов строки

Очень часто требуется выбрать из столбца не весь текст, а лишь несколько первых символов. Пусть используется таблица `tbl` (см. листинг 27.86), содержащая сведения о сотрудниках, из которой требуется извлечь первые 5 символов поля `fio`. Эту задачу удобно решать при помощи функции `SUBSTRING()` (листинг 27.94).

Листинг 27.94. Извлечение из столбца первых пяти символов

```
SELECT SUBSTRING(fio,1,5) FROM tbl;
+--------------------+
| SUBSTRING(fio,1,5) |
+--------------------+
| Тимир              |
| Мальш              |
| Абрам              |
| Ганюш              |
+--------------------+
```

Извлечение инициалов

Пусть имеется таблица `tbl`, которая содержит полные фамилию, имя и отчество сотрудников в трех разных столбцах (листинг 27.95).

Листинг 27.95. Таблица `tbl`

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  family TINYTEXT NOT NULL,
  name TINYTEXT NOT NULL,
  patronymic TINYTEXT NOT NULL
);
INSERT INTO tbl VALUES (1, 'Тимирязев', 'Владимир', 'Константинович');
INSERT INTO tbl VALUES (2, 'Мальшев', 'Юрий', 'Гаврилович');
```

```
INSERT INTO tbl VALUES (3, 'Абрамов', 'Константин', 'Тимурович');
INSERT INTO tbl VALUES (4, 'Ганюшкин', 'Валерий', 'Владимирович');
```

Пусть требуется извлечь только фамилию и инициалы, т. е. вместо "Тимирязев Владимир Константинович" получить строку "Тимирязев В. К.". Решить данную задачу можно при помощи запроса, представленного в листинге 27.96.

Листинг 27.96. Извлечение инициалов работников из таблицы `tbl`

```
SELECT CONCAT(family, ' ',
              SUBSTRING(name,1,1), '. ',
              SUBSTRING(patronymic,1,1), '.') AS fio
FROM tbl;
+-----------------+
| fio             |
+-----------------+
| Тимирязев В. К. |
| Малышев Ю. Г.   |
| Абрамов К. Т.   |
| Ганюшкин В. В.  |
+-----------------+
```

Как видно из листинга 27.96, инициалы извлекаются при помощи функции `SUBSTRING()`, а полученный результат объединяется в одну строку при помощи функции `CONCAT()`.

Изменение регистра строки

Пусть требуется перевести все строки таблицы `tbl` (см. листинг 27.95) в верхний регистр. Для этого можно воспользоваться запросом, представленным в листинге 27.97.

Листинг 27.97. Преобразование строк в верхний регистр

```
UPDATE tbl SET family = UPPER(family),
               name = UPPER(name),
               patronymic = UPPER(patronymic);
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | ТИМИРЯЗЕВ | ВЛАДИМИР   | КОНСТАНТИНОВИЧ |
|  2 | МАЛЬШЕВ   | ЮРИЙ       | ГАВРИЛОВИЧ     |
|  3 | АБРАМОВ   | КОНСТАНТИН | ТИМУРОВИЧ      |
|  4 | ГАНЮШКИН  | ВАЛЕРИЙ    | ВЛАДИМИРОВИЧ   |
+----+-----------+------------+----------------+
```

Как видно из листинга 27.97, преобразование строки в верхний регистр осуществляется при помощи функции `UPPER()`; для решения обратной задачи предназначена функция `LOWER()` (листинг 27.98).

Листинг 27.98. Преобразование строк в нижний регистр

```
UPDATE tbl SET family = LOWER(family),
               name = LOWER(name),
               patronymic = LOWER(patronymic);
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | тимирязев | владимир   | константинович |
|  2 | малышев   | юрий       | гаврилович     |
|  3 | абрамов   | константин | тимурович      |
|  4 | ганюшкин  | валерий    | владимирович   |
+----+-----------+------------+----------------+
```

В листинге 27.99 в верхний регистр переводится лишь первая буква каждой строки таблицы `tbl`.

Листинг 27.99. Перевод первого символа в верхний регистр

```
UPDATE tbl SET
family = CONCAT(UPPER(SUBSTRING(family,1,1)),SUBSTRING(family,2)),
name = CONCAT(UPPER(SUBSTRING(name,1,1)),SUBSTRING(name,2)),
patronymic = CONCAT(UPPER(SUBSTRING(patronymic,1,1)),
                    SUBSTRING(patronymic,2));
SELECT * FROM tbl;
+----+-----------+------------+----------------+
| id | family    | name       | patronymic     |
+----+-----------+------------+----------------+
|  1 | Тимирязев | Владимир   | Константинович |
|  2 | Малышев   | Юрий       | Гаврилович     |
|  3 | Абрамов   | Константин | Тимурович      |
|  4 | Ганюшкин  | Валерий    | Владимирович   |
+----+-----------+------------+----------------+
```

27.23.4. Функции шифрования

Различают два вида шифрования: обратимое и необратимое. При обратимом шифровании зашифрованный текст может быть подвергнут дешифровке. При необратимом

шифровании результатом является хэш, который невозможно восстановить в исходный текст. В табл. 27.16 представлены функции шифрования, доступные в СУБД MySQL.

ЗАМЕЧАНИЕ

Если функции передается в качестве одного из аргументов значение `NULL`, она также возвращает `NULL`.

Таблица 27.16. *Функции шифрования*

Функция	Описание
`AES_ENCRYPT(str, key_str)`	Принимает в качестве аргумента *str* строку, которую необходимо подвергнуть шифрованию, и секретный ключ *key_str*. Входные аргументы могут быть любой длины
`AES_DECRYPT(crypt_str, key_str)`	Принимает в качестве первого аргумента зашифрованную при помощи `AES_ENCRYPT()` строку. Ключ *key_str* при этом должен совпадать как в первой, так и во второй строках. Если функция `AES_DECRYPT()` обнаруживает некорректные данные или некорректное заполнение строки, должно возвращаться значение `NULL`. Однако `AES_DECRYPT()` вполне может вернуть величину, отличную от `NULL`, или, возможно, просто "мусор"
`ENCODE(str, pass_str)`	Шифрует строку *str*, используя аргумент *pass_str* как секретный ключ
`DECODE(crypt_str, pass_str)`	Дешифрует строку *crypt_str*, зашифрованную функцией `AES_DECRYPT()`. Аргумент *pass_str* используется как секретный ключ
`DES_ENCRYPT(str [, (key_number | key_string) ])`	Шифрует строку *str*, используя аргумент *pass_str* как секретный ключ. В качестве второго необязательного параметра может выступать строка *key_string*, задающая секретный ключ. В случае использования секретного ключа необходимо приводить его в качестве второго параметра и в функции дешифровки `DES_DECRYPT()`. Вместо секретного ключа можно указать номер *key_number*, принимающий значения от 0 до 9. Номер указывает на запись в ключевом DES-файле сервера, местоположение которого можно задать при старте сервера MySQL в параметре `--des-key-file`
`DES_DECRYPT(str [, key_string])`	Дешифрует строку *str*, зашифрованную при помощи функции `DES_ENCRYPT()`. Если при шифровании в качестве второго параметра функции `DES_ENCRYPT()` было передано число или второй параметр был опущен, то параметр *key_string* функции `DES_DECRYPT()` указывать уже не требуется, т. к. он прописывается в зашифрованную строку. Такой подход, когда секретный ключ хранится на сервере и не передается через сетевое соединение, значительно безопаснее, поскольку значение ключа невозможно извлечь из сетевого трафика. Если второй параметр не указывается, то предполагается, что используется первая строка DES-файла

Таблица 27.16 *(окончание)*

Функция	Описание
ENCRYPT(*str*[, *salt*])	Подвергает строку *str* необратимому шифрованию, используя вызов системной функции crypt() UNIX. Если второй необязательный параметр *salt* не указывается, то результат каждый раз получается новым
MD5(*str*)	Принимает строковый параметр *str* и возвращает 128-битную контрольную сумму, вычисленную по алгоритму MD5. Возвращаемая величина представляет собой 32-разрядное шестнадцатеричное число, которое уникально для строки, таким образом для строк, отличающихся хотя бы одним символом, результат функции MD5() будет разный. В то же время для двух одинаковых строк всегда возвращается одинаковый результат
PASSWORD(*str*)	Подвергает необратимому шифрованию данные *str*. Именно эта функция используется для шифрования паролей в MySQL
OLD_PASSWORD(*str*)	Эмулирует работу функции PASSWORD() версий MySQL, предшествующих MySQL 4.1
SHA1(*str*)	Вычисляет 160-битную контрольную сумму по алгоритму SHA1 (Secure Hash Algorithm) для строки *str*

Обратимое шифрование

При обратимом шифровании, как правило, предоставляются две функции для шифровки и расшифровки данных. Наиболее серьезную защиту обеспечивают функции AES_ENCRYPT(*str*, *key_str*) и AES_DECRYPT(*crypt_str*, *key_str*), которые используют официальный алгоритм AES (Advanced Encryption Standard). В данном алгоритме применяется кодирование с 128-битным ключом, однако при помощи добавки (патча, patch) к исходному коду СУБД MySQL длину ключа можно увеличить до 256 битов.

Функция AES_ENCRYPT() принимает в качестве аргумента *str* строку, которую необходимо подвергнуть шифрованию, и секретный ключ *key_str*. Входные аргументы могут быть любой длины. Если один из аргументов равен NULL, то результат этой функции также будет иметь значение NULL.

Функция AES_DECRYPT() принимает в качестве первого аргумента зашифрованную при помощи AES_ENCRYPT() строку. Ключ *key_str* при этом должен совпадать как в первой, так и во второй строках. Если функция AES_DECRYPT() обнаруживает некорректные данные или некорректное заполнение строки, должно возвращаться значение NULL, однако она вполне может вернуть величину, отличную от NULL, или, возможно, просто "мусор".

В листинге 27.100 приводится пример использования функций AES_ENCRYPT() и AES_DECRYPT().

Листинг 27.100. Использование функций `AES_ENCRYPT()` и `AES_DECRYPT()`

```
SELECT AES_ENCRYPT('MySQL','секретный ключ');
+--------------------------------------+
| AES_ENCRYPT('MySQL','секретный ключ') |
+--------------------------------------+
| ...ЗАьZЭj9к•-еП™Mћ                    |
+--------------------------------------+
SELECT AES_DECRYPT('...ЗАьZЭj9к•-еП™Mћ','секретный ключ');
+-------------------------------------------------+
| AES_DECRYPT('...ЗАьZЭj9к•-еП™Mћ','секретный ключ') |
+-------------------------------------------------+
| MySQL                                           |
+-------------------------------------------------+
```

Необратимое шифрование

Функция `MD5()` (листинг 27.101) осуществляет необратимое шифрование данных по алгоритму MD5 (Message-Digest Algorithm) и имеет следующий синтаксис:

`MD5(`*`str`*`)`

Функция принимает строковый параметр *`str`* и возвращает 128-битную контрольную сумму, вычисленную по алгоритму MD5. Возвращаемая величина представляет собой 32-разрядное шестнадцатеричное число, которое уникально для строки, то есть для строк, отличающихся хотя бы одним символом, результат функции `MD5()` будет разный. В то же время для двух одинаковых строк всегда возвращается одинаковый результат (листинг 27.101).

Листинг 27.101. Использование функции `MD5()`

```
SELECT MD5('MySQL'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL')                     | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| 62a004b95946bb97541afa471dcca73a | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
SELECT MD5('MySQL1'), MD5('MySQL');
+----------------------------------+----------------------------------+
| MD5('MySQL1')                    | MD5('MySQL')                     |
+----------------------------------+----------------------------------+
| fc3dd4ddcb132de1f9552818344e1b09 | 62a004b95946bb97541afa471dcca73a |
+----------------------------------+----------------------------------+
```

Алгоритм MD5 часто применяется также для создания уникального хэш-кода объемных файлов, которые передаются по сети. Загрузив файл, всегда можно проверить его

целостность, вычислив код по алгоритму MD5 и сравнив полученный результат с хэш-кодом, предоставляемым распространителем. Это позволяет отследить повреждения файла, вызванные передачей через сеть, а также предотвращает фальсификацию дистрибутива.

Необратимое шифрование может применяться также для шифрования паролей — в базе данных хранятся не сами пароли, а их MD5-хэши. Во время аутентификации пароль, введенный пользователем, также подвергается обработке функцией `MD5()`, после чего MD5-хэши подвергаются сравнению. Такая схема позволяет хранить пароли в защищенном виде, и никто, даже администратор базы данных, не имеет возможности узнать пользовательские пароли. В последнее время такой вид аутентификации приобретает все большую популярность. Поскольку вследствие применения паузы после ввода некорректного пароля время, необходимое для подбора пароля, многократно возросло, средства подбора единичных паролей по сети практически вышли из употребления, — злоумышленники стараются получить всю базу с паролями. Если же в базе хранятся только MD5-хэши, злоумышленнику придется восстанавливать методом перебора каждый из зашифрованных паролей.

27.23.5. Агрегатные функции

Функции, применяемые совместно с конструкцией `GROUP BY`, часто называют *агрегатными* или *суммирующими функциями*. Они предназначены для вычисления одного значения для каждой группы, создаваемой конструкцией `GROUP BY`. Агрегатные функции позволяют определить количество строк, входящих в группу, подсчитать среднее значение или получить сумму значений столбцов. Результирующее значение рассчитывается только для значений, не равных `NULL` (исключение составляет лишь функция `COUNT(*)`, которая подсчитывает общее количество строк). Данные функции допустимо использовать и в запросах без группировки: в этом случае вся выборка выступает как одна большая группа. В табл. 27.17 представлен список агрегатных функций, доступных в MySQL.

Таблица 27.17. *Агрегатные функции MySQL*

Функция	Описание
`AVG(`*expr*`)`	Возвращает среднее значение аргумента *expr*
`BIT_AND(`*expr*`)`	Возвращает побитовое И для всех битов в *expr*
`BIT_OR(`*expr*`)`	Возвращает побитовое ИЛИ для всех битов в *expr*
`BIT_XOR(`*expr*`)`	Возвращает исключающее побитовое ИЛИ для всех битов в *expr*
`COUNT(`*expr*`)`, `COUNT(*)`, `COUNT(DISTINCT` *expr1*, *expr2*, ...`)`	Подсчитывает количество записей в *expr*

Таблица 27.17 *(окончание)*

Функция	Описание
GROUP_CONCAT(*expr*)	Объединяет значения отдельных групп, полученных в результате применения конструкции GROUP BY, в одну строку
MIN([DISTINCT] *expr*)	Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr*
MAX([DISTINCT] *expr*)	Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr*
STD(*expr*), STDDEV(*expr*), STDDEV_POP(*expr*)	Возвращает стандартное среднеквадратичное отклонение значения в аргументе *expr*
STDDEV_SAMP()	Возвращает выборочное среднеквадратическое отклонение *expr*
SUM([DISTINCT] *expr*)	Возвращает сумму величин в столбце *expr*
VAR_POP(*expr*), VARIANCE(*expr*)	Возвращает стандартное отклонение значения в столбце *expr*
VAR_SAMP(*expr*)	Возвращает выборочное отклонение значения в аргументе *expr*

Среднее значение

Функция AVG() возвращает среднее значение аргумента *expr* и имеет следующий синтаксис:

```
AVG([DISTINCT] expr)
```

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово DISTINCT позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

> ***Замечание***
>
> Ключевое слово DISTINCT в функции AVG() добавлено, начиная с версии 5.0.3.

В листинге 27.102 представлена таблица catalogs, состоящая из трех полей: id_catalog — первичный ключ, name — название раздела и total — количество единиц данного товара на складе.

Листинг 27.102. Таблица catalogs

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL auto_increment,
  name TINYTEXT NOT NULL,
  total INT(11) NOT NULL,
  PRIMARY KEY (id_catalog)
```

```
);
INSERT INTO catalogs VALUES (1, 'Процессоры', 15);
INSERT INTO catalogs VALUES (2, 'Материнские платы', 4);
INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 7);
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 18);
INSERT INTO catalogs VALUES (5, 'Оперативная память', 9);
```

В листинге 27.103 приводится запрос к таблице `catalogs`, который возвращает среднее количество товаров на складе.

Листинг 27.103. Среднее количество товарных позиций на складе

```
SELECT AVG(total) FROM catalogs;
+------------+
| AVG(total) |
+------------+
|    10.6000 |
+------------+
```

При использовании конструкции `GROUP BY` функция `AVG()` вычисляет среднее значение для каждой из группы. Создадим таблицу `products` (листинг 27.104), в которой хранится информация обо всех имеющихся товарах. Таблица содержит следующие поля:

- `id_product` — первичный ключ;
- `name` — название товара;
- `price` — цена за единицу товара;
- `id_catalog` — ссылка на раздел каталога, к которому относится товар.

Листинг 27.104. Таблица `products`

```
CREATE TABLE products (
  id_product INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  price decimal(7,2) NOT NULL,
  id_catalog int(11) NOT NULL,
  PRIMARY KEY  (id_product),
  KEY id_catalog (id_catalog)
);
INSERT INTO products VALUES (1, 'Материнская плата N 1', '2050.00', 2);
INSERT INTO products VALUES (2, 'Оперативная память N 1', '1500.00', 5);
INSERT INTO products VALUES (3, 'Процессор N 1', '6000.00', 1);
INSERT INTO products VALUES (4, 'Процессор N 2', '3523.00', 1);
INSERT INTO products VALUES (5, 'Процессор N 3', '4856.00', 1);
```

```
INSERT INTO products VALUES (6, 'Оперативная память N 2', '1028.00', 5);
INSERT INTO products VALUES (7, 'Материнская плата N 2', '1845.00', 2);
INSERT INTO products VALUES (8, 'Жесткий диск N 1', '4210.00', 4);
INSERT INTO products VALUES (9, 'Видеоадаптер N 1', '3670.00', 3);
```

Если сгруппировать записи таблицы `products` по полю `id_catalog`, при помощи встроенной функции `AVG()` можно узнать среднюю цену по каждому из пяти разделов каталога. В листинге 27.105 выводится результирующая таблица с уникальными номерами `id_catalog` и соответствующие каждому разделу каталога средние цены.

Листинг 27.105. Использование функции `AVG()` совместно с `GROUP BY`

```
SELECT id_catalog, AVG(price) FROM products GROUP BY id_catalog;
+------------+-------------+
| id_catalog | AVG(price)  |
+------------+-------------+
|          1 | 4793.000000 |
|          2 | 1947.500000 |
|          3 | 3670.000000 |
|          4 | 4210.000000 |
|          5 | 1264.000000 |
+------------+-------------+
```

Средние значения, полученные при помощи функции `AVG()`, могут использоваться в вычисляемых столбцах. Например, для того чтобы увеличить средние значения для каждого раздела каталога на 20%, достаточно умножить либо столбец `price`, либо функцию `AVG()` на 1.2 (листинг 27.106).

Листинг 27.106. Использование функции `AVG()` в выражениях

```
SELECT id_catalog, AVG(price*1.2) FROM products
GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price*1.2) |
+------------+----------------+
|          1 |   5751.6000000 |
|          2 |   2337.0000000 |
|          3 |   4404.0000000 |
|          4 |   5052.0000000 |
|          5 |   1516.8000000 |
+------------+----------------+
5 rows in set (0.02 sec)
SELECT id_catalog, AVG(price)*1.2 FROM products
```

```
GROUP BY id_catalog;
+------------+----------------+
| id_catalog | AVG(price*1.2) |
+------------+----------------+
|          1 |   5751.6000000 |
|          2 |   2337.0000000 |
|          3 |   4404.0000000 |
|          4 |   5052.0000000 |
|          5 |   1516.8000000 |
+------------+----------------+
5 rows in set (0.00 sec)
```

В первом случае сначала каждое из значений столбца `price` умножается на 1.2, а затем производится группировка и вычисление результата при помощи функции `AVG()`. Во втором сначала осуществляется группировка значений и вычисление средних значений, и лишь после этого результат умножается на 1.2. Как следствие, первый запрос выполняется 0.02 секунды, а второй — 0.00 секунды.

Сортировка агрегатных значений

По полученным в результате выполнения агрегатных функций значениям может выполняться сортировка результирующей таблицы. Для этого столбцу назначается псевдоним при помощи оператора `AS`, который передается конструкции `ORDER BY` (листинг 27.107).

Листинг 27.107. Сортировка результирующей таблицы

```
SELECT id_catalog, AVG(price) AS price FROM products
GROUP BY id_catalog ORDER BY price DESC;
+------------+-------------+
| id_catalog | price       |
+------------+-------------+
|          1 | 4793.000000 |
|          4 | 4210.000000 |
|          3 | 3670.000000 |
|          2 | 1947.500000 |
|          5 | 1264.000000 |
+------------+-------------+
```

Подсчет количества записей в таблице

Подсчет количества записей в таблице осуществляется при помощи функции `COUNT()`, имеющей несколько форм со следующим синтаксисом:

```
COUNT(expr)
COUNT(*)
COUNT(DISTINCT expr1, expr2, ...)
```

Первая форма возвращает количество записей в таблице, поле *expr* для которых не равно NULL. В листинге 27.108 представлен дамп таблицы tbl.

Листинг 27.108. Дамп таблицы tbl

```
CREATE TABLE tbl (
  id int(11) NOT NULL,
  value int(11) default NULL
);
INSERT INTO tbl VALUES (1, 230);
INSERT INTO tbl VALUES (2, NULL);
INSERT INTO tbl VALUES (3, 405);
INSERT INTO tbl VALUES (4, NULL);
```

В листинге 27.109 представлен запрос, возвращающий количество записей в таблице tbl.

Листинг 27.109. Использование функции COUNT()

```
SELECT * FROM tbl;
+----+-------+
| id | value |
+----+-------+
|  1 |   230 |
|  2 |  NULL |
|  3 |   405 |
|  4 |  NULL |
+----+-------+
SELECT COUNT(id), COUNT(value) FROM tbl;
+-----------+--------------+
| COUNT(id) | COUNT(value) |
+-----------+--------------+
|         4 |            2 |
+-----------+--------------+
```

Как видно из листинга 27.109, для полей id и value возвращаются различные значения. Это связано с тем, что количество NULL-полей в столбцах различается.

Форма функции COUNT(*) возвращает общее количество строк в таблице, независимо от того, принимает какое-либо поле значение NULL или нет (листинг 27.110). Запись учитывается в результате, даже если все ее поля равны NULL.

Листинг 27.110. Использование функции `COUNT(*)`

```
SELECT COUNT(*) FROM tbl;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Функция `COUNT(*)` оптимизирована для очень быстрого возврата результата при условии, что команда `SELECT` извлекает данные из одной таблицы, никакие другие столбцы не обрабатываются и запрос не содержит условия `WHERE`.

Функция `COUNT()` может быть использована не только для подсчета общего количества записей в таблице, но и для подсчета количества строк в выборке с условием `WHERE` (листинг 27.111).

Листинг 27.111. Использование функции `COUNT()` совместно с `WHERE`

```
SELECT COUNT(*) FROM tbl WHERE value < 300;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

В листинге 27.111 представлен запрос, извлекающий из таблицы `tbl` записи, чье поле `value` меньше 300. Следует обратить внимание, что поля, содержащие `NULL`, не удовлетворяют этому условию, т. к. `NULL` обозначается отсутствие данных (поле может быть как больше, так и меньше 300 — информация об этом не известна).

Разумеется, как агрегатная функция `COUNT()` может быть использована для вычисления количества записей в каждой из групп, полученных в результате применения группировки результата с помощью конструкции `GROUP BY`.

В листинге 27.112 представлен запрос к таблице `products`, сообщающий о количестве имеющегося товара по каждому разделу каталога.

Листинг 27.112. Подсчет количества товара для каждого раздела каталога

```
SELECT id_catalog, COUNT(*) AS total FROM products
GROUP BY id_catalog ORDER BY total DESC;
+------------+-------+
| id_catalog | total |
+------------+-------+
|          1 |     3 |
```

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

Сортировка строк в результирующей таблице производится по количеству имеющегося товара. Для этого столбцу `COUNT(*)` при помощи ключевого слова `AS` назначается псевдоним `total`, который затем используется в выражении `ORDER BY`.

Третий вариант функции `COUNT()` позволяет использовать ключевое слово `DISTINCT`, которое обеспечивает подсчет только уникальных значений столбца (листинг 27.113).

Листинг 27.113. Использование ключевого слова `DISTINCT` совместно с `COUNT()`

```
SELECT COUNT(id_catalog), COUNT(DISTINCT id_catalog) FROM products;
+------------------+----------------------------+
| COUNT(id_catalog) | COUNT(DISTINCT id_catalog) |
+------------------+----------------------------+
|                9 |                          5 |
+------------------+----------------------------+
```

Объединение значений группы

Для объединения значения группы предназначена функция `GROUP_CONCAT()`, которая имеет следующий синтаксис:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
             [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

В простейшем случае функция принимает имя столбца *expr* и возвращает строку со значениями столбца, разделенными запятыми. В листинге 27.114 приводится пример использования функции `GROUP_CONCAT()` для поля `id_catalog` таблицы `products` (листинг 27.104).

Листинг 27.114. Использование функции `GROUP_CONCAT()`

```
SELECT GROUP_CONCAT(id_catalog) FROM products;
+-------------------------+
| GROUP_CONCAT(id_catalog) |
+-------------------------+
| 1,1,1,2,2,3,4,5,5       |
+-------------------------+
```

При передаче в качестве аргумента имени числового столбца его значения автоматически преобразуются к текстовому типу.

Ключевое слово `DISTINCT` требует вернуть только уникальные значения столбца, а ключевое слово `SEPARATOR` позволяет задать в качестве разделителя значений произвольный символ. В листинге 27.115 представлен запрос, извлекающий уникальные значения столбца `id_catalog` таблицы `products` с использованием в качестве разделителя символа `-`.

Листинг 27.115. Использование ключевых слов `DISTINCT` и `SEPARATOR`

```
SELECT GROUP_CONCAT(DISTINCT id_catalog SEPARATOR '-')
FROM catalogs;
+-----------------------------------------------+
| GROUP_CONCAT(DISTINCT id_catalog SEPARATOR '-') |
+-----------------------------------------------+
| 1-2-3-4-5                                     |
+-----------------------------------------------+
```

Ключевое слово `ORDER BY` позволяет отсортировать значения в рамках возвращаемой строки. Отсортируем значения, представленные в листинге 27.115, в обратном порядке (листинг 27.116).

Листинг 27.116. Использование ключевого слова `ORDER BY`

```
SELECT GROUP_CONCAT(DISTINCT id_catalog
                    ORDER BY id_catalog
                    DESC SEPARATOR '-') AS str
FROM catalogs;
+-----------+
| str       |
+-----------+
| 5-4-3-2-1 |
+-----------+
```

В листинге 27.117 возвращается список цен (в порядке убывания) на товары из каждого раздела каталога.

Листинг 27.117. Список цен на товары по разделам каталога

```
SELECT GROUP_CONCAT(price ORDER BY price DESC)
FROM products GROUP BY id_catalog;
```

```
+---------------------------------------+
| GROUP_CONCAT(price ORDER BY price DESC) |
+---------------------------------------+
| 6000.00,4856.00,3523.00               |
| 2050.00,1845.00                       |
| 3670.00                               |
| 4210.00                               |
| 1500.00,1028.00                       |
+---------------------------------------+
```

Поиск минимального и максимального значений

Для поиска минимального значения в столбце *expr* предназначена функция `MIN()`, которая имеет следующий синтаксис:

```
MIN([DISTINCT] expr)
```

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово `DISTINCT` позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

В листинге 27.118 приводится пример запроса, который ищет в таблице `products` (листинг 27.104) минимальную цену.

Листинг 27.118. Использование функции `MIN()`

```
SELECT MIN(price) FROM products;
+------------+
| MIN(price) |
+------------+
|    1028.00 |
+------------+
```

Использование конструкции `GROUP BY id_catalog` позволяет найти минимальную цену для каждого из разделов каталога (листинг 27.119).

Листинг 27.119. Использование функции `MIN()` совместно с конструкцией `GROUP BY`

```
SELECT id_catalog, MIN(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MIN(price) |
+------------+------------+
|          1 |    3523.00 |
|          2 |    1845.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    1028.00 |
+------------+------------+
```

Как видно из результатов запроса, товар с минимальной ценой находится в пятом разделе каталога.

> ***ЗАМЕЧАНИЕ***
>
> Функцию `MIN()` можно использовать также со строковыми столбцами. В этом случае возвращается минимальное лексикографическое значение.

Для поиска максимального значения в столбце *expr* предназначена функция `MAX()`, которая имеет следующий синтаксис:

```
MAX([DISTINCT] expr)
```

В качестве аргумента *expr* обычно выступает имя столбца. Необязательное ключевое слово `DISTINCT` позволяет дать указание СУБД MySQL обрабатывать только уникальные значения столбца *expr*.

В листинге 27.120 демонстрируется использование функции `MAX()`.

Листинг 27.120. Использование функции `MAX()`

```
SELECT MAX(price) FROM products;
+------------+
| MAX(price) |
+------------+
|    6000.00 |
+------------+
SELECT id_catalog, MAX(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | MAX(price) |
+------------+------------+
|          1 |    6000.00 |
|          2 |    2050.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    1500.00 |
+------------+------------+
```

Сумма столбца

Сумму столбца *expr* позволяет подсчитать функция `SUM()`, которая имеет следующий синтаксис:

```
SUM([DISTINCT] expr)
```

Если возвращаемый набор данных не содержит ни одной строки, то функция возвращает `NULL`. Необязательное ключевое слово `DISTINCT` позволяет потребовать от СУБД MySQL обрабатывать только уникальные значения столбца *expr*. В листинге 27.121 приводится пример использования функции `SUM()`.

Листинг 27.121. Использование функции SUM()

```
SELECT SUM(price) FROM products;
+------------+
| SUM(price) |
+------------+
|   28682.00 |
+------------+
SELECT id_catalog, SUM(price) FROM products GROUP BY id_catalog;
+------------+------------+
| id_catalog | SUM(price) |
+------------+------------+
|          1 |   14379.00 |
|          2 |    3895.00 |
|          3 |    3670.00 |
|          4 |    4210.00 |
|          5 |    2528.00 |
+------------+------------+
```

27.23.6. Разное

В англоязычной литературе этот вид функций обозначается как miscellaneous-функции, что переводится как смешанные функции, т. е. функции, которые не поддаются классификации. Список функций данной категории представлен в табл. 27.18.

Таблица 27.18. *Miscellaneous-функции MySQL*

Функция	Описание
DEFAULT(*col_name*)	Возвращает значение по умолчанию для столбца *col_name*, которое назначается при помощи ключевого слова DEFAULT
GET_LOCK(*str*, *timeout*)	Предназначена для получения блокировки для имени *str* со временем ожидания ответа сервера *timeout* секунд и возвращает 1 (истина) в случае успешной установки блокировки на имя *str* и 0 (ложь), если время ожидания ответа превысило величину *timeout*
INET_ATON(*address*)	Принимает IP-адрес *address* и представляет его в виде целого числа
INET_NTOA(*address*)	Принимает IP-адрес в виде числа (результат выполнения функции INET_ATON()) и возвращает адрес в виде строки, состоящей из четырех чисел, разделенных точкой
IS_FREE_LOCK(*str*)	Проверяет, свободна ли блокировка с именем *str*, которая устанавливается при помощи функции GET_LOCK(), рассмотренной ранее. Функция возвращает 1 (истина), если блокировка свободна (никем не используется), и 0 (ложь), если занята

Таблица 27.18 *(окончание)*

Функция	Описание
IS_USED_LOCK(*str*)	Проверяет, установлена ли блокировка с именем *str*. В случае, если блокировка установлена, функция возвращает идентификатор соединения клиента, который удерживает блокировку. Если блокировка не установлена, возвращается NULL
NAME_CONST(*name*, *value*)	Возвращает значение параметра *value*, назначая в качестве имени столбца строку *name*
RELEASE_LOCK(*str*)	Снимает блокировку с именем *str*, которая была установлена функцией GET_LOCK(). Функция возвращает 1 (истина), если блокировка успешно снята, 0 (ложь), если блокировка установлена другим потоком и не может быть снята, и NULL, если блокировка с таким именем не существует
SLEEP(*duration*)	Останавливает работу на время, равное количеству секунд, указанному в параметре *duration* (значение параметра может быть как целым, так и дробным с точностью учитывается до микросекунд). В случае успешного выполнения функция возвращает значение 0, если работа функции прерывается, возвращается 1
UUID()	Возвращает универсальный уникальный идентификатор (Universal Unique Identifier, UUID). Идентификатор UUID реализован в виде числа, которое является глобально уникальным во времени и пространстве. Два вызова функции UUID() вернут два разных значения, если они производятся одновременно на двух разных компьютерах или на одном и том же компьютере в разное время

Преобразование IP-адреса

Функция INET_ATON(*address*) принимает IP-адрес *address* и представляет его в виде целого числа (листинг 27.122). Для числа XXX.YYY.ZZZ.WWW результат функции вычисляется по формуле:

$$\text{XXX} \times 256^3 + \text{YYY} \times 256^2 + \text{ZZZ} \times 256 + \text{WWW}.$$

Замечание

Для того чтобы можно было поместить в целочисленное поле весь диапазон IP-адресов, следует использовать тип BIGINT.

Листинг 27.122. Использование функции INET_ATON()

```
SELECT INET_ATON('62.145.69.10'), INET_ATON('127.0.0.1');
+--------------------------+-----------------------+
| INET_ATON('62.145.69.10') | INET_ATON('127.0.0.1') |
+--------------------------+-----------------------+
|               1049707786 |            2130706433 |
+--------------------------+-----------------------+
```

Начиная с версии 4.1.2, функция `INET_ATON()` способна принимать IP-адреса в сокращенной форме, как показано в листинге 27.123.

Листинг 27.123. Работа с сокращенной формой IP-адреса

```
SELECT INET_ATON('127.0.01'), INET_ATON('127.1');
+----------------------+--------------------+
| INET_ATON('127.0.01') | INET_ATON('127.1') |
+----------------------+--------------------+
|            2130706433 |         2130706433 |
+----------------------+--------------------+
```

Функция `INET_NTOA(address)` принимает IP-адрес в виде числа (результат выполнения функции `INET_ATON()`) и возвращает адрес в виде строки, состоящей из четырех чисел, разделенных точкой (листинг 27.124).

Листинг 27.124. Работа с функцией `INET_NTOA()`

```
SELECT INET_NTOA(1049707786), INET_NTOA(2130706433);
+----------------------+----------------------+
| INET_NTOA(1049707786) | INET_NTOA(2130706433) |
+----------------------+----------------------+
| 62.145.69.10         | 127.0.0.1            |
+----------------------+----------------------+
```



Глава 28

Взаимодействие MySQL и PHP

Язык программирования PHP является сравнительно молодым языком программирования, специально созданным для генерации динамических Web-страниц. Так сложилось, что связка Web-сервера Apache, базы данных MySQL и языка программирования PHP получила очень широкое распространение среди Web-разработчиков.

В настоящий момент PHP поддерживает две библиотеки доступа к СУБД MySQL:

- php_mysql — традиционная процедурная библиотека;
- php_mysqli — объектно-ориентированная библиотека.

В данной главе будут рассмотрены обе библиотеки, а также приемы взаимодействия PHP и MySQL.

28.1. Библиотека php_mysql

Функции библиотеки php_mysql позволяют устанавливать соединение с сервером MySQL, выполнять запрос и получать доступ к результирующей таблице.

28.1.1. Установка соединения с базой данных

В табл. 28.1 приведен список функций, ответственных за управление соединением с базой данных. Любой сеанс с базой данных начинается с открытия соединения с последующим его закрытием либо явно при помощи функции `mysql_close()`, либо в результате завершения работы скрипта.

***Таблица 28.1.** Функции управления соединением с сервером*

Функция	Описание
`mysql_connect()`	Открывает соединение с сервером
`mysql_close()`	Закрывает соединение с сервером
`mysql_ping()`	Проверяет соединение с сервером и повторно соединяется при необходимости

Соединение с базой данных MySQL осуществляется при помощи функции `mysql_connect()`, которая имеет следующий синтаксис:

```
mysql_connect ([$server [,
                $username [,
                $password [,
                $new_link [,
                $client_flags]]]]])
```

Эта функция устанавливает соединение с сервером MySQL, сетевой адрес которого задается параметром `$server`. Вторым и третьим аргументами этой функции являются имя пользователя базы данных `$username` и его пароль `$password`, соответственно.

По умолчанию повторный вызов функции `mysql_connect()` с теми же аргументами не приводит к установлению нового соединения, вместо этого функция возвращает дескриптор уже существующего соединения. Если четвертому параметру `$new_link` присвоить значение `TRUE`, будет открыто новое соединение с сервером. Параметр `$client_flags` должен быть комбинацией из следующих констант:

- `MYSQL_CLIENT_COMPRESS` — предписывает использование протокола сжатия при обмене информацией между сервером и клиентом;
- `MYSQL_CLIENT_IGNORE_SPACE` — в SQL-запросах после имен функций разрешается использование пробелов;
- `MYSQL_CLIENT_INTERACTIVE` — ждать `interactive_timeout` секунд до закрытия соединения, если между сервером и клиентом не происходит обмен данными.

> ***ЗАМЕЧАНИЕ***
>
> Все аргументы функции являются необязательными. В случае их отсутствия, по умолчанию, для этой функции устанавливаются следующие параметры: `server = 'localhost:3306'`, `username` принимает значение владельца процесса сервера, а `password` принимает пустую строку.

В случае успеха функция возвращает дескриптор соединения с сервером, при неудаче возвращает значение `FALSE`.

В листинге 28.1 приведен пример использования функции `mysql_connect()`.

Листинг 28.1. Функция `mysql_connect()`

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
```

```
  if (!$dbcnx) // Если дескриптор равен 0, соединение не установлено
  {
    exit ("<P>В настоящий момент сервер базы данных не доступен, поэтому
             корректное отображение страницы невозможно.</P>");
  }
  else
  {
    echo "<P>Соединение установлено.</P>";
  }
?>
```

ЗАМЕЧАНИЕ

Для подавления вывода сообщений об ошибках, генерируемых PHP в окно браузера, в листинге 28.1 перед функцией `mysql_connect()` размещен символ `@`.

Переменные `$dblocation`, `$dbuser` и `$dbpasswd` хранят имя сервера, имя пользователя и пароль и, как правило, прописываются в отдельном файле (к примеру, config.php), который потом вставляется в каждый PHP-файл. В последнем имеется код для работы с MySQL.

Начиная с MySQL версии 4.1, изменился порядок работы с кодировками — теперь перед началом работы необходимо выставить кодировки при помощи запроса `SET NAMES`, представленного в листинге 28.2. Иначе русский текст будет попадать в базу данных в виде знаков вопроса.

ЗАМЕЧАНИЕ

Функция `mysql_query()` рассматривается более подробно в *разд. 28.1.3*.

Листинг 28.2. Установка кодировки

```
<?php
  mysql_query ("SET NAMES 'cp1251'");
?>
```

Закрытие соединения осуществляется при помощи функции `mysql_close()`. Ее синтаксис таков:

```
mysql_close([$link_identifier])
```

В качестве необязательного параметра функция принимает дескриптор открытого соединения *`$link_identifier`*. Если этот параметр не указан, закрывается последнее открытое соединение. Функция возвращает `TRUE` в случае успеха и `FALSE` при возникновении ошибки.

В листинге 28.3 приведен пример использования функции.

Листинг 28.3. Закрытие соединения при помощи функции `mysql_close()`

```
<?php
  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx)
  {
    // Выводим предупреждение
    exit ("<P>В настоящий момент сервер базы данных не доступен, поэтому
    корректное отображение страницы невозможно.</P>");
  }
  if (mysql_close($dbcnx)) // закрываем соединение
  {
    echo "Соединение с базой данных прекращено";
  }
  else
  {
    echo "Не удалось завершить соединение";
  }
?>
```

Если скрипт выполняется непродолжительное время, явное закрытие соединения при помощи функции `mysql_close()` не обязательно, т. к. скрипт закрывает все соединения после своего завершения. Явное закрытие соединения требуется в том случае, если после выполнения всех запросов выполняется какое-то длительное действие, не дающее скрипту завершить свою работу, например, загрузка объемного файла, отдаваемого клиенту PHP-скриптом. В этом случае соединение будет оставаться занятым и бездействовать все время, пока клиент загружает скрипт.

28.1.2. Выбор базы данных

После того как соединение установлено, необходимо выбрать базу данных по умолчанию, с которой будет производиться работа. Для этого используется функция `mysql_select_db()`, вызов которой эквивалентен вызову команды `USE` в SQL-запросе. Функция имеет следующий синтаксис:

```
mysql_select_db($database_name [, $link_identifier])
```

Функция принимает в качестве аргументов название выбираемой базы данных *`$database_name`* и дескриптор соединения *`$link_identifier`*. Функция возвращает `TRUE` при успешном выполнении операции и `FALSE` в противном случае. В листинге 28.4 приводится пример использования данной функции.

Листинг 28.4. Использование функции `mysql_select_db()`

```
<?php
  // Код соединения с базой данных
  if (!@mysql_select_db($dbname, $dbcnx))
  {
    echo "<P>В настоящий момент база данных не доступна, поэтому
              корректное отображение страницы невозможно.</P>";
    exit();
  }
?>
```

Имеет смысл помещать функции для соединения и выбора базы данных в тот же файл (config.php), где объявлены переменные с именами сервера, пользователя и паролем (листинг 28.5), и подключать данный файл при помощи конструкции `include_once()` или `require_once()`, если требуется обращение к базе данных MySQL.

Листинг 28.5. Универсальный файл config.php

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "shop";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
             поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
              поэтому корректное отображение страницы невозможно.</P>" );
  }

  @mysql_query("SET NAMES 'cp1251'");
?>
```

28.1.3. Выполнение SQL-запросов

Выполнение SQL-запросов осуществляется при помощи функции `mysql_query()`, которая имеет следующий синтаксис:

```
mysql_query($query [, $link_identifier])
```

Первый аргумент функции представляет собой строку с запросом `$query`, второй, `$link_identifier` — дескриптор соединения, возвращаемый функцией `mysql_connect()`.

> ***ЗАМЕЧАНИЕ***
>
> Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции `mysql_connect()` без параметров.

> ***ЗАМЕЧАНИЕ***
>
> При передаче запроса функции `mysql_query()` точку с запятой в конце запроса, обязательную при работе с клиентом `mysql`, можно не ставить.

Функция возвращает дескриптор результирующей таблицы в случае успеха и `FALSE` в случае неудачного выполнения запроса. Если запрос не возвращает результирующей таблицы, можно не сохранять дескриптор в отдельной переменной. В листинге 28.6 представлен скрипт, создающий новую таблицу `catalogs`.

Листинг 28.6. Использование функции `mysql_query()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";
  // Формируем и выполняем SQL-запрос
  $query = "CREATE TABLE catalogs (
             id_catalog INT(11) NOT NULL AUTO_INCREMENT,
             name TINYTEXT NOT NULL,
             PRIMARY KEY (id_catalog))";
  if (mysql_query($query))
  {
     echo "Таблица создана успешно";
  }
  else
  {
     exit(mysql_erorr());
  }
?>
```

В листинге 28.6 при неудачном выполнении функции `mysql_query()` функции `exit()`, останавливающей работу скрипта, в качестве параметра передается значение ошибки, возвращаемое функцией `mysql_erorr()`.

ЗАМЕЧАНИЕ

По аналогии с API C, PHP поддерживает также функцию `mysql_errno()`, которая возвращает номер ошибки.

Дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`, используется далее для получения значений возвращаемых СУБД. Обычно это осуществляется при помощи одной из пяти функций: `mysql_result()`, `mysql_fetch_row()`, `mysql_fetch_assoc()`, `mysql_fetch_array()` и `mysql_fetch_object()`.

В примерах следующих разделов будет использоваться таблица `catalogs`, дамп которой представлен в листинге 28.7.

Листинг 28.7. Дамп таблицы `catalogs`

```
CREATE TABLE catalogs (
  id_catalog int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_catalog),
  FULLTEXT KEY `name` (`name`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
INSERT INTO catalogs VALUES (1,'Процессоры');
INSERT INTO catalogs VALUES (2,'Материнские платы');
INSERT INTO catalogs VALUES (3,'Видеоадаптеры');
INSERT INTO catalogs VALUES (4,'Жесткие диски');
INSERT INTO catalogs VALUES (5,'Оперативная память');
```

28.1.4. Получение результатов запроса

Функция `mysql_query()` сама по себе не возвращает результатов запроса, она лишь возвращает дескриптор, при помощи которого можно получить доступ к результирующей таблице посредством одной из функций, представленных в табл. 28.2.

***Таблица 28.2.** Таблицы доступа к результирующей таблице*

Функция	Описание
`mysql_result()`	Доступ к единичному значению из результирующей таблицы
`mysql_fetch_row()`	Доступ к строке результирующей таблицы и представление результатов в виде числового массива
`mysql_fetch_assoc()`	Доступ к строке результирующей таблицы и представление результатов в виде ассоциативного массива
`mysql_fetch_array()`	Доступ к строке результирующей таблицы и представление результатов в виде смешанного массива
`mysql_fetch_object()`	Доступ к строке результирующей таблицы и представление результатов в виде массива

Первая функция `mysql_result()` возвращает результат запроса, выполненного функцией `mysql_query()`. С ее помощью можно получить доступ к отдельному полю записи. Синтаксис функции представлен далее:

```
mysql_result($result, $row [, $field])
```

В качестве первого аргумента *`$result`* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Второй аргумент *`$row`* представляет собой номер столбца, который необходимо вернуть. Третий необязательный параметр *`$field`* — это имя поля таблицы. В листинге 28.8 при помощи данной функции выводится название первого каталога в таблице `catalogs` учебной базы данных `shop`.

Листинг 28.8. Использование функции `mysql_result()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM catalogs WHERE id_catalog = 1";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Обрабатываем результаты запроса
  if ($cat) echo mysql_result($cat, 0, 'name');
  else exit(mysql_error());
?>
```

Результат работы скрипта из листинга 28.8 представлен на рис. 28.1.

Функция `mysql_fetch_row()` возвращает текущую запись в результирующей таблице в виде неассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_row()` имеет следующий синтаксис:

```
mysql_fetch_row($result)
```

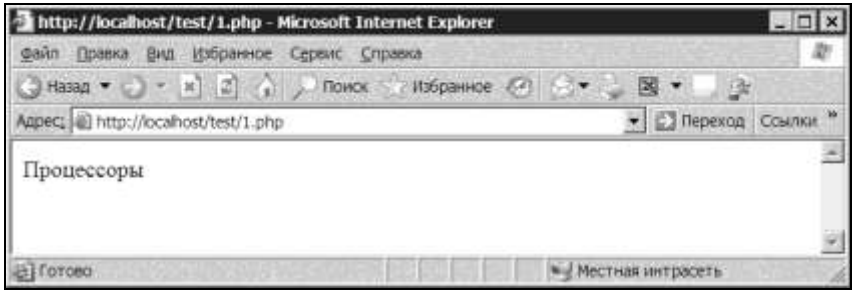


Рис. 28.1. Результат работы скрипта из листинга 28.8

В качестве единственного аргумента `$result` функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. При работе с массивом, который возвращает функция `mysql_fetch_row()`, удобно использовать конструкцию `list()`, преобразующую элементы массива в переменные. В листинг 28.9 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

Листинг 28.9. Применение функции `mysql_fetch_row()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit(mysql_erorr());
  // Так как запрос может возвращать несколько строк, применяем цикл
  echo "<table border=1>";
  while(list($id_author, $name) = mysql_fetch_row($cat))
  {
    echo "<tr>
            <td>$id_author</td>
            <td>$name</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 28.9 представлен на рис. 28.2.

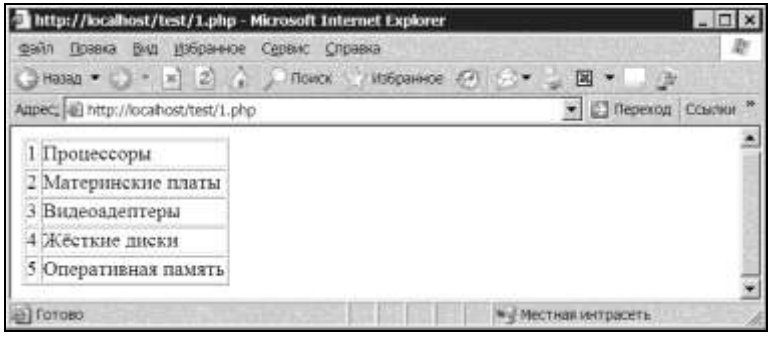


Рис. 28.2. Результат работы скрипта из листинга 28.9

Функция `mysql_fetch_assoc()` возвращает текущую запись в результирующей таблице в виде ассоциативного массива. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_assoc()` имеет следующий синтаксис:

```
mysql_fetch_assoc($result)
```

В качестве единственного аргумента *$result* функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`.Функция возвращает массив, каждый элемент которого соответствует одному полю в записи, или `FALSE`, если курсор достиг конца результирующей таблицы. Причем в качестве ключей массива выступают имена полей в результирующей таблице. В листинг 28.10 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

Листинг 28.10. Использование функции `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit(mysql_error());
  // Формируем таблицу
  echo "<table border=1>";
  while($catalogs = mysql_fetch_assoc($cat))
  {
    echo "<tr>
            <td>".$catalogs['id_catalog']."</td>
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Результат работы скрипта из листинга 28.10 выглядит точно так же, как результат работы скрипта из листинга 28.9 (см. рис. 28.2). Следует обратить внимание, что на каждой итерации цикла `while` формируется массив `$catalogs`, в качестве ключей которого выступают имена столбцов из таблицы `catalogs`. Если столбец является вычисляемым, то, для того чтобы обратиться к нему, потребуется полное обращение к имени столбца. Так в листинге 28.11 при помощи встроенной функции MySQL `VERSION()` извлекается версия сервера MySQL. При этом в качестве ключа массива выступает строка `'VERSION()'`.

Листинг 28.11. Получение версии сервера с использованием `mysql_fetch_assoc()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT VERSION()";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$ver) exit(mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['VERSION()'];
?>
```

Использование в качестве ключей полной сигнатуры вычисляемого столбца не всегда удобно, т. к. ключ может получиться громоздким или динамически изменяться. Для упрощения имени столбца в этом случае прибегают к назначению псевдонима при помощи ключевого слова `AS`. Поэтому скрипт из листинга 28.11 можно переписать так, как показано в листинге 28.12.

Листинг 28.12. Использование псевдонима, назначаемого при помощи оператора `AS`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT VERSION() AS ver";
  // Выполняем SQL-запрос
  $ver = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$ver) exit(mysql_error());
  // Определяем таблицу и заголовок
  $version = mysql_fetch_assoc($ver);
  echo $version['ver'];
?>
```

Функция `mysql_fetch_array()` возвращает текущую запись в результирующей таблице в виде массива. Причем тип массива (численный или ассоциативный) может быть

выбран. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_array()` имеет такой синтаксис:

```
mysql_fetch_array($result [, $result_type])
```

В качестве первого аргумента `$result` функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`. Второй необязательный параметр может принимать три значения:

- `MYSQL_ASSOC` — возврат результата работы в виде ассоциативного массива;
- `MYSQL_NUM` — возврат результата работы в виде численного массива;
- `MYSQL_BOTH` — возврат результата работы в виде массива, содержащего как численные, так и ассоциативные индексы.

> ***ЗАМЕЧАНИЕ***
>
> По умолчанию второй аргумент принимает значение `MYSQL_BOTH`.

> ***ЗАМЕЧАНИЕ***
>
> Режим работы функции `mysql_fetch_array()`, принимающей в качестве второго аргумента константы `MYSQL_ASSOC` и `MYSQL_NUM`, аналогичен функциям `mysql_fetch_assoc()` и `mysql_fetch_row()` соответственно.

При возврате ассоциативного массива в качестве индексов выступают имена столбцов результирующей таблицы, из которой производится выборка. Наиболее интересен режим `MYSQL_BOTH`. В листинге 28.13 извлекается первая запись таблицы `catalogs` и выводится дамп массива, возвращаемого функцией `mysql_fetch_array()`.

> ***ЗАМЕЧАНИЕ***
>
> Для вывода дампа массива применяется функция `print_r()`. Результат функции заключается в теги `<pre>` и `</pre>`, чтобы сохранить форматирование, т. к. браузер воспринимает символы перевода строки как пробельные символы.

Листинг 28.13. Использование функции `mysql_fetch_array()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $prd = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$prd) exit(mysql_error());
  // Возвращаем результат в виде массива
  $product = mysql_fetch_array($prd);
```

```
  echo "<pre>";
  print_r($product);
  echo "</pre>";
?>
```

Использовать функцию `mysql_fetch_array()` можно и для получения записей результирующей таблицы в цикле по аналогии с функциями `mysql_fetch_assoc()` и `mysql_fetch_row()` (листинг 28.14).

Листинг 28.14. Извлечение содержимого таблицы `catalogs`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit(mysql_error());
  // Формируем таблицу
  echo "<table border=1>";
  while($catalog = mysql_fetch_array($cat))
  {
    echo "<tr>
            <td>".$catalogs['id_catalog']."</td>
            <td>".$catalogs['name']."</td>
          </tr>";
  }
  echo "</table>";
?>
```

Функция `mysql_fetch_object()` возвращает текущую запись в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Повторный вызов функции переводит курсор в результирующей таблице на следующую запись. Функция `mysql_fetch_object()` имеет такой синтаксис:

```
mysql_fetch_object($result)
```

В качестве единственного аргумента `$result` функция принимает дескриптор запроса, возвращаемый функцией `mysql_query()`. Возвращает объект с членами, соответствующими столбцам в результирующей таблице, или `FALSE`, если рядов больше нет.

> ***ЗАМЕЧАНИЕ***
>
> Имена полей, возвращаемых этой функцией, не зависят от регистра.

В листинге 28.15 приведен пример, в котором с помощью этой функции из таблицы `catalogs` выводятся первичный ключ таблицы `id_catalog` и имя элемента каталога `name`.

Листинг 28.15. Использование функции `mysql_fetch_object()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT * FROM catalogs";
  // Выполняем SQL-запрос
  $cat = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$cat) exit(mysql_error());
  while($catalog = mysql_fetch_object($cat))
  {
    echo "id_catalog: ".$catalog->id_catalog."<br>";
    echo "name: ".$catalog->name."<br>";
  }
?>
```

Результат работы скрипта из листинга 28.15 представлен на рис. 28.3.

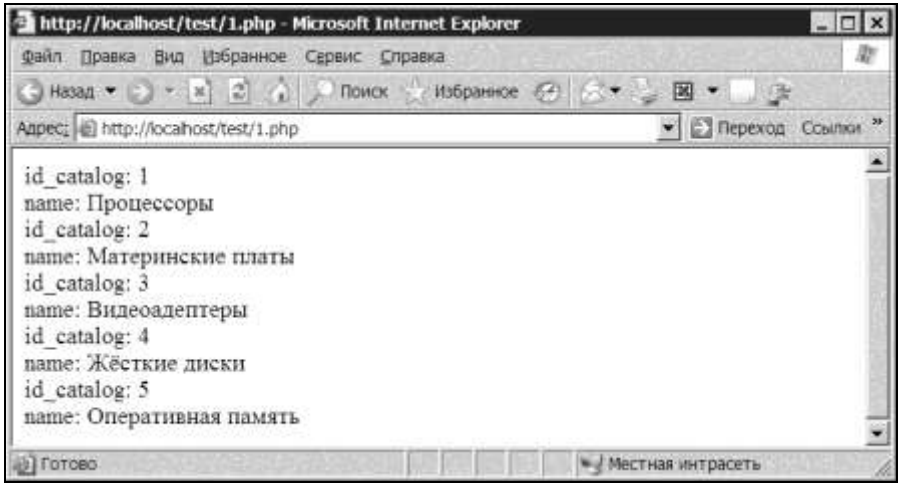


Рис. 28.3. Результат работы скрипта из листинга 28.15

28.1.5. Количество строк в таблице

Одной из распространенных задач при выборке из базы данных является определение числа записей, которые возвращает функция `mysql_query()`. Для этого предназначена функция `mysql_num_rows()`, которая имеет следующий синтаксис:

```
mysql_num_rows($result)
```

В качестве единственного аргумента `$result` функция принимает дескриптор результирующей таблицы, возвращаемый функцией `mysql_query()`.

> ***ЗАМЕЧАНИЕ***
>
> Эта функция работает только с запросами `SELECT`. Чтобы получить количество рядов, обработанных функциями `INSERT`, `UPDATE`, `DELETE`, следует использовать функцию `mysql_affected_rows()`.

> ***ЗАМЕЧАНИЕ***
>
> Функция `mysql_num_fields()` позволяет определить число столбцов в результате.

Создадим таблицу `test`, содержащую два поля: первичный ключ `id_test` и текстовое поле `name` (листинг 28.16).

Листинг 28.16. Таблица `test`

```
CREATE TABLE test (
  id_test int(11) NOT NULL auto_increment,
  name tinytext NOT NULL,
  PRIMARY KEY  (id_test)
);
```

В настоящий момент в таблице нет ни одной записи, поэтому попытка извлечь результирующую таблицу будет завершаться неудачей (листинг 28.17).

Листинг 28.17. Попытка извлечь несуществующую запись

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$tst) exit(mysql_error());
  // Выводим результат
  echo mysql_result($tst,0);
?>
```



Рис. 28.4. Попытка извлечь несуществующую запись

В результате работы скрипта из листинга 28.17 будет возникать ошибка (рис. 28.4).

Для предотвращения возникновения такой ошибки удобно воспользоваться функцией `mysql_num_rows()` (листинг 28.18).

Листинг 28.18. Использование функции `mysql_num_rows()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Формируем SQL-запрос
  $query = "SELECT name FROM test LIMIT 1";
  // Выполняем SQL-запрос
  $tst = mysql_query($query);
  // Проверяем успешность выполнения SQL-запроса
  if (!$tst) exit(mysql_error());
  // Выводим результат, если в результирующей
  // таблице имеется хотя бы одна запись
  if (mysql_num_rows($tst) > 0)
  {
    echo mysql_result($tst,0);
  }
?>
```

Число строк можно так же определить при помощи отдельного SQL-запроса, воспользовавшись встроенной функцией MySQL — `COUNT()`:

```
SELECT COUNT(*) FROM test;
```

Так, скрипт из листинга 28.18 можно переписать следующим образом (листинг 28.19).

Листинг 28.19. Использование встроенной функции `COUNT()`

```
<?php
  // Устанавливаем соединение с базой данных
  include "config.php";

  // Определяем количество записей в таблице test
  $query = "SELECT COUNT(*) FROM test";
  $cnt = mysql_query($query);
  if (!$cnt) exit(mysql_error());
  $total = mysql_result($cnt,0);
  if ($total > 0)
  {
    // Формируем SQL-запрос
    $query = "SELECT name FROM test LIMIT 1";
    // Выполняем SQL-запрос
    $tst = mysql_query($query);
    // Проверяем успешность выполнения SQL-запроса
    if (!$tst) exit(mysql_error());
    // Выводим результат
    echo mysql_result($tst,0);
  }
?>
```

Использование функции `COUNT()` для определения числа записей на первый взгляд может показаться нерациональным, т. к. вместо одного запроса (см. листинг 28.18) используются два (см. листинг 28.19). Однако такой подход является единственно верным, когда результирующая таблица содержит только часть выборки, например, за счет ограничения конструкцией `LIMIT`, а для дальнейших вычислений требуется точное количество записей в таблице.

28.1.6. Экранирование данных. SQL-инъекции

При работе с СУБД часто возникает задача экранирования данных, которые поступают извне. Пусть имеется таблица `userslist` с данными пользователей (листинг 28.20).

Листинг 28.20. Таблица `userslist`

```
CREATE TABLE userslist (
  id_user INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  pass TINYTEXT NOT NULL,
  email TINYTEXT NOT NULL,
```

```
  url TINYTEXT NOT NULL,
  PRIMARY KEY (id_user)
);
```

Каждая запись таблицы `userslist` состоит из пяти полей:

- `id_user` — первичный ключ таблицы, обладающий атрибутом `AUTO_INCREMENT`;
- `name` — имя пользователя;
- `pass` — его пароль;
- `email` — адрес электронной почты пользователя;
- `url` — адрес домашней страницы пользователя.

Список пользователей можно вывести при помощи скрипта из листинга 28.21. Имя каждого пользователя подсвечивается:

```
<a href='user.php?id_user=1'>name</a>
```

Здесь `1` — первичный ключ записи в таблице `userslist`, а `name` — имя пользователя.

Листинг 28.21. Список пользователей

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist ORDER BY name";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка - ".mysql_error());
  while($user = mysql_fetch_array($usr))
  {
    echo "<a href='user.php?id_user=$user[id_user]'>$user[name]</a><br>";
  }
?>
```

На странице user.php происходит предоставление информации о каждом из посетителей. Код скрипта из файла user.php представлен в листинге 28.22.

Листинг 28.22. Содержимое файла user.php

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");
  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
```

```
  if (!$usr) exit("Ошибка - ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя - $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail - $user[email]<br>";
  if (!empty($user['url'])) echo "URL - $user[url]<br>";
?>
```

Как видно из листинга 28.22, GET-параметр `id_user` вставляется в SQL-запрос без всякой дополнительной проверки. С одной стороны, это может приводить к ошибке запроса, если вместо числа параметр будет содержать строковые данные, с другой стороны, это может приводить к уязвимости типа SQL-инъекции, когда злоумышленник получает возможность выполнять на сервере собственный SQL-код.

Вместо числа в SQL-запрос

```
SELECT * FROM userslist WHERE id_user = 1
```

может быть внедрена произвольная строка. Так, если в строку запроса подставить значение `'какой-то%20текст'`, то скрипт выдаст сообщение об ошибке (рис. 28.5), т. к. будет осуществлена попытка выполнения запроса:

```
SELECT * FROM userslist WHERE id_user = 'какой-то текст'
```

> ***ЗАМЕЧАНИЕ***
>
> Символ `%20` обозначает пробел. Запоминать коды недопустимых символов необязательно, корректную для URL строку всегда можно получить, пропустив текст через функцию `urlencode()`.

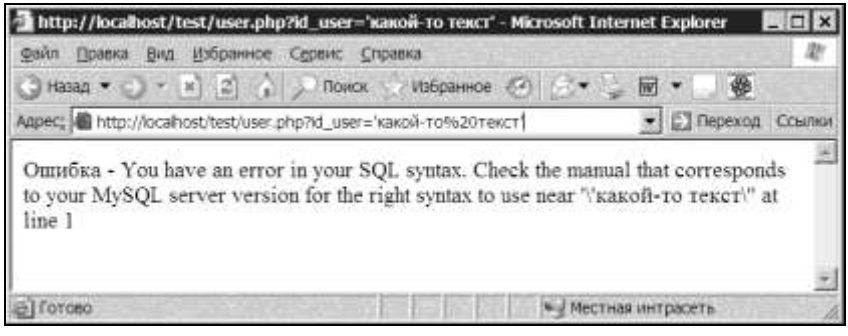


Рис. 28.5. Передача через строку запроса произвольного текста

Воспользовавшись конструкцией `UNION`, доступной, начиная с MySQL 4.0, можно добавить еще один запрос того же формата, что и первый (листинг 28.23).

Листинг 28.23. Использование конструкции `UNION`

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT * FROM userslist WHERE id_user = 2
```

Код SQL-инъекции выделен жирным шрифтом. Результирующая таблица будет содержать записи как из первого, так и из второго запроса. URL-запрос в этом случае может выглядеть следующим образом:

```
user.php?id_user=1%20UNION%20SELECT%20*%20FROM%20userslist%20WHERE%20id_user%2
0=%202
```

Тем не менее, на странице по-прежнему выводится информация о посетителе с первичным ключом 1 (рис. 28.6).

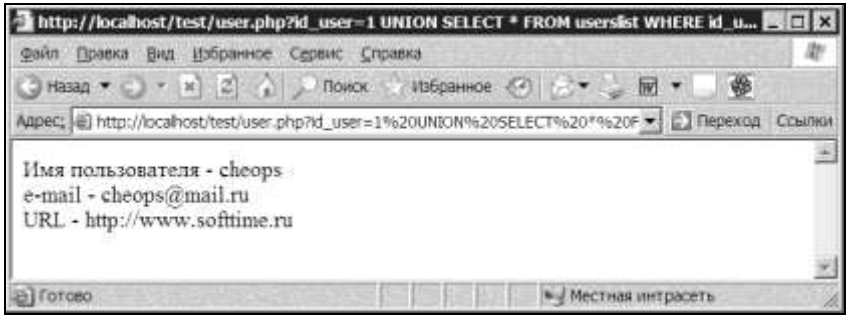


Рис. 28.6. Вывод информации о первом пользователе с первичным ключом 1

Запрос из листинга 28.24 возвращает две записи, которые соответствуют пользователю с первичным ключом 1 и пользователю с первичным ключом 2 (листинг 28.24).

Листинг 28.24. Записи, возвращаемые запросом из листинга 28.23

```
1, 'cheops', ******, 'cheops@mail.ru', 'http://www.softtime.ru'
2, 'barton', ******, 'barton@mail.ru', ''
```

Однако результаты выводятся только для самого первого запроса, т. к. в коде (см. листинг 28.21) присутствует только один вызов функции `mysql_fetch_array()` и, следовательно, все последующие записи игнорируются. Первой задачей злоумышленника является вывод в окно браузера результатов второго запроса из конструкции `UNION`, т. к. первый запрос изменить уже нельзя. Для реализации этой задачи можно воспользоваться двумя способами. Первый из них заключается в формировании такого условия первого запроса, который заведомо невыполним (листинг 28.25). Для этого переменной `id_user` можно присвоить отрицательное значение. Поскольку первичный ключ принимает только положительные значения, можно гарантировать, что первый запрос не вернет ни одной записи.

Листинг 28.25. Подавление вывода первого запроса

```
SELECT * FROM userslist WHERE id_user = -1
UNION
SELECT * FROM userslist WHERE id_user = 2
```

URL-запрос для SQL-инъекции из листинга 28.25 может выглядеть так, как это представлено ниже:

```
user.php?id_user=-1%20UNION%20SELECT%20*%20FROM%20userslist%20WHERE%20
id_user%20=%202
```

В результате такой инъекции в действие вступает второй `SELECT`-запрос из конструкции `UNION`, благодаря которому выводится информация о пользователе с первичным ключом 2 (рис. 28.7).

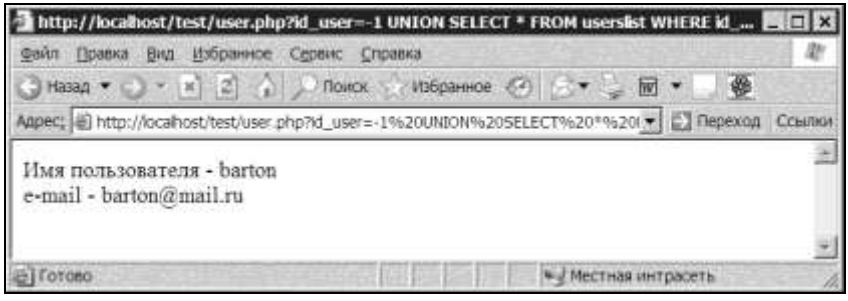


Рис. 28.7. Передача управления второму `SELECT`-запросу из конструкции `UNION`

Иногда передать управление второму `SELECT`-запросу описанным приемом сложно, поэтому прибегают к конструкции `ORDER BY`, которая сортирует результаты запросов так, чтобы результаты из инъекционного запроса оказались в начале. В нашем случае можно подвергнуть результаты обратной сортировке по параметру `id_user`. Для осуществления такой операции необходимо использовать конструкцию `ORDER BY id_user DESC`, которая размещается в конце конструкции `UNION` (листинг 28.26).

Листинг 28.26. Сортировка результатов

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT * FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Добившись вывода результатов инъекционного `SELECT`-запроса, можно приступать к выводу пароля в окно браузера. Для этого следует расшифровать символ ***** во втором запросе (листинг 28.27).

Листинг 28.27. Расшифровка символа * в инъекционном запросе

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT id_user, name, pass, email, url FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Их количество должно совпадать с количеством столбцов из первого `SELECT`-запроса, иначе СУБД отклонит запрос как ошибочный. Однако столбцы `name` и `pass` являются текстовыми, поэтому ничто не мешает нам поменять их местами (листинг 28.28).

Листинг 28.28. Перемена местами полей `name` и `pass`

```
SELECT * FROM userslist WHERE id_user = 1
UNION
SELECT id_user, pass, name, email, url FROM userslist WHERE id_user = 2
ORDER BY id_user DESC
```

Так как имена столбцов формируются по первому запросу, вместо имени пользователя (`name`) подставляется его пароль (`pass`) (рис. 28.8).

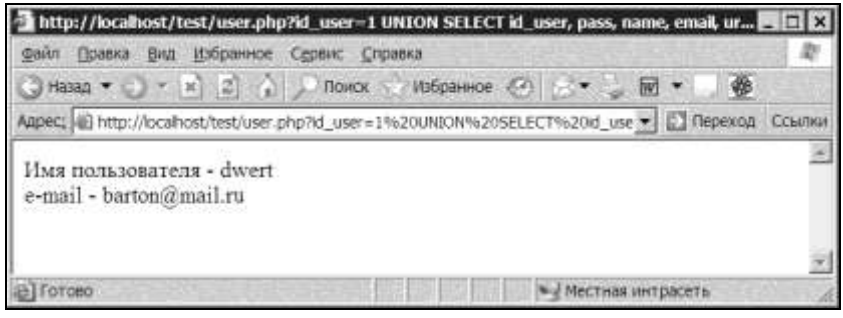


Рис. 28.8. Вывод пароля пользователя в окно браузера

ЗАМЕЧАНИЕ

Конструкция `UNION` применяется только совместно с `SELECT`-запросами, поэтому деструктивных действий такая SQL-инъекция не несет, если пароли не предоставляют доступ к панели управления, при помощи которой эти действия можно осуществить.

Для того чтобы избежать взлома по SQL-инъекции, следует проверять все числовые параметры при помощи регулярных выражений (листинг 28.29) или явно приводить их к числовой форме при помощи функции `intval()` (листинг 28.30).

Листинг 28.29. Предотвращаем SQL-инъекцию при помощи регулярных выражений

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем GET-параметр id_user, который должен содержать только числа
  if (!preg_match("|^[\d]+$|",$_GET['id_user']))
  {
```

```
    exit("Недопустимый формат URL-запроса");
  }

  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка - ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя - $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail - $user[email]<br>";
  if (!empty($user['url'])) echo "URL - $user[url]<br>";
?>
```

Листинг 28.30. Явное преобразование GET-параметра к числу

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем GET-параметр id_user, который должен содержать только числа
  $_GET['id_user'] = intval($_GET['id_user']);

  // Запрашиваем список всех пользователей
  $query = "SELECT * FROM userslist WHERE id_user = $_GET[id_user]";
  $usr = mysql_query($query);
  if (!$usr) exit("Ошибка - ".mysql_error());
  $user = mysql_fetch_array($usr);
  echo "Имя пользователя - $user[name]<br>";
  if (!empty($user['email'])) echo "e-mail - $user[email]<br>";
  if (!empty($user['url'])) echo "URL - $user[url]<br>";
?>
```

Если вместо числового параметра используется текстовый параметр, то необходимо экранировать кавычки, которые могут нарушить синтаксис или послужить причиной SQL-инъекции. Ситуация усложняется тем, что на сервере может быть включен режим "магических кавычек" *(см. разд. 20.1.8)*, и он может самостоятельно пытаться произвести экранирование. В листинге 28.31 приводится типичный обработчик строковых параметров, который применяется перед тем, как использовать внешние данные в SQL-запросах.

ЗАМЕЧАНИЕ

Для экранирования кавычек используется функция `mysql_escape_string()`.

Листинг 28.31. Обработка строковых параметров

```
<?php
  ...
  if (!get_magic_quotes_gpc())
  {
    // Режим "магических кавычек" отключен
    $_POST['name'] = mysql_escape_string($_POST['name']);
    $_POST['pass'] = mysql_escape_string($_POST['pass']);
  }
  ...
?>
```

28.2. Библиотека php_mysqli

Расширение php_mysqli является новым объектно-ориентированным интерфейсом для доступа к СУБД MySQL из PHP-кода и доступно при работе с СУБД MySQL версии выше 4.1. Интерфейс включает в себя классы для доступа к СУБД MySQL, отложенных запросов и класса результирующей таблицы. Библиотека не заменяет собой функции из расширения php_mysql — наряду с объектно-ориентированным расширением доступен процедурный стиль программирования.

> ***ЗАМЕЧАНИЕ***
>
> В отличие от старого расширения php_mysql, в новом расширении php_mysqli функции начинаются с префикса mysqli, а не с mysql.

Главным классом расширения является `mysqli`, при помощи которого осуществляется установка соединения с сервером и выполнения запросов. В листинге 28.32 приводится пример, в котором устанавливается соединение с сервером MySQL и выводятся сведения о его версии.

Листинг 28.32. Версия MySQL-сервера

```
<?php
  // Объявляем объект класса mysqli, который устанавливает
  // соединение с сервером
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Выводим версию сервера
  echo $mysqli->server_info;

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

Методы класса mysqli представлены в табл. 28.3.

Таблица 28.3. *Методы класса* `mysqli`

Метод	Описание
`autocommit(`*`mode`*`)`	Включает или отключает режим автозавершения транзакции. По умолчанию автозавершение включено, и каждый оператор рассматривается как отдельная транзакция. Отключение режима автозавершения требует явного завершения транзакции при помощи методов `commit()` и `rollback()`. Параметр *`mode`* принимает значение `TRUE` или `FALSE`
`change_user(`*`user`*`, `*`password`*`, `*`database`*`)`	Изменяет параметры соединения для текущего объекта `mysqli`, устанавливая нового пользователя *`user`*, его пароль *`password`* и текущую базу данных *`database`*
`character_set_name()`	Возвращает текущую кодировку соединения, которую извлекает из системной переменной MySQL — `character_set`
`close()`	Закрывает соединение с сервером MySQL
`commit()`	Подтверждает, что транзакция успешно завершена и все изменения записаны на жесткий диск
`get_client_info()`	Возвращает версию клиентской библиотеки
`kill(`*`processid`*`)`	Уничтожает процесс с идентификатором *`processid`*
`multi_query (`*`query`*`)`	Выполняет SQL-запросы в строке *`query`*, в отличие от метода `query()` в строке *`query`* допускается несколько SQL-запросов, разделенных точкой с запятой
`more_results()`	Проверяет, имеются ли записи в таблице, полученной в результате выполнения метода `multi_query()`, и возвращает `TRUE`, если записи имеются, и `FALSE` — в противном случае
`next_result()`	Получает следующую запись из таблицы, возвращенной в результате выполнения метода `multi_query()`
`options(`*`option`*`, `*`value`*`)`	Устанавливает для параметра *`option`* значение *`value`*
`ping()`	Проверяет соединение с сервером, и, если оно отсутствует, выполняет повторную попытку соединения
`prepare(`*`query`*`)`	Создает отложенный запрос *`query`*, возвращая дескриптор отложенного запроса. При помощи метода `execute()` отложенного запроса его можно выполнить в любой удобный момент
`query(`*`query`*`)`	Выполняет SQL-запрос в строке *`query`*, в отличие от метода `multi_query()` возможно выполнение только одного запроса
`real_connect([`*`hostname`*`[, `*`username`*`[, `*`passwd`*`[, `*`dbname`*`[, `*`port`*`[, `*`socket`*`[, `*`flags`*`]]]]]] )`	Устанавливает соединение с сервером, если оно не было установлено заранее конструктором класса `mysqli`, а объект был получен при помощи функции `mysqli_init()`

Таблица 28.3 *(окончание)*

Метод	Описание
`mysqli_real_escape_string(`*`escapestr`*`)`	Экранирует специальные символы для SQL-запроса, который передается методу `query()` или `multi_query()`
`rollback()`	Выполняет откат транзакции, возвращая СУБД в состояние, которое было до начала транзакции
`select_db(`*`dbname`*`)`	Позволяет выбрать текущую базу данных *`dbname`*
`set_charset(`*`charset`*`)`	Устанавливает текущую кодировку соединения *`charset`*
`ssl_set(`*`key`*`, `*`cert`*`, `*`ca`*`, `*`capath`*`, `*`cipher`*`)`	Устанавливает безопасное SSL-соединение с MySQL-сервером
`stat()`	Возвращает строку, содержащую информацию о состоянии MySQL-сервера
`stmt_init()`	Возвращает объект отложенного запроса; является альтернативой методу `prepare()`
`store_result()`	Возвращает дескриптор результирующей таблицы последнего запроса
`use_result()`	Инициирует результирующую таблицу последнего запроса
`thread_safe()`	Возвращает `TRUE`, если клиентская библиотека скомпилирована в безопасном режиме, и `FALSE` — в противном случае

Помимо методов, класс `mysqli` имеет ряд открытых членов, список которых представлен в табл. 28.4.

Таблица 28.4. *Члены класса* `mysqli`

Член	Описание
`affected_rows`	Возвращает количество записей, обработанных операторами `DELETE`, `UPDATE` и `REPLACE`
`client_info`	Возвращает версию клиентской библиотеки в виде строки
`client_version`	Возвращает версию клиентской библиотеки в виде числа
`errno`	Возвращает номер ошибки для последней операции
`error`	Возвращает текстовое сообщение об ошибке для последней операции
`field_count`	Возвращает количество полей результирующей таблицы из последнего запроса
`host_info`	Возвращает адрес MySQL-сервера и тип соединения

Таблица 28.4 *(окончание)*

Член	Описание
info	Возвращает информацию о последнем запросе, количество затронутых столбцов, предупреждений и т. п.
insert_id	Возвращает первичный ключ, сгенерированный по механизму AUTO_INCREMENT при последнем INSERT-запросе
protocol_version	Возвращает версию протокола взаимодействия клиента и MySQL-сервера
server_info	Возвращает версию MySQL-сервера в виде строки
server_version	Возвращает версию MySQL-сервера в виде числа
Sqlstate	Возвращает номер и текстовое сообщение о последней ошибке
thread_id	Возвращает уникальный идентификатор текущего потока
warning_count	Определяет количество предупреждений, которое возвратил MySQL-сервер для последнего SQL-запроса

Второй класс — mysqli_stmt — применяется для формирования и обработки отложенных запросов. В листинге 28.33 приводится типичный вариант использования отложенного запроса.

Листинг 28.33. Использование класса mysqli_stmt

```
<?php
  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli("localhost", "root", "", "test");
  if (mysqli_connect_errno())  exit("Ошибка установки соединения");

  // Формируем подготовленный запрос
  $query = "SELECT name, family FROM users ORDER BY name";
  if ($stmt = $mysqli->prepare($query))
  {
    // Выполняем запрос
    $stmt->execute();

    // Связываем переменные $name и $family со столбцами
    // name и family таблицы users
    $stmt->bind_result('ss', $name, $family);

    // Сохраняем результирующую таблицу
    $stmt->store_result();

    // Перемещаемся на вторую строку (нумерация начинается с 0)
```

```
    $stmt->data_seek(1);

    // Помещаем данные в переменные $name и $family
    $stmt->fetch();

    // Выводим результат
    echo "Имя : $name<br>";
    echo "Фамилия : $family<br>";

    // Освобождаем дескриптор подготовленного запроса
    $stmt->close();
  }

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

В табл. 28.5 приводится список методов класса `mysqli_stmt`.

Таблица 28.5. *Методы класса* `mysqli_stmt`

Метод	Описание
`bind_param(`*`types`*`, &`*`var1`* `[, &...])`	Связывает переменные, предаваемые в качестве аргумента, с переменными в SQL-запросе, которые обозначаются символом `?`. Аргумент *`types`* задает тип данных и может принимать значения `'i'`, `'d'`, `'s'` и `'b'` для целочисленных, вещественных, строковых и BLOB-полей
`bind_result(&`*`var1`* `[, &...] )`	Связывает переменные, передаваемые в качестве аргумента, со столбцами результирующей таблицы
`close()`	Освобождает дескриптор отложенного запроса
`data_seek(`*`offset`*`)`	Перемещается на строку *`offset`* в результирующей таблице (нумерация строк начинается с 0)
`execute()`	Выполняет отложенный запрос
`fetch()`	Помещает данные из текущей строки результирующей таблицы в связанные переменные, назначенные ранее при помощи метода `bind_result()`
`free_result()`	Освобождает память, выделенную под результирующую таблицу; после вызова метода данные результирующей таблицы становятся недоступны
`result_metadata()`	Возвращает метаданные отложенного запроса
`prepare (`*`query`*`)`	Создает отложенный запрос

***Таблица 28.5** (окончание)*

Метод	Описание
`send_long_data(param_nr, data)`	Параметру `param_nr` присваивается значение `data`. Функция используется для передачи объемных BLOB-значений, которые превышают `max_allowed_packet`. Параметр, назначенный данной функцией, будет передан на сервер в несколько приемов
`reset()`	Сбрасывает отложенный запрос
`store_result()`	Сохраняет результирующую таблицу

Помимо методов, класс `mysqli_stmt` имеет ряд открытых членов, список которых представлен в табл. 28.6.

***Таблица 28.6.** Члены класса `mysqli_stmt`*

Член	Описание
`affected_rows`	Возвращает количество записей, обработанных операторами `DELETE`, `UPDATE` и `REPLACE`
`errno`	Возвращает номер ошибки для последней операции
`error`	Возвращает текстовое сообщение об ошибке для последней операции
`field_count`	Возвращает количество полей результирующей таблицы из последнего запроса
`id`	Возвращает идентификатор запроса
`insert_id`	Возвращает первичный ключ, сгенерированный по механизму `AUTO_INCREMENT` при последнем `INSERT`-запросе
`num_rows`	Возвращает количество записей в результирующей таблице
`param_count`	Возвращает количество параметров в отложенном запросе
`sqlstate`	Возвращает номер и текстовое сообщение о последней ошибке

Последний класс `mysqli_result` предназначен для работы с результирующей таблицей. Методы класса представлены в табл. 28.7.

***Таблица 28.7.** Методы класса `mysqli_result`*

Метод	Описание
`close()`	Освобождает память, выделенную под результирующую таблицу
`data_seek(offset)`	Отыскивает в результирующей таблице строку с номером `offset` (нумерация строк начинается с 0)
`fetch_field()`	Возвращает информацию о столбцах результирующей таблицы в виде объекта

***Таблица 28.7** (окончание)*

Метод	Описание
`fetch_fields()`	Возвращает массив объектов с информацией о столбцах результирующей таблицы
`fetch_field_direct (fieldnr)`	Возвращает объект с информацией о столбце с номером *fieldnr*. Нумерация столбцов в результирующей таблице начинается с 0
`fetch_array(resulttype)`	Возвращает текущую запись результирующей таблицы в виде массива, элементы которого соответствуют полям записи. Параметр *resulttype* определяет тип возвращаемого массива: `MYSQLI_NUM` — числовой массив, `MYSQLI_ASSOC` — ассоциативный массив, `MYSQLI_BOTH` — возвращает и числовой, и ассоциативный массив
`fetch_assoc()`	Возвращает текущую запись результирующей таблицы в виде ассоциативного массива. Элементы массива соответствуют полям записи
`fetch_object()`	Возвращает текущую запись результирующей таблицы в виде объекта
`fetch_row()`	Возвращает текущую запись в виде обычного массива
`free_result()`	Освобождает память, выделенную под результирующую таблицу
`field_seek(fieldnr)`	Устанавливает курсор результирующей таблицы на запись номер *fieldnr* (нумерация записей начинается с 0)

Помимо методов, класс `mysqli_result` имеет ряд открытых членов, список которых представлен в табл. 28.8.

***Таблица 28.8.** Члены класса `mysqli_result`*

Член	Описание
`current_field`	Возвращает номер текущего столбца (нумерация столбцов начинается с 0) при извлечении информации методом `fetch_field()`
`field_count`	Возвращает количество столбцов в результирующей таблице
`lengths`	Возвращает длину текущего столбца при извлечении информации методом `fetch_field()`
`num_rows`	Возвращает количество записей в результирующей таблице
`type`	Возвращает одну из констант: `MYSQLI_STORE_RESULT` или `MYSQLI_USE_RESULT`, в зависимости от того, сохранен результат или нет

В листинге 28.34 представлен файл config.php, устанавливающий соединение с MySQL-сервером. Такой файл включается при помощи конструкции `require_once` во все скрипты, где требуется наличие класса объекта класса `mysqli`.

Листинг 28.34. Установка соединения с СУБД MySQL config.php

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с сервером MySQL
  $mysqli = new mysqli($dblocation, $dbuser, $dbpasswd, $dbname);
  if (mysqli_connect_errno()) exit("Ошибка установки соединения");

  // Устанавливаем кодировку соединения. Следует выбрать ту кодировку,
  // в которой данные будут отправляться MySQL-серверу
  $mysqli->query("SET NAMES 'cp1251'");
?>
```

Объявление объекта класса `mysqli` приводит к установке соединения с сервером, расположенным по адресу `$dblocation`, и выбору базы данных `$dbname` от имени пользователя `$dbuser`, имеющего пароль `$dbpasswd`.

MySQL-серверу отправляется запрос `"SET NAMES 'cp1251'"`. Данный запрос сообщает, что все данные будут передаваться на сервер в кодировке Windows-1251, и MySQL-сервер, в свою очередь, в какой бы кодировке не хранились данные в таблицах, должен предоставлять результат в кодировке Windows-1251.

В следующих разделах будут рассмотрены наиболее типичные задачи, которые возникают при взаимодействии PHP-скриптов и СУБД MySQL.

28.2.1. Создание базы данных

Создать новую базу данных можно путем выполнения оператора `CREATE DATABASE`. В листинге 28.35 представлен скрипт, создающий базу данных `dbase`.

Листинг 28.35. Создание базы данных

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");
```

```
  // Формируем и выполняем запрос
  $query = "CREATE DATABASE dbase";
  if (!$mysqli->query($query))
  {
    exit("База данных не создана ".$mysqli->error);
  }
?>
```

Метод `query()` объекта класса `mysqli` в случае успешного завершения возвращает либо `TRUE`, если запрос не возвращает результирующей таблицы, либо объект результирующей таблицы, который в контексте оператора `if` также воспринимается как истинное значение (`TRUE`). Если запрос заканчивается неудачей, возвращается `FALSE`.

Следует обязательно проверить, не возвращает ли СУБД MySQL ошибку выполнения последнего запроса, поскольку интерпретатор PHP никак не сигнализирует об ошибках синтаксиса MySQL: PHP-скрипт и SQL-запросы выполняются в разных потоках, и PHP-интерпретатор не имеет сведений об успешности или неудаче выполнения SQL-запроса.

28.2.2. Создание и заполнение таблицы

Процесс создания и заполнения таблицы схож с рассмотренным в предыдущем разделе процессом создания базы данных. Отличие заключается лишь в том, что потребуется выполнить несколько SQL-запросов.

Создадим таблицу `catalogs`, состоящую из трех полей:

- `id_catalog` — первичный ключ таблицы, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название элемента каталога;
- `putdate` — дата последней модификации каталога.

Для выполнения нескольких запросов удобно воспользоваться методом `multi_query()`, который позволяет за один раз выполнить сразу несколько запросов, отделенных друг от друга точкой с запятой (листинг 28.36).

Листинг 28.36. Создание таблицы `catalogs`

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем запросы
  $query = "CREATE TABLE catalogs (
            id_catalog int(11) NOT NULL AUTO_INCREMENT,
            name tinytext NOT NULL,
            putdate datetime NOT NULL,
            PRIMARY KEY  (id_catalog)
            ) ENGINE=MyISAM;
```

```
            INSERT INTO catalogs
            VALUES (NULL, 'Процессоры', '2007-01-10 12:47:00');
            INSERT INTO catalogs
            VALUES (NULL, 'Материнские платы', '2006-12-28 18:32:41');
            INSERT INTO catalogs
            VALUES (NULL, 'Видеоадаптеры', '2007-01-10 16:48:05');
            INSERT INTO catalogs
            VALUES (NULL, 'Жесткие диски', '2007-01-05 20:01:58');
            INSERT INTO catalogs
            VALUES (NULL, 'Оперативная память', '2006-12-20 10:00:13')";
  if (!$mysqli->multi_query($query))
  {
    exit("Таблицу не удалось развернуть ".$mysqli->error);
  }
?>
```

Замечание

В процедурной библиотеке php_mysql отсутствует аналог, позволяющий выполнить одновременно несколько SQL-запросов — обычно SQL-дамп разбивается на отдельные запросы, которые выполняются в цикле.

28.2.3. Заполнение связанных таблиц

Еще один интересный случай представляет собой заполнение связанных таблиц. Пусть имеется таблица `news`, предназначенная для хранения новостных сообщений и состоящая из трех полей:

- `id_news` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `name` — название новостной позиции;
- `putdate` — дата размещения новости.

С таблицей `news` связана таблица `news_contents`, предназначенная для хранения текста новости и также состоящая из трех полей:

- `id_content` — первичный ключ, снабженный атрибутом `AUTO_INCREMENT`;
- `content` — содержимое новостного сообщения;
- `id_news` — внешний ключ, содержащий значения полей `id_news` из таблицы `news`, связывая новостной блок и текст новости.

В листинге 28.37 представлены операторы `CREATE TABLE`, которые создают таблицы `news` и `news_contents`.

Листинг 28.37. Таблицы `news` и `news_contents`

```
CREATE TABLE news (
  id_news INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
```

```
  putdate DATETIME NOT NULL,
  PRIMARY KEY (id_news)
);
CREATE TABLE news_contents (
  id_content INT(11) NOT NULL AUTO_INCREMENT,
  content TEXT NOT NULL,
  id_news INT(11) NOT NULL,
  PRIMARY KEY (id_content)
);
```

На рис. 28.9 представлена HTML-форма, которая позволяет добавить название и текст новостного сообщения в базу данных.



Рис. 28.9. HTML-форма для добавления новостного сообщения в базу данных

HTML-код, воссоздающий форму, представленную на рис. 28.9, приведен в листинге 28.38.

Листинг 28.38. HTML-форма для добавления новостного сообщения в базу данных

```
<table>
<form action='addnews.php' method='POST'>
  <tr>
    <td>Название:</td>
    <td><input type='text' name='name'></td>
  </tr>
  <tr>
```

```
    <td>Содержимое:</td>
    <td><textarea name='content' rows='10' cols='40'></textarea></td>
  </tr>
  <tr>
    <td></td>
    <td><input type='submit' value='Добавить'></td>
  </tr>
</form>
</table>
```

Как видно из листинга 28.38, обработчиком HTML-формы назначен файл addnews.php. В листинге 28.39 представлен обработчик HTML-формы, который осуществляет вставку новостного сообщения в таблицы `news` и `news_contents`.

Листинг 28.39. Добавление новостного сообщения в базу данных

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Проверяем, заполнены ли поля HTML-формы
  if (empty($_POST['name'])) exit('Не заполнено поле "Название"');
  if (empty($_POST['content'])) exit('Не заполнено поле "Содержимое"');

  // Экранируем специальные символы
  if (!get_magic_quotes_gpc())
  {
    $_POST['name'] = $mysqli->real_escape_string($_POST['name']);
    $_POST['content'] = $mysqli->real_escape_string($_POST['content']);
  }

  // Добавляем новостное сообщение в таблицу news
  $query = "INSERT INTO news VALUES (NULL, '$_POST[name]', NOW())";
  if (!$mysqli->query($query))
  {
    exit("Ошибка добавления новостного сообщения".$mysqli->error);
  }

  // Получаем только что сгенерированный идентификатор id_news
  $id_news = $mysqli->insert_id;

  // Вставляем содержимое новостного сообщения в таблицу news_contents.
```

```
  // Формируем запросы
  $query = "INSERT INTO news_contents
            VALUES (NULL, '$_POST[content]', $id_news)";
  if (!$mysqli->query($query))
  {
    exit("Ошибка добавления новостного сообщения ".$mysqli->error);
  }

  // Осуществляем переадресацию на главную страницу
  header("Location: form.php");
?>
```

Так как данные передаются методом POST, то содержимое полей `name` и `content` попадает в элементы суперглобального массива `$_POST['name']` и `$_POST['name']` соответственно. В начале обработчика производится проверка правильности заполнения полей формы — если хоть одно из полей остается незаполненным, процесс добавления информации приостанавливается с выдачей соответствующего предупреждения. Наличие символов в строке можно проверить при помощи функции `empty()`, которая возвращает `TRUE`, если строка пустая, и `FALSE`, если строка содержит хотя бы один символ.

После этого при помощи метода `real_escape_string()` экранируются специальные символы в строковых переменных. Если строка содержит символ `'`, то синтаксис SQL-оператора может быть нарушен. Такой символ следует заменить на последовательность `\'`. Именно для этого и предназначен метод `real_escape_string()` класса `mysqli`.

> ***ЗАМЕЧАНИЕ***
>
> На ряде серверов включен режим "магических кавычек". В этом режиме экранирование специальных символов в `POST`-, `GET`- и `COOKIE`-данных осуществляется автоматически, и повторно экранировать данные не требуется. Выяснить, включен режим "магических кавычек" или нет, можно при помощи функции `get_magic_quotes_gpc()`, которая возвращает `TRUE`, если режим включен, и `FALSE` — в противном случае.

Основной особенностью добавления данных в связанные таблицы является то, что для добавления записи во вторую таблицу `news_content` необходимо знать первичный ключ `id_news`, который был сгенерирован по механизму `AUTO_INCREMENT` для только что созданной записи в таблице `news`. Получить его можно, обратившись к члену `insert_id` класса `mysqli`.

После того как записи добавлены, следует вернуться на главную страницу HTML-формы или страницу, где выводятся новостные позиции. Для этого браузеру оправляется HTTP-заголовок `Location` с адресом страницы, на которую следует осуществить переход.

28.2.4. Вывод данных

Еще одной часто встречающейся задачей является вывод содержимого таблицы из базы данных. Пусть имеется таблица `catalogs`, содержимое которой представлено в листинге 28.40.

Листинг 28.40. Таблица `catalogs`

```
CREATE TABLE catalogs (
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,
  name TINYTEXT NOT NULL,
  PRIMARY KEY (id_catalog)
);
INSERT INTO catalogs VALUES (NULL, 'Процессоры'),
                            (NULL, 'Материнские платы'),
                            (NULL, 'Видеоадаптеры'),
                            (NULL, 'Жесткие диски'),
                            (NULL, 'Оперативная память');
```

В листинге 28.41 представлен скрипт, выводящий содержимое таблицы в окно браузера.

Листинг 28.41. Вывод содержимого таблицы `catalogs`

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";
  $cat = $mysqli->query($query);
  if (!$cat) exit("Ошибка обращения к каталогу");

  // Если имеется хотя бы одна запись, выводим список
  if ($cat->num_rows)
  {
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга 28.41, метод `query()` объекта `mysqli` возвращает объект результирующей таблицы `$cat`. Член `num_rows` объекта `$cat` возвращает количество записей в результирующей таблице. Если это значение больше нуля, содержимое таблицы извлекается при помощи метода `fetch_array()`. За один подход функция извлекает из результирующей таблицы одну запись, которая представляется в виде ассоциативного массива `$catalog`. В качестве ключей массива выступают имена столбцов

таблицы catalogs, а в качестве значений — элементы результирующей таблицы. Повторный вызов метода fetch_array() приводит к извлечению следующего столбца и т. д. Поэтому вызов функции в цикле while() приводит к последовательному извлечению всех строк результирующей таблицы.

28.2.5. Повторное чтение результирующей таблицы

Следует отметить, что вызов метода fetch_array() объекта класса mysqli_stmt в цикле while приводит к тому, что курсор останавливается на последней записи результирующей таблицы и повторная попытка извлечения записей заканчивается неудачей. Для повторного чтения результирующей таблицы без отправки серверу MySQL SELECT-запроса необходимо вернуть курсор в начало результирующей таблицы. Для этого предназначен метод data_seek(), который имеет следующий синтаксис:

```
mysql_data_seek($row_number)
```

Метод устанавливает внутренний курсор на строку *$row_number* (нумерация строк начинается с 0). В случае успеха метод возвращает TRUE, в случае неудачи — FALSE.

Вернемся к задаче вывода содержимого таблицы catalogs, которая была рассмотрена в предыдущем разделе. Пусть требуется подсчитать количество символов в выводимых строках, после чего вывести содержимое результирующей таблицы (листинг 28.42).

Листинг 28.42. Повторное считывание результирующей таблицы

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT * FROM catalogs";

  $cat = $mysqli->query($query);
  if (!$cat) exit("Ошибка обращения к каталогу");
  // Если имеется хотя бы одна запись, выводим список
  if ($cat->num_rows)
  {
    // Подсчитываем количество символов во всех строках таблицы
    $count = 0;
    while($catalog = $cat->fetch_array())
    {
      $count += strlen($catalog['name']);
    }
    echo "Общая длина всех строк ".$count."<br>";

    // Устанавливаем внутренний курсор в начало результирующей таблицы
```

```
    $cat->data_seek(0);

    // Повторно считываем результирующую таблицу
    while($catalog = $cat->fetch_array())
    {
      echo $catalog['name']."<br>";
    }
  }
?>
```

Как видно из листинга 28.42, для того чтобы осуществить повторное считывание результирующей таблицы, необходимо установить курсор на нулевую запись при помощи метода `data_seek()`. Результатом работы скрипта из листинга 28.42 будут следующие строки:

```
Общая длина всех строк 71
Процессоры
Материнские платы
Видеоадаптеры
Жесткие диски
Оперативная память
```

28.2.6. Количество строк в таблице

В предыдущих разделах для определения количества строк в результирующей таблице использовалось свойство `num_rows` объекта результирующей таблицы. Если `SELECT`-выборка не имеет ограничений и таблица выбирается полностью, можно использовать данные способы для оценки количества строк в таблице. Однако зачастую на одну страницу выводится лишь часть таблицы, в то время как для организации навигации требуется знать общее количество строк во всей таблице. Если таблица объемная, полная выборка таблицы не приемлема ни по скорости, ни по объему памяти, которая потребуется для ее хранения. В этом случае разумнее прибегнуть к MySQL-функции `COUNT(*)`, которая подсчитывает количество строк в таблице. В листинге 28.43 представлен скрипт, подсчитывающий количество строк в таблице `catalogs` (см. листинг 28.40).

Листинг 28.43. Подсчет количества строк в таблице

```
<?php
  // Установка соединения с базой данных
  require_once("config.php");

  // Формируем SELECT-запрос
  $query = "SELECT COUNT(*) AS total FROM catalogs";

  $cat = $mysqli->query($query);
```

```
  if (!$cat) exit("Ошибка обращения к каталогу");

  $result = $cat->fetch_object();
  echo $result->total;
?>
```

В листинге 28.43 при помощи метода `fetch_object()` строка результирующей таблицы возвращается в виде объекта, имена членов которого совпадают с именами полей в результирующей таблице. Для удобства единственный столбец в результирующей таблице переименовывается в `total` при помощи оператора `AS`.

28.2.7. Удаление данных

При удалении данных часто прибегают к флажкам — можно отметить сразу несколько позиций, которые подвергаются удалению (рис. 28.10).

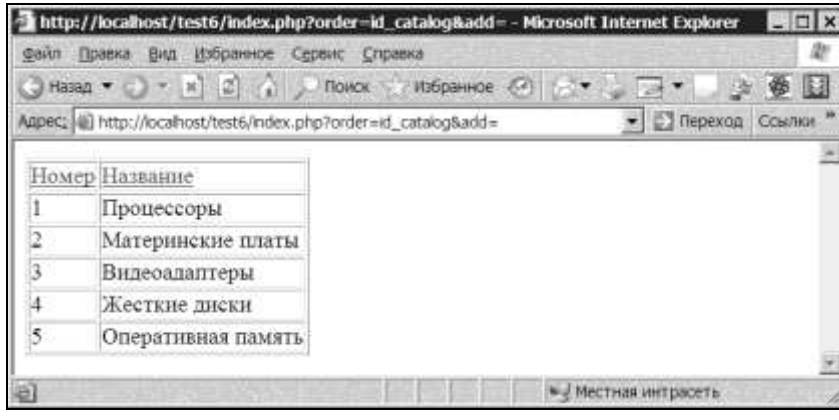


Рис. 28.10. Web-интерфейс для удаления нескольких записей таблицы

В листинге 28.44 представлен скрипт, позволяющий удалять сразу несколько записей, которые пользователь отмечает флажками.

Листинг 28.44. Удаление нескольких записей таблицы

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Если суперглобальный массив $_POST не пуст,
  // производим обработку запроса
  if (!empty($_POST))
  {
```

```
  // Для удаления позиций необходимо сформировать запрос вида
  // DELETE FROM catalogs WHERE id_catalog IN (4, 6, 8, ..., 20),
  // где цифры в скобках являются элементами
  // массива $_POST['catalog'][]
  $temp = array();
  foreach($_POST['catalog'] as $id_catalog)
  {
    // Проверяем, является ли переменная $id_catalog числом
    if (preg_match("|^[\d]+$|",$id_catalog))
    {
      $temp[] = $id_catalog;
    }
  }
  // Формируем и выполняем запрос на удаление нескольких элементов
  $query = "DELETE FROM catalogs
            WHERE id_catalog IN (".implode(",",$temp).")";
  if ($mysqli->query($query))
  {
    echo "<HTML><HEAD>
        <META HTTP-EQUIV='Refresh' CONTENT='0; URL=$_SERVER[PHP_SELF]'>
          </HEAD></HTML>";
    exit();
  }
}

// Выводим список элементов каталога
$query = "SELECT * FROM catalogs
          ORDER BY id_catalog";
$cat = $mysqli->query($query);
if (!$cat) exit($mysqli->error);
// Если имеется хотя бы одна позиция - выводим ее
if ($cat->num_rows)
{
  echo "<form method='post'>";
  echo "<table border='1'>
          <tr>
            <td> </td>
            <td>Номер</td>
            <td>Название</td>
          </tr>";
  $i = 0;
  while($catalog = $cat->fetch_array())
```

```
    {
      echo "<tr>
              <td><input type='checkbox' name='catalog[]'
                         value='$catalog[id_catalog]'></td>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
      $i++;
    }
    echo "</table>";
    echo "</br><input type='submit' name='send' value='Удалить'>";
    echo "</form>";
  }
?>
```

28.2.8. Сортировка

При выводе содержимого таблицы для удобства восприятия информации часто требуется организовывать сортировку по столбцам. Выведем содержимое таблицы `catalogs` (см. листинг 28.40) в таблицу, заголовки которой будут представлять собой гиперссылки, переход по которым осуществлял бы сортировку выбранного столбца (рис. 28.11). Повторный переход по гиперссылке должен приводить к сортировке данных в обратном порядке.

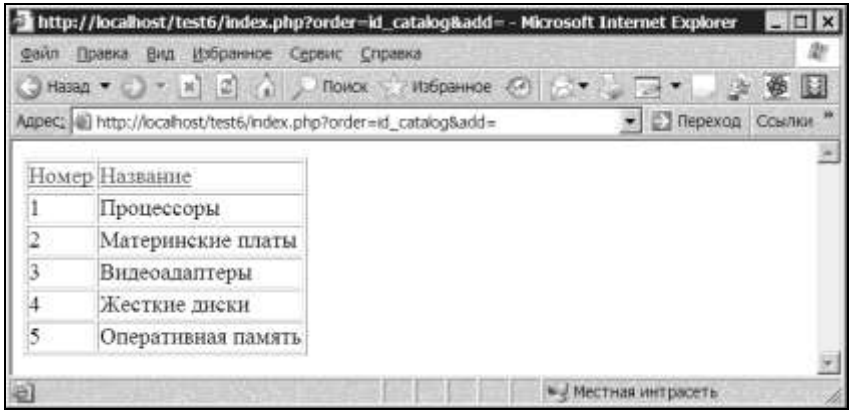


Рис. 28.11. Таблица с возможностью сортировки столбцов

Сортировка результатов SQL-запроса производится при помощи конструкции `ORDER BY`, после которой указывается имя столбца, подвергающегося сортировке. Чтобы сортировка выполнялась в обратном порядке, следует указать необязательное ключевое слово `DESC`. Таким образом, задача сводится к формированию динамического

SQL-запроса, в конструкцию ORDER BY которого подставлялись бы имена выбранных столбцов. Для этого будем использовать запрос вида:

```
SELECT * FROM catalogs
         ORDER BY $order $desc
```

Переменная $order будет содержать имя столбца, а $desc — принимать либо пустое значение, либо ключевое слово DESC в зависимости от того, какой вид сортировки выбран (листинг 28.45).

Листинг 28.45. Сортировка столбцов

```
<?php
  // Устанавливаем соединение с базой данных
  require_once("config.php");

  // Проверяем параметры, переданные скрипту
  $order = "name";
  if ($_GET['order'] == 'name')  $order = "name";
  if ($_GET['order'] == 'id_catalog') $order = "id_catalog";
  if ($_GET['add']   == 'desc')
  {
    $desc = "DESC";
    $add  = "";
  }
  else
  {
    $desc = "";
    $add  = "desc";
  }

  // Выводим элементы каталога
  $query = "SELECT * FROM catalogs
            ORDER BY $order $desc";
  $cat = $mysqli->query($query);
  if (!$cat) exit($mysqli->error);
  if ($cat->num_rows)
  {
    echo "<table border='1'>
            <tr>
  <td><a href=$_SERVER[PHP_SELF]?order=id_catalog&add=$add>Номер</a></td>
  <td><a href=$_SERVER[PHP_SELF]?order=name&add=$add>Название</a></td>
            </tr>";
    while($catalog = $cat->fetch_array())
```

```
    {
      echo "<tr>
              <td>$catalog[id_catalog]</td>
              <td>$catalog[name]</td>
            </tr>";
    }
    echo "</table>";
  }
?>
```

В начале скрипта GET-параметр `order` подвергается проверке, и если его значение равно одному из столбцов, присутствующих в таблице, в переменную `$order` помещается имя столбца. В противном случае переменная `$order` принимает значение по умолчанию, равное `"name"`. Такая громоздкая проверка необходима для того, чтобы избежать SQL-инъекции. При включенном режиме `register_globals` интерпретатор PHP автоматически создает для `GET`-параметров переменные. То есть при обращении к скрипту появляется возможность передать через параметр `order` SQL-инъекцию. Поэтому при работе с методами GET, POST или COOKIE необходимо всегда явно проверять переменные, которые передаются от одной страницы к другой, т. к. в этих методах передачи одним из посредников выступает машина клиента, где передаваемые данные могут подвергнуться подделке.

28.2.9. Параметризация SQL-запросов

В листинге 28.33 приводился пример работы с классом `mysqli_stmt`, реализующим отложенные запросы. Результирующая таблица может быть импортирована непосредственно в переменные PHP. Другой интересной особенностью класса `mysqli_stmt` является создание параметризованных запросов. Неизвестные переменные в таких запросах обозначаются при помощи знака вопроса, а значения могут быть подставлены посредством метода `bind_param()`. В листинге 28.46 демонстрируется создание отложенного `INSERT`-запроса вставки данных в таблицу `catalogs`.

Листинг 28.46. Параметризация SQL-запроса

```
<?php
  // Устанавливаем соединение с сервером MySQL
  require_once("config.php");

  // Создаем отложенный запрос
  $stmt =  $mysqli->stmt_init();

  // Формируем подготовленный запрос
  $query = "INSERT INTO catalogs
            VALUES (NULL, ?)";
```

```
  if ($stmt = $mysqli->prepare($query))
  {
    // Подставляем два строковых параметра в SQL-запрос $query
    $stmt->bind_param('s', $name);
    $name = "Периферия";

    // Выполняем запрос
    if (!$stmt->execute()) exit($stmt->error);

    // Освобождаем дескриптор подготовленного запроса
    $stmt->close();
  }

  // Закрываем соединение с сервером
  $mysqli->close();
?>
```

В качестве параметра выступает строка "Периферия", которая передается через переменную $name. Переменная $name передается в качестве второго параметра метода bind_param(). В качестве первого параметра передается строка, каждый символ которой обозначает тип последующих параметров:

- i — для целочисленного типа;
- d — для типа с плавающей точкой;
- s — для строки;
- b — для поля типа BLOB.

Таким образом, для запроса с двумя параметрами получаем:

```
INSERT INTO catalogs VALUES (?, ?)
```

Вызов метода bind_param() мог бы выглядеть следующим образом:

```
$stmt->bind_param('is', $id_catalog, $name);
```

Переменная $id_catalog при этом могла бы принимать целочисленное значение, например, 0, а переменная $name — строку с именем каталога. Важно помнить, что параметры передаются по ссылке, а не по значению, т. е. следующий вызов метода bind_param() вызовет ошибку.

```
$stmt->bind_param('is', 0, "Переменная");
```

Метод принимает только переменные, значения которых используются в момент вызова метода execute(). Это позволяет определять параметры в переменных даже после вызова метода bind_param().



Глава 29

FTP-менеджер

Протокол передачи файлов FTP (File Transfer Protocol) — один из старейших прикладных протоколов, появившийся задолго до Web в 1971 году. До начала 90-х годов на долю FTP приходилось половина трафика Интернета. Данный протокол и по сегодняшний день используется для распространения программного обеспечения и доступа к удаленным хостам, например, для обновления файлов на сайте.

> ***Замечание***
>
> Протокол FTP описан в документе RFC 959, доступном по ссылке **http://www.faqs.org/rfcs/rfc959.html**.

FTP-менеджер должен предоставлять следующие возможности:

- свободное перемещение по структуре FTP-сервера;
- предоставление информации о владельцах, группах, правах доступа к файлам и каталогам;
- создание, переименование, удаление и изменение прав доступа к каталогам;
- загрузка на сервер и с сервера файлов, а также их переименование и изменение прав доступа на сервере.

29.1. Функции для работы с FTP-сервером

Соединение с сервером устанавливается при помощи функции `ftp_connect()`, имеющей следующий синтаксис:

```
ftp_connect($ftp_server [, $port [, $timeout]])
```

Функция принимает в качестве первого параметра *`$ftp_server`* адрес FTP-сервера. Второй необязательный параметр *`$port`* может передавать номер порта, по которому необходимо установить соединение. По умолчанию соединение устанавливается стандартно, через порт 21. При помощи третьего параметра *`$timeout`* можно указать время в секундах, в течение которого функция будет ожидать ответа от сервера. По умолчанию, если данный параметр не задан, функция ожидает ответ в течение 90 секунд.

> ***Замечание***
>
> При указании адреса *`$ftp_server`* не следует указывать префикса http:// или ftp://, поскольку помимо доменного имени в качестве параметра можно использовать IP-

адрес сервера. Зачастую доступ к локальному FTP-серверу осуществляется именно по IP-адресу, отличному от того, который используется для Web-сервера, поэтому если установить соединение не удается, следует уточнить адрес `$ftp_server` у службы технической поддержки сервера.

В случае если установить соединение не удалось, функция возвращает `FALSE`. При успешном соединении с FTP-сервером функция возвращает дескриптор соединения, который далее используется для работы с FTP-сервером.

После установки соединения с FTP-сервером осуществляется регистрация пользователя при помощи функции `ftp_login()`, имеющей следующий синтаксис:

```
ftp_login($ftp_handle, $ftp_user, $ftp_password)
```

Функция принимает три параметра: дескриптор потока `$ftp_handle`, возвращаемый функцией `ftp_connect()`, имя пользователя `$ftp_user` и его пароль `$ftp_password`. В случае успешной регистрации на FTP-сервере функция возвращает `TRUE`, в противном случае — `FALSE`.

Закрыть FTP-соединение можно при помощи функции `ftp_close()`, которая имеет следующий синтаксис:

```
ftp_close($ftp_handle)
```

В качестве единственного аргумента `$ftp_handle` функция принимает дескриптор открытого FTP-соединения. В листинге 21.1 представлен скрипт, осуществляющий FTP-соединение с удаленным сервером.

Замечание

Если осуществляется доступ к анонимному публичному FTP-серверу, то в качестве имени пользователя можно ввести "anonymous", а в качестве пароля — адрес электронной почты.

Листинг 29.1. Соединение с FTP-сервером, ftp_connect.php

```
<?php
  // Имя пользователя
  $ftp_user = "name";
  // Пароль
  $ftp_password = "password";
  // Сервер
  $ftp_server = "ftp.server.ru";
  // Устанавливаем время исполнения скрипта 120 с
  @set_time_limit(120);
  // Устанавливаем соединение с FTP-сервером
  $ftp_handle = ftp_connect($host);
  if (!$ftp_handle)
  {
    exit("Невозможно установить соединение с FTP-сервером");
```

```
  }
  if(!@ftp_login($ftp_handle, $ftp_user, $ftp_password))
  {
    exit("Ошибка авторизации на FTP-сервере");
  }
?>
```

Для установки соединения в переменные `$ftp_server`, `$ftp_user` и `$ftp_password` необходимо поместить адрес FTP-сервера, имя пользователя и его пароль соответственно.

> ***ЗАМЕЧАНИЕ***
>
> Для дальнейшей работы с протоколом FTP код из листинга 29.1 удобно поместить в отдельный файл config/ftp_connect.php, который далее следует подключать ко всем скриптам при помощи инструкции `require_once()`, чтобы не переписывать код установки соединения в каждом скрипте.

В табл. 29.1 приводится список наиболее часто используемых функций для работы с FTP-сервером.

> ***ЗАМЕЧАНИЕ***
>
> С полным списком функций можно ознакомиться в официальном руководстве по PHP. Многие функции являются специфическими и зависят от реализации конкретного FTP-сервера. В табл. 29.1 приводятся наиболее распространенные функции, которые поддерживаются подавляющим большинством серверов.

Таблица 29.1. *Функции для работы с FTP-сервером*

Функция	Описание
`ftp_cdup($ftp_handle)`	Осуществляет переход к родительскому каталогу
`ftp_chdir($ftp_handle, $directory)`	Осуществляет переход в каталог, указанный в параметре `$directory`
`ftp_chmod($ftp_handle, $mode, $filename)`	Устанавливает файлу `$filename` права доступа `$mode`
`ftp_close($ftp_handle)`	Закрывает соединение с FTP-сервером
`ftp_connect($ftp_server [, $port [, $timeout]])`	Устанавливает соединение с FTP-сервером `$ftp_server` через порт `$port` (если не указано, используется 21 порт) и ожидает ответа в течение `$timeout` (если не указано, соединение устанавливается в течение 90 секунд)
`ftp_delete($ftp_handle, $filename)`	Удаляет файл `$filename`
`ftp_exec($ftp_handle, $command)`	Выполняет команду `$command` на FTP-сервере. Функция позволяет выполнять специфические команды FTP-сервера, для которых не предусмотрены специальные функции

***Таблица 29.1** (продолжение)*

Функция	Описание
`ftp_fget($ftp_handle, $handle, $remote_file, $mode)`	Скачивает с сервера файл `$remote_file` и сохраняет его в предварительно открытом файле с дескриптором `$handle`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_fput($ftp_handle, $remote_file, $handle, $mode)`	Загружает на сервер предварительно открытый файл с дескриптором `$handle`. Файл размещается по пути `$remote_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_get($ftp_handle, $local_file, $remote_file, $mode)`	Скачивает с сервера файл `$remote_file` и сохраняет его содержимое в локальном файле `$local_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_login($ftp_handle, $ftp_name, $ftp_password)`	Осуществляет вход на сервер с именем `$ftp_name` и паролем `$ftp_password`
`ftp_mdtm($ftp_handle, $remote_file)`	Возвращает время последней модификации файла `$remote_file`
`ftp_mkdir($ftp_handle, $directory)`	Создает директорию с именем `$directory`
`ftp_nb_continue ($ftp_handle)`	Функция, используемая совместно с функциями `ftp_nb_fget()`, `ftp_nb_fput()`, `ftp_nb_get()` и `ftp_nb_put()`, которые, в свою очередь, применяются для асинхронного режима передачи файлов
`ftp_nb_fget ($ftp_handle, $handle, $remote_file, $mode)`	Скачивает с сервера файл `$remote_file` и сохраняет его в предварительно открытом файле с дескриптором `$handle`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_nb_fput ($ftp_handle, $remote_file, $handle, $mode)`	Загружает на сервер предварительно открытый файл с дескриптором `$handle`. Файл размещается по пути `$remote_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_nb_get($ftp_handle, $local_file, $remote_file, $mode)`	Скачивает с сервера файл `$remote_file` и сохраняет его содержимое в локальном файле `$local_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_nb_put($ftp_handle, $remote_file, $local_file, $mode)`	Загружает на сервер файл `$local_file` и сохраняет его под именем `$remote_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов

***Таблица 29.1** (окончание)*

Функция	Описание
`ftp_nlist($ftp_handle, $directory)`	Возвращает список файлов в каталоге `$directory`
`ftp_pasv($ftp_handle, $pasv)`	Если параметр `$pasv` принимает значение `TRUE`, передача данных осуществляется в пассивном режиме, иначе действует активный режим
`ftp_put($ftp_handle, $remote_file, $local_file, $mode)`	Загружает на сервер файл `$local_file` и сохраняет его под именем `$remote_file`. Режим `$mode` может принимать два значения: `FTP_ASCII` — для текстовых файлов или `FTP_BINARY` — для бинарных файлов
`ftp_pwd($ftp_handle)`	Возвращает путь к текущему каталогу
`ftp_raw($ftp_handle, $command)`	Отправляет FTP-серверу произвольную команду `$command`, допускаемую FTP-протоколом
`ftp_rawlist($ftp_handle, $directory [, $recursive] )`	Отправляет FTP-серверу команду `LIST`, которая возвращает содержимое каталога `$directory`. Если параметр `$recursive` принимает значение `TRUE`, серверу отправляется команда `LIST -R`, которая требует рекурсивного спуска по подкаталогам и вывода их содержимого
`ftp_rename($ftp_handle, $oldname, $newname)`	Переименовывает файл или каталог `$oldname` в `$newname`
`ftp_rmdir($ftp_handle, $directory)`	Удаляет каталог `$directory`, если он не содержит вложенных каталогов и файлов
`ftp_site($ftp_handle, $command)`	Отправляет команду `$command` на FTP-сервер
`ftp_size($ftp_handle, $remote_file)`	Возвращает размер файла `$remote_file` в байтах
`ftp_systype($ftp_handle)`	Возвращает тип операционной системы, под управлением которой работает сервер

Перед тем как приступить к созданию FTP-менеджера, рассмотрим два небольших Web-приложения, демонстрирующих работу FTP-функций.

29.2. Какой объем жесткого диска занимает сайт?

Для подсчета объема памяти, занимаемой сайтом на жестком диске, при помощи FTP удобно, установив соединение, рекурсивно спуститься по всем подкаталогам сайта, подсчитывая объем встречающихся файлов. Для рекурсивного спуска создадим функцию `get_ftp_size()`, которая будет принимать три параметра:

- `$ftp_handle` — дескриптор FTP-соединения;
- `$dir` — текущий каталог;
- `$global_size` — текущий размер файлов на FTP-сервере.

В листинге 29.2 представлена возможная реализация функции `get_ftp_size()`.

Листинг 29.2. Функция рекурсивного обхода FTP-сервера

```
<?php
  // Функция, подсчитывающая количество байтов, занимаемых каталогом $dir
  function get_ftp_size($ftp_handle, $dir, $global_size = 0)
  {
    $file_list = ftp_rawlist($ftp_handle, $dir);
    if(!empty($file_list))
    {
      foreach($file_list as $file)
      {
        // Разбиваем строку по пробельным символам
        list($acc,
             $bloks,
             $group,
             $user,
             $size,
             $month,
             $day,
             $year,
             $file) = preg_split("/[\s]+/", $file);
        if($acc[0] == 'd' && $file != ".." && $file != ".")
        {
          // Если перед нами каталог, рекурсивно вызываем
          // для него функцию get_ftp_size()
          $dir_new = trim($dir."/".$file,"/");
          $global_size = get_ftp_size($ftp_handle,
                                      $dir_new,
                                      $global_size);
        }
        else
        {
          // Если перед нами файл, учитываем его объем
          $global_size += $size;
        }
      }
    }
    return $global_size;
  }
?>
```

Для подсчета объема FTP-сервера достаточно вызвать функцию `get_ftp_size()`, передав ей дескриптор открытого FTP-соединения и имя каталога на сервере. В листинге 29.3 выводится размер всех файлов на FTP-сервере (т. к. указывается корень `/`), однако функция позволяет узнать размер произвольного каталога, если указать путь к нему во втором параметре.

Листинг 29.3. Использование функции `get_ftp_size()`

```
<?php
  // Устанавливаем соединение с FTP-сервером
  require_once("config/ftp_config.php");

  // Выводим размер файлов на FTP-сервере
  echo get_ftp_size($ftp_handle, "/");
?>
```

29.3. Перенос сайта с одного хоста на другой

Для перемещения файлов с одного хоста на другой потребуется одновременно открыть два соединения: `$ftp_handle_to` — дескриптор FTP-сервера, на который переносятся файлы, и `$ftp_handle_from` — дескриптор FTP-сервера, с которого файлы перемещаются. В листинге 29.4 приводится функция `copy_ftp_list()`, которая переносит файлы с одного сервера из папки `$path_from` на другой сервер в папку `$path_to`.

Листинг 29.4. Перенос файлов с одного сервера на другой

```
<?php
  function copy_ftp_list($path_from,
                         $path_to,
                         $ftp_handle_to,
                         $ftp_handle_from)
  {
    //
    $list_form = ftp_rawlist($ftp_handle_from, $path_from);
    foreach($list_form as $file)
    {
      // Разбиваем строку по пробельным символам
      list($acc,
           $bloks,
           $group,
           $user,
           $size,
           $month,
```

```
      $day,
      $year,
      $filename) = preg_split("/[\s]+/", $file);
if($filename != "." && $filename != "..")
{
  if($acc[0] == 'd')
  {
    $next_dir_to = $path_to."/".$filename;
    $next_dir_to = preg_replace("|/+|","/",$next_dir_to);
    @ftp_mkdir($ftp_handle_to, $next_dir_to);
    ftp_chmod($ftp_handle_to,
              getchmod(substr($acc, 1)),
              $next_dir_to);
    // Копируем данные
    $next_dir_from = $path_from."/".$filename;
    $next_dir_from = preg_replace("|/+|","/",$next_dir_from);
    copy_ftp_list($next_dir_from,
                  $next_dir_to,
                  $ftp_handle_to,
                  $ftp_handle_from);
  }
  else
  {
    $file_from = $path_from."/".$filename;
    $file_from = preg_replace("|/+|","/",$file_from);
    $mk_file_to = $path_to."/".$dirname."/".$filename;
    $mk_file_to = preg_replace("|/+|","/",$mk_file_to);
    echo str_repeat("|", $lev)."--Копирование файла
         <b>$mk_file_to</b>...";
    flush();
    if(ftp_get($ftp_handle_from,"tf.fl",$file_from,FTP_BINARY) &&
       ftp_put($ftp_handle_to,$mk_file_to,"tf.fl",FTP_BINARY) &&
       ftp_chmod($ftp_handle_to,
                 getchmod(substr($acc, 1)),
                 $mk_file_to) &&
       unlink("tf.fl"))
    {
      echo "OK<br>";
    }
    else
    {
      echo "ERROR<br>";
```

```
          }
        }
      }
    }
  }
?>
```

Файлы последовательно загружаются с сервера-источника во временный файл tf.fl, из которого затем перемещаются на сервер-приемник. После переноса очередного файла временный файл tf.fl уничтожается. Для копирования прав доступа используется вспомогательная функция `getchmod()`, которая преобразует строковое представление прав доступа в восьмеричное число (листинг 29.5).

Листинг 29.5. Вспомогательная функция

```
<?php
  function getchmod($perm)
  {
    $chmod = 0;
    if($perm[0] == "r") $chmod = $chmod+0400;
    if($perm[1] == "w") $chmod = $chmod+0200;
    if($perm[2] == "x") $chmod = $chmod+0100;
    if($perm[3] == "r") $chmod = $chmod+0040;
    if($perm[4] == "w") $chmod = $chmod+0020;
    if($perm[5] == "x") $chmod = $chmod+0010;
    if($perm[6] == "r") $chmod = $chmod+0004;
    if($perm[7] == "w") $chmod = $chmod+0002;
    if($perm[8] == "x") $chmod = $chmod+0001;
    return $chmod;
  }
?>
```

В листинге 29.6 приводится пример использования функции `copy_ftp_list()`.

Листинг 29.6. Перенос файлов с одного хоста на другой

```
<?php
  // Подключаем вспомогательные файлы
  require_once("function.copy_ftp_list.php");

  // Сервер назначения
  $ftp_handle_to = ftp_connect("XXX.XXX.XXX.XXX");
  if (!$ftp_handle_to)
```

```
  {
    exit(" Ошибка соединения с FTP-сервером назначения");
  }

  // Сервер-источник
  $ftp_handle_from = ftp_connect("XXX.XXX.XXX.XXX");
  if (!$ftp_handle_from)
  {
    exit(" Ошибка соединения с FTP-сервером источника");
  }

  // Копируем файлы с одного сервера на другой
  copy_ftp_list("/", "/", $ftp_handle_to, $ftp_handle_from);
?>
```



Глава 30

Электронная почта

Почта является основным средством общения для миллионов пользователей Интернета. Большинство пользователей Интернета привыкло именно к этому виду общения, поэтому для успешной разработки сайтов необходимо владеть приемами отправки и получения почты через PHP.

30.1. Отправка почтового сообщения

Отправка почты средствами PHP осуществляется при помощи функции `mail()`, которая имеет следующий синтаксис:

```
mail($to, $subject, $body [, $headers] [, $parameters])
```

Эта функция принимает следующие аргументы:

- `$to` — адрес электронной почты получателя;
- `$subject` — тема сообщения;
- `$message` — текст сообщения;
- `$headers` — дополнительные заголовки, которые можно задать в сообщении;
- `$parameters` — дополнительные параметры, которые можно задать в сообщении.

Если не указывается четвертый параметр `$headers`, письмо не снабжается никакими дополнительными почтовыми заголовками. Однако очень часто требуется изменить формат письма с обычного текста (`text/plain`) на HTML-формат (`text/html`) или указать кодировку сообщения. Установка формата письма и его кодировки осуществляется при помощи почтовых заголовков `Content-Type` и `charset`, соответственно.

```
Content-Type: text/html; charset=KOI8-R\r\n
```

> ***Замечание***
>
> Для отправки почтового сообщения в кодировке cp1251, вместо KOI8-R следует прописать windows-1251.

Таким образом, скрипт, выполняющий отправку почтового сообщения, может выглядеть так, как это представлено в листинге 30.1.

Листинг 30.1. Отправка почтового сообщения при помощи функции `mail()`

```
<?php
  $theme = "Отчет с сайта";
  $theme =  convert_cyr_string($theme, 'w', 'k');
  $message = "<html>
              <head></head>
              <body>
              Письмо отправлено - ".date("d.m.Y H:i:s")."<br>
              Размер скрипта отправителя - ".filesize($_SERVER['PHP_SELF'])."
              </body>
              </html>";
  $message =  convert_cyr_string($message, 'w', 'k');
  $headers = "Content-Type: text/html; charset=KOI8-R\r\n";
  if(mail($to, $subject, $message, $headers))
  {
    echo "Письмо успешно отправлено";
  }
  else
  {
    echo "Произошла ошибка - письмо не отправлено";
  }
?>
```

При получении письма в качестве отправителя будет подставлен адрес сервера. Для того чтобы избежать этого, в качестве обратного адреса можно назначить произвольный адрес при помощи почтового заголовка `From`:

`From:` *`name <e-mail>`*

Вместо *`name`* указывается имя, которое будет отображаться клиентским почтовым агентом как имя отправителя, а *`e-mail`* содержит обратный почтовый адрес. Так, строки формирования переменной `$headers` могут выглядеть следующим образом (листинг 30.2).

Листинг 30.2. Формирование почтовых заголовков

```
<?php
  $headers = "Content-Type: text/html; charset=KOI8-R\r\n";
  $headers .= "From: server <someone@somewhere.ru>\r\n\r\n";
?>
```

При отправке электронного письма, снабженного почтовыми заголовками, из листинга 30.2 оно будет представлено как письмо от пользователя `server` с электронным адресом `someone@somewhere.ru`.

30.2. Рассылка писем

При реализации рассылки писем можно следовать двумя путями:

- осуществлять рассылку писем в цикле, последовательно перебирая почтовые адреса — к такому приему прибегают в том случае, если письмо для каждого пользователя уникально, и нельзя всем послать одно и то же шаблонное письмо;
- осуществлять рассылку писем, указав через почтовые заголовки список адресатов. К этому приему прибегают в том случае, если адресатам следует отправить одно и то же письмо.

В листинге 30.3 представлен вариант кода, осуществляющего рассылку с использованием первого приема.

Листинг 30.3. Рассылка писем в цикле

```
<?php
  // Тема письма
  $theme = "Тема письма";
  // Содержимое письма
  $body = "Тело письма";
  // Имя файла с e-mail-адресами
  $filename = "mail.txt";
  // Читаем содержимое файла в массив $listemail
  // при помощи функции file()
  $listemail = file($filename);
  // В цикле осуществляем отправку писем
  foreach($listemail as $email)
  {
    mail(trim($email), $theme, $body);
  }
?>
```

Однако вызов функции `mail()` является достаточно трудоемкой и ресурсозатратной процедурой. Отправка более сотни писем таким способом может привести к значительной временной задержке, что может быть неприемлемо, если скрипт отправки связан со скриптом добавления нового сообщения в форуме или на доске объявлений. В этом случае лучше прибегнуть к отправке почтового сообщения с перечислением адресатов в почтовом заголовке `Cc`, если необходимо, чтобы адресаты видели, кому, помимо их, было разослано письмо, или заголовок `Bcc`, если этого не требуется. Скрипт, реализующий данную рассылку, может выглядеть так, как это представлено в листинге 30.4.

> ***ЗАМЕЧАНИЕ***
>
> Число почтовых заголовков `Bcc` и `Cc` не ограничено, каждый заголовок принимает в качестве значения электронный адрес одного адресата. Например, `"Bcc: somebody@mail.ru\r\n"`.

Листинг 30.4. Рассылка писем с применением почтовых заголовков

```
<?php
  // Тема письма
  $theme = "Тема письма";
  // Содержимое письма
  $body = "Тело письма";
  // Имя файла с e-mail-адресами
  $filename = "mail.txt";
  // Читаем содержимое файла в массив $listemail
  // при помощи функции file()
  $listemail = file($filename);
  // В цикле формируем почтовые заголовки
  $header = "";
  foreach($listemail as $email)
  {
    $header .= "Bcc: ".trim($email)."\r\n";
  }
  $header .= "\r\n";
  mail(trim($listemail[0]), $theme, $body, $header);
?>
```

Как видно из листинга 30.4, в цикле формируются лишь почтовые заголовки. Функция `mail()`, отправляющая письма, вызывается только один раз.

30.3. Отправка писем с вложением

Для того чтобы отправить почтовое сообщение с прикрепленным к нему файлом, необходимо особым способом подготовить текст письма и снабдить его соответствующими почтовыми заголовками. Для отправки почтового сообщения создадим HTML-форму, состоящую из двух текстовых полей для адреса назначения `mail_to`, темы сообщения `mail_subject`, текстовой области `mail_msg` для ввода содержимого письма, поля типа `file` для выбора файла отправки и кнопки, позволяющей отправить данные из HTML-формы обработчику (листинг 30.5).

Листинг 30.5. HTML-форма для отправки почтового сообщения

```
<?php
if(!empty($_POST))
{
  // Обработчик HTML-формы
  include "handler.php";
}
?>
<table>
```

```
<form enctype='multipart/form-data' method='post'>
<tr>
  <td width=50%>To:</td>
  <td align='right'>
    <input type='text' name='mail_to' maxlength='32'>
  </td>
</tr>
<tr>
  <td width=50%>Subject:</td>
  <td align='right'>
    <input type='text' name='mail_subject' maxlength='64'>
  </td>
</tr>
<tr>
  <td colspan='2'>
   Сообщение:<br><textarea cols='50' rows='8' name='mail_msg'></textarea>
  </td>
</tr>
<tr>
  <td width='50%'>Photo:</td>
  <td align='right'>
    <input type='file' name='mail_file' maxlength='64'>
  </td>
</tr>
<tr><td colspan='2'><input type='submit' value='Отправить'></td></tr>
</form>
</table>
```

Внешний вид HTML-формы из листинга 30.5 представлен на рис. 30.1.

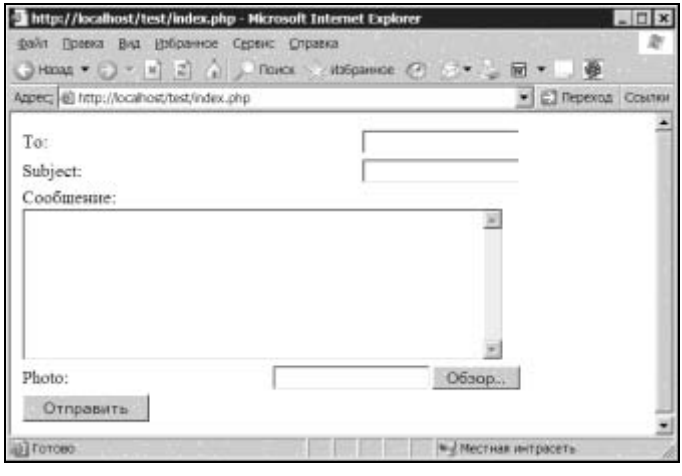


Рис. 30.1. HTML-форма для отправки почтового сообщения

В качестве обработчика формы служит файл handler.php, который включается в начале формы и имеет следующее содержание (листинг 30.6).

Листинг 30.6. Обработчик HTML-формы

```
<?php
  if(empty($_POST['mail_to'])) exit("Введите адрес получателя");
  // проверяем правильность заполнения с помощью регулярного выражения
  $pattern = "/^[0-9a-z_]+@[0-9a-z_^\.]+\.[a-z]{2,6}$/i";
  if (!preg_match($pattern, $_POST['mail_to']))
  {
   exit("Введите адрес в виде somebody@server.com");
  }
  $_POST['mail_to'] = htmlspecialchars(stripslashes($_POST['mail_to']));
  $_POST['mail_subject'] =
                htmlspecialchars(stripslashes($_POST['mail_subject']));
  $_POST['mail_msg'] =
                    htmlspecialchars(stripslashes($_POST['mail_msg']));
  $picture = "";
  // Если поле выбора вложения не пустое - закачиваем его на сервер
  if (!empty($_FILES['mail_file']['tmp_name']))
  {
    // Закачиваем файл
    $path = $_FILES['mail_file']['name'];
    if (copy($_FILES['mail_file']['tmp_name'], $path)) $picture = $path;
  }
  $thm = $_POST['mail_subject'];
  $msg = $_POST['mail_msg'];
  $mail_to = $_POST['mail_to'];
  // Отправляем почтовое сообщение
  if(empty($picture)) mail($mail_to, $thm, $msg);
  else send_mail($mail_to, $thm, $msg, $picture);
  // Вспомогательная функция для отправки почтового сообщения с вложением
  function send_mail($to, $thm, $html, $path)
  {
    $fp = fopen($path,"r");
    if (!$fp)
    {
      print "Файл $path не может быть прочитан";
      exit();
    }
    $file = fread($fp, filesize($path));
    fclose($fp);
```

```
    $boundary = "--".md5(uniqid(time())); // генерируем разделитель
    $headers .= "MIME-Version: 1.0\n";
    $headers .="Content-Type: multipart/mixed; boundary=\"$boundary\"\n";
    $multipart .= "--$boundary\n";
    $kod = 'koi8-r'; // или $kod = 'windows-1251';
    $multipart .= "Content-Type: text/html; charset=$kod\n";
    $multipart .= "Content-Transfer-Encoding: Quot-Printed\n\n";
    $multipart .= "$html\n\n";

    $message_part = "--$boundary\n";
    $message_part .= "Content-Type: application/octet-stream\n";
    $message_part .= "Content-Transfer-Encoding: base64\n";
    $message_part .= "Content-Disposition: attachment; filename =
\"".$path."\"\n\n";
    $message_part .= chunk_split(base64_encode($file))."\n";
    $multipart .= $message_part."--$boundary--\n";

    if(!mail($to, $thm, $multipart, $headers))
    {
      exit("К сожалению, письмо не отправлено");
    }
  }
  // Автоматический переход на главную страницу форума
  echo "<HTML><HEAD>
    <META HTTP-EQUIV='Refresh' CONTENT='0; URL=".$_SERVER['PHP_SELF']."'>
        </HEAD></HTML>";
?>
```

После проверки корректности ввода проверяется, прикрепил ли пользователь файл к сообщению. Если файл отсутствует — письмо отправляется при помощи стандартной функции `mail()`, при наличии вложения, оно загружается из временного каталога в текущий, а письмо отправляется при помощи собственной функции `send_mail()`. Первые три параметра функции `send_mail()` совпадают с параметрами функции `mail()`. В качестве четвертого параметра передается имя файла, который необходимо прикрепить к почтовому сообщению. Далее функция подготавливает тело сообщения и почтовые заголовки для передачи сообщения с прикрепленным файлом.

30.4. Отправка писем со встроенными изображениями

При отправке писем в HTML-формате часто возникает задача отправки дополнительных изображений, которые бы встраивались в HTML-код, однако не прикреплялись бы в виде вложения. Для этого можно использовать функцию из листинга 30.7.

Листинг 30.7. Отправка встроенных изображений

```
<?php
  function htmlimgmail($mail_to, $thema, $html, $path)
  {
    $EOL = "\n";
    $boundary     = "--".md5(uniqid(time()));
    $headers    = "MIME-Version: 1.0;$EOL";
    $headers .= "From: somebody@somewhere.ru$EOL";
    $headers   .= "Content-Type: multipart/related; ".
                  "boundary=\"$boundary\"$EOL";

    $multipart  = "--{$boundary}$EOL";
    $multipart .= "Content-Type: text/html; charset=koi8-r$EOL";
    $multipart .= "Content-Transfer-Encoding: 8bit$EOL";
    $multipart .= $EOL;
    if($EOL == "\n")  $multipart .= str_replace("\r\n", $EOL, $html);
    $multipart .= $EOL;

    if (!empty($path))
    {
      for($i = 0; $i < count($path); $i++)
      {
        $file = file_get_contents($path[$i]);
        $name = basename($path[$i]);
        $multipart .=  "$EOL--$boundary$EOL";
        $multipart .= "Content-Type: image/jpeg; name=\"$name\"$EOL";
        $multipart .= "Content-Transfer-Encoding: base64$EOL";
        $multipart .= "Content-ID: <".md5($name).">$EOL";
        $multipart .= $EOL;
        $multipart .= chunk_split(base64_encode($file), 76, $EOL);
      }
    }

    $multipart .= "$EOL--$boundary--$EOL";
    if(!is_array($mail_to))
    {
      // Письмо отправляется одному адресату
      if(!mail($mail_to, $thema, $multipart, $headers))
        return False;
      else
        return True;
      exit;
```

```
    }
    else
    {
      // Письмо отправляется на несколько адресов
      foreach($mail_to as $mail)
      {
        mail($mail, $thema, $multipart, $headers);
      }
    }
  }
?>
```

В листинге 30.8 приводится пример использования функции `htmlimgmail()`. Функция принимает в качестве первого параметра `$mail_to` электронный адрес, на который необходимо направить письмо (или массив электронных адресов, если их несколько), через второй параметр `$thema` передается тема сообщения, третий параметр содержит тело письма в HTML-формате. Последний параметр содержит массив с путями файлов. Путь должен указываться от каталога, из которого скрипт вызывается, т. к. функции `htmlimgmail()` потребуется содержимое каждого из изображений. Из HTML-кода ссылаться на встроенные изображения можно при помощи префикса cid, после которого следует уникальный хэш изображения, задаваемый в почтовом заголовке `Content-ID`. Данный хэш вычисляется из названия файла при помощи функции `md5()`.

Листинг 30.8. Использование функции `htmlimgmail()`

```
<?php
  // Отправляем почтовое сообщение
  $picture[0] = "s_20040815135808.jpg";
  $picture[1] = "s_20040815135939.jpg";
  $mail_to = "sombody@somewhere.ru";
  $thm     = convert_cyr_string("Тема сообщения", "w", "k");
  $html = "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML ".
          "4.01 Transitional//EN\">
           <html>
             <head><title>Почтовая рассылка</title></head>
             <body><img src='cid:".md5($picture[0])."' border='0'>Тело
сообщения<br><br><img src='cid:".md5($picture[1])."' border='0'></body>
           </html>";
  $html = convert_cyr_string($html, "w", "k");
  if(send_mail($mail_to, $thm, $html, $picture))
    echo "Успех ".date("d.m.Y H:i");
  else
    echo "Не отправлено";
?>
```



Глава 31

Динамические изображения. Библиотека GDLib

Библиотека GDLib предназначена для создания динамических изображений: программных изображений или наложения текста и графики на уже существующее изображение.

> ***Замечание***
>
> Официальный сайт библиотеки GDLib находится по адресу **http://www.libgd.org/**. По умолчанию расширение GDLib не подключено, для его подключения необходимо снять комментарий напротив строки `extension=php_gd2.dll` в конфигурационном файле php.ini. Для подключения exif-функций, возвращающих справочную информацию о файлах формата JPEG и TIFF, необходимо также подключить библиотеки php_mbstring.dll и php_exif.dll.

Все функции, которые необходимы для создания динамических изображений, будут условно разбиты на восемь разделов:

- информационные функции;
- функции создания изображений;
- функции сохранения и вывода изображений;
- функции преобразования изображений;
- функции для работы с цветом;
- функции рисования;
- функции настройки рисования;
- функции для работы с текстом.

> ***Замечание***
>
> При работе с функциями GDLib следует учитывать, что координаты абсцисс *X* и ординат *Y* отсчитываются так, как это принято в большинстве графических систем (из левого верхнего угла), а не в декартовой системе координат (из левого нижнего угла).

31.1. Информационные функции

Информационные функции предоставляют сведения о текущей версии библиотеке GDLib, а также параметры изображений. В табл. 31.1 приводится список функций, относящихся к данной группе.

***Таблица 31.1.** Информационные функции библиотеки GDLib*

Функция	Назначение
`gd_info()`	Возвращает ассоциативный массив с информацией об установленной библиотеке GD
`exif_imagetype($filename)`	Читает несколько первых байтов файла `$filename` и определяет формат изображения
`exif_read_data($filename [, $sections [, $arrays [, $thumbnail]]])`	Читает заголовки файла формата JPEG и TIFF и возвращает ассоциативный массив со свойствами секций этого файла
`exif_thumbnail($filename [, &$width [, &$height [, &$imagetype]]])`	Возвращает встроенные эскизы (уменьшенные изображения) файлов форматов JPEG и TIFF
`getimagesize($filename [, &$imageinfo])`	Возвращает размер в пикселах и тип файла `$filename`, кроме того, может возвращать дополнительную информацию об изображении в параметре `$imageinfo`
`imagesx($image)`	Возвращает ширину изображения в пикселах. В качестве параметра `$image` функция принимает дескриптор открытого изображения
`imagesy($image)`	Возвращает высоту изображения в пикселах. В качестве параметра `$image` функция принимает дескриптор открытого изображения
`image_type_to_mime_type ($imagetype)`	Определяет MIME-типы для констант `imagetype`, возвращаемых функциями `getimagesize()`, `exif_read_data()`, `exif_thumbnail()`, `exif_imagetype()`
`imagetypes()`	Возвращает битовое поле, соответствующее форматам изображений, которые поддерживаются текущей реализацией PHP и библиотеки GD. Возвращаемое битовое поле может содержать следующие биты: `IMG_GIF|IMG_JPG|IMG_PNG|IMG_WBMP`

Рассмотрим более подробно наиболее часто используемые функции из табл. 31.1.

31.1.1. Текущая версия библиотеки GDLib

Функция `gd_info()` возвращает ассоциативный массив, элементы которого позволяют определить версию библиотеки, а также поддерживаемые форматы. В листинге 31.1 демонстрируется вызов функции.

Листинг 31.1. Вызов функции `gd_info()`

```
<?php
  $arr = gd_info();
  echo "<pre>";
```

```
  print_r($arr);
  echo "</pre>";
?>
```

В качестве результата скрипт из листинга 31.1 может вернуть следующую информацию:

```
Array
(
    [GD Version] => bundled (2.0.28 compatible)
    [FreeType Support] => 1
    [FreeType Linkage] => with freetype
    [T1Lib Support] => 1
    [GIF Read Support] => 1
    [GIF Create Support] => 1
    [JPG Support] => 1
    [PNG Support] => 1
    [WBMP Support] => 1
    [XPM Support] =>
    [XBM Support] => 1
    [JIS-mapped Japanese Font Support] =>
)
```

Первый элемент массива "GD Version" сообщает о том, что текущей является версия 2.0.28. Элементы возвращаемого функцией `gd_info()` массива описываются в табл. 31.2.

***Таблица 31.2.** Атрибуты и значения возвращаемого ассоциативного массива*

Элемент	Описание
`GD Version`	Версия библиотеки GD
`Freetype Support`	Возвращает `TRUE`, если установлена поддержка шрифтов Freetype
`Freetype Linkage`	Описывает, каким образом включена поддержка шрифтов Freetype. Возможны следующие значения: `'with freetype'`, `'with TTF library'` и `'with unknown library'`. Этот параметр существует, только если `Freetype Support = TRUE`
`T1Lib Support`	Возвращает `TRUE`, если включена поддержка шрифтов PostScript Type1
`GIF Read Support`	Возвращает `TRUE`, если включена поддержка чтения файлов формата GIF
`GIF Create Support`	Возвращает `TRUE`, если включена возможность создания изображений в формате GIF
`JPG Support`	Возвращает `TRUE`, если включена поддержка формата JPEG
`PNG Support`	Возвращает `TRUE`, если включена поддержка формата PNG

***Таблица 31.2** (окончание)*

Элемент	Описание
`WBMP Support`	Возвращает `TRUE`, если включена поддержка формата WBMP
`XBM Support`	Возвращает `TRUE`, если включена поддержка формата XBM
`JIS-mapped Japanese Font Support`	Возвращает `TRUE`, если включена поддержка японских иероглифов

ЗАМЕЧАНИЕ

О параметрах библиотеки GDLib можно узнать также из отчета функции `phpinfo()`.

31.1.2. Формат файла

Определить формат файла позволяет функция `exif_imagetype()`, которая читает несколько первых байтов файла и определяет формат изображения. Если формат определен, то возвращается константа, соответствующая типу файла (табл. 31.3), в противном случае возвращается `FALSE`.

***Таблица 31.3.** Соответствие констант идентификаторов форматов файлов*

Возвращаемое значение	Соответствующая значению константа
1	`IMAGETYPE_GIF`
2	`IMAGETYPE_JPEG`
3	`IMAGETYPE_PNG`
4	`IMAGETYPE_SWF`
5	`IMAGETYPE_PSD`
6	`IMAGETYPE_BMP`
7	`IMAGETYPE_TIFF_II`
8	`IMAGETYPE_TIFF_MM`
9	`IMAGETYPE_JPC`
10	`IMAGETYPE_JP2`
11	`IMAGETYPE_JPX`
12	`IMAGETYPE_JB2`
13	`IMAGETYPE_SWC`
14	`IMAGETYPE_IFF`
15	`IMAGETYPE_WBMP`
16	`IMAGETYPE_XBM`

31.1.3. Размер файла

Для получения размера файла предназначена функция `getimagesize()`, которая принимает в качестве первого параметра путь к файлу (допускается в том числе и сетевой путь, начинающийся с http://) и возвращает массив из четырех элементов: ширину изображения, высоту изображения, одно из значений, представленных в табл. 31.3, и строку формата `"width=x height=y"`, которая может быть использована в HTML-теге `<IMG>` (вывод изображений). Для изображений формата JPEG возвращаются три дополнительных элемента: `chanel`, `bits` и `mime`. Элемент `chanel` содержит значение 3 для RGB-изображений и 4 для CMYK-изображений; `bits` — количество бит для каждого из цветов; `mime` возвращает MIME-тип файла, например, для JPEG-файлов будет возвращена строка `"image/jpeg"`.

В листинге 31.2 демонстрируется использование функции `getimagesize()` применительно к JPEG-изображению, которое выводится в окно браузера.

Листинг 31.2. Использование функции `getimagesize()`

```
<?php
  list($width, $height) = getimagesize("PICT0073.JPG");
  echo "<img src=PICT0073.JPG width=$width height=$height>";
?>
```

В листинге 31.2 размеры изображения определяются автоматически при помощи функции `getimagesize()` и подставляются в HTML-тег `<IMG>`. В листинге 31.3 представлен альтернативный вариант, в котором для указания размера изображений используется четвертый элемент массива, возвращаемого функций `getimagesize()`.

Листинг 31.3. Альтернативный способ указания размера в HTML-теге `<IMG>`

```
<?php
  list($width, $height, $type, $size) = getimagesize("PICT0073.JPG");
  echo "<img src=PICT0073.JPG $size>";
?>
```

Как будет показано далее, основная часть функций библиотеки GDLib работает с изображением по схеме, напоминающей схему работы с файлами. Изображение открывается при помощи одной из функций открытия, которые возвращают дескриптор изображения. Этот дескриптор передается всем функциям, которые осуществляют манипуляции с изображением (при помощи дескрипторов одно открытое изображение отличается от другого). После завершения всех операций изображение закрывается. При работе в таком режиме более удобными являются функции `imagesx()` и `imagesy()`, возвращающие соответственно высоту и ширину изображения. Однако в отличие от функции `getimagesize()` в качестве параметра эти функции принимают не имя файла, а дескриптор открытого изображения.

31.1.4. Получение MIME-типа файла

Аббревиатура MIME дословно расшифровывается как Multiporpose Internet Mail Extensions (многоцелевые расширения электронной почты). Это расширение было введено для обозначения форматов файлов, которые прикрепляются в качестве вложения к электронным письмам. В разных средах это позволяет автоматически подбирать приложения, которые способны адекватно открыть такой файл.

В настоящее время MIME-типы используются не только в почте, но и в Web-среде. Так, при отправке любого документа с сервера MIME-тип передается в HTTP-заголовке `Content-Type`. Это позволяет браузеру клиента правильно работать с текстом, HTML-страницами, графикой, музыкальными и видеофайлами. Например, для HTML-страниц посылается MIME-тип `text/html`, который сообщает браузеру, что ему пересылается HTML-страница для интерпретации. Однако если послать MIME-тип `text/plain`, соответствующий обычному тексту, браузер вместо интерпретации отобразит HTML-код страницы как есть.

Строка с MIME-типом состоит из двух частей: идентификатора типа и идентификатора подтипа, разделенных косой чертой `/`. Так для текстовой информации идентификатор типа принимает значение `text`, а идентификатор подтипа может принимать два значения: `html` и `plain`. Для изображений идентификатор типа принимает значение `image`, а идентификатор подтипа расшифровывает формат файла:

- `image/gif` — изображение в формате GIF;
- `image/jpeg` — изображение в формате JPEG;
- `image/png` — изображение в формате PNG

и т. п.

Запоминать значения MIME-типов необязательно, их всегда можно получить автоматически. Функция `image_type_to_mime_type()` принимает в качестве параметра значение константы из табл. 31.3 и возвращает соответствующий ей MIME-тип (листинг 31.4).

Листинг 31.4. Получение MIME-типа изображения

```
<?php
  list($width, $height, $type) = getimagesize("PICT0073.JPG");
  echo image_type_to_mime_type($type); // image/jpeg
?>
```

При использовании функции `getimagesize()` для получения MIME-типа можно воспользоваться элементом `mime` возвращаемого массива (листинг 31.5).

Листинг 31.5. Получение MIME-типа изображения

```
<?php
  $arr = getimagesize("PICT0073.JPG");
  echo $arr['mime']; // image/jpeg
?>
```

В табл. 31.4 приводится соответствие принимаемых функцией `image_type_to_mime_type()` значений и возвращаемых ею MIME-типов.

***Таблица 31.4.** Соответствие констант GDLib и MIME-типов*

Константа	Числовое значение	MIME-тип
IMAGETYPE_GIF	1	image/gif
IMAGETYPE_JPEG	2	image/jpeg
IMAGETYPE_PNG	3	image/png
IMAGETYPE_SWF	4	application/x-shockwave-flash
IMAGETYPE_PSD	5	image/psd
IMAGETYPE_BMP	6	image/bmp
IMAGETYPE_TIFF_II	7	image/tiff
IMAGETYPE_TIFF_MM	8	image/tiff
IMAGETYPE_JPC	9	application/octet-stream
IMAGETYPE_JP2	10	image/jp2
IMAGETYPE_JPX	11	application/octet-stream
IMAGETYPE_JB2	12	application/octet-stream
IMAGETYPE_SWC	13	application/x-shockwave-flash
IMAGETYPE_IFF	14	image/iff
IMAGETYPE_WBMP	15	image/vnd.wap.wbmp
IMAGETYPE_XBM	16	image/xbm

MIME-типы часто используются для отправки изображений в среде Web. В листинге 31.6 приводится скрипт, передающий изображение в окно браузера без использования HTML-тега `<IMG>`. Если перед телом изображения браузеру не отправить HTTP-заголовок `Content-Type` с соответствующим MIME-типом вместо изображения, браузер попытается вывести бинарное содержимое файла.

Листинг 31.6. Вывод изображения без использования HTML-тега `<IMG>`

```
<?php
  list($width, $height, $type) = getimagesize("PICT0073.JPG");
  // Отправляем HTTP-заголовок Content-type
  $header = "Content-type: ".image_type_to_mime_type($type);
  header($header);
  // Отправляем размер файла
  $header = "Content-lenght: ".filesize("PICT0073.JPG");
```

```
  header($header);
  // Отправляем содержимое файла
  echo file_get_contents("PICT0073.JPG");
?>
```

31.2. Функции создания изображений

Для работы с изображениями средствами библиотеки GDLib используются функции создания изображений, которые создают либо новое изображение, либо изображение на основе существующего. При этом само изображение сохраняется в оперативной памяти, а функции возвращают дескриптор, который используется другими функциями для доступа к изображению. После завершения работы с изображением оно либо сохраняется на жесткий диск, либо непосредственно выводится в окно браузера. В табл. 31.5 приводится список функций, относящихся к данной группе.

> ***ЗАМЕЧАНИЕ***
>
> Большинство функций создания изображения возвращает дескриптор открытого изображения. В случае неудачи возвращается пустая строка и выводится соответствующее предупреждение в окно браузера. Если вывод предупреждений нежелателен, его можно подавить, добавив перед функцией символ `@`.

Таблица 31.5. *Функции создания изображений*

Функция	Описание
`imagecreate($width, $height)`	Создает пустое изображение, ограниченное 256 цветами, размерами `$width` на `$height` пикселов. Для создания полноцветных изображений следует воспользоваться функцией `imagecreatetruecolor()`
`imagecreatetruecolor ($width, $height)`	Создает пустое полноцветное изображение черного цвета размерами `$width` на `$height` пикселов
`imagecreatefromjpeg ($filename)`	Создает изображение на основе JPEG-файла `$filename`. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
`imageinterlace($image [, $interlace])`	Для открытого JPEG-изображения (дескриптор `$image`) включается (`interlace` = 1) или выключается (`interlace` = 0) чересстрочное формирование изображения. Параметр `$interlace` указывает, сколько бит изображения должна вернуть функция
`imagecreatefromgif ($filename)`	Создает изображение на основе GIF-файла `$filename`. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
`imagecreatefrompng ($filename)`	Создает изображение на основе PNG-файла `$filename`. В качестве имени файла может использоваться сетевой путь, начинающийся с http://

***Таблица 31.5** (окончание)*

Функция	Описание
imagesavealpha($*image*, $*saveflag*)	Если $*saveflag* = 1, для открытого PNG-изображения $*image* создается полноценный альфа-канал; в противном случае, если $*saveflag* = 2, в качестве прозрачного может быть установлен только один цвет изображения
imagecreatefromgd ($*filename*)	Создает изображение на основе GD-файла $*filename*. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
imagecreatefromgd2 ($*filename*)	Создает изображение на основе GD2-файла $*filename*. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
imagecreatefromgd2part ($*filename*, $*x*, $*y*, $*width*, $*height*)	Создает изображение на основе файла $*filename*, при этом за основу берется не все изображение, а лишь прямоугольник, координаты левого верхней части которого определяются параметрами $x и $y, а ширина и высота соответственно параметрами $width и $height
imagecreatefromstring ($*str*)	Создает изображения из потока, передаваемого в строке $str
imagecreatefromwbmp ($*filename*)	Создает изображение на основе WBMP-файла $*filename*. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
imagecreatefromxbm ($*filename*)	Создает изображение на основе XBM-файла $*filename*. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
imagecreatefromxpm ($*filename*)	Создает изображение на основе XPM-файла $filename. В качестве имени файла может использоваться сетевой путь, начинающийся с http://
imagedestroy($*image*)	Принимает в качестве параметра $*image* дескриптор изображения и освобождает занятую им память

В листинге 31.7 демонстрируется создание изображения при помощи функции `imagecreatetruecolor()` с последующим уничтожением при помощи функции `imagedestroy()`.

Листинг 31.7. Создание и уничтожение изображения

```
<?php
  // Создаем пустое изображение размером 100x100 пикселов
  $image = imagecreatetruecolor(100, 100);
  if(!$image)
  {
    // Если изображение успешно создано, уничтожаем его
    imagedestroy($image);
  }
?>
```

В листинге 31.8 приводится скрипт, создающий изображение на основе уже существующего JPEG-файла.

> ***ЗАМЕЧАНИЕ***
>
> Важно отметить, что функция `imagedestroy()` уничтожает не файл на жестком диске PICT0073.JPG, а его образ в оперативной памяти.

Листинг 31.8. Создание изображения на основе существующего JPEG-файла

```
<?php
  // Создаем изображение на основе JPEG-файла
  $image = imagecreatefromjpeg("PICT0073.JPG");
  if(!$image)
  {
    // Если изображение успешно создано, уничтожаем его
    imagedestroy($image);
  }
?>
```

31.3. Функции сохранения и вывода изображений

Данная группа функций позволяет сохранять открытые файлы на жесткий диск или выводить их в окно браузера. В табл. 31.6 приводится список функций, относящихся к данной группе.

> ***ЗАМЕЧАНИЕ***
>
> Функции данной группы в качестве первого параметра принимают дескриптор открытого изображения `$image`. Если больше никаких параметров не передается, изображение выводится непосредственно в окно браузера. Если же в качестве второго параметра `$filename` указывается путь к файлу, изображение сохраняется на жестком диске. В случае успеха функции возвращают `TRUE`, в случае неудачи — `FALSE`.

Таблица 31.6. *Функции сохранения и вывода изображений*

Функция	Описание
`imagejpeg($image [, $filename [, $quality]])`	Возвращает или сохраняет в файл `$filename` изображение в формате JPEG со степенью сжатия `$quality`. Параметр `$quality` может принимать значения от 0 до 100 (по умолчанию 75), что соответствует максимальному и минимальному сжатию. При максимальной степени сжатия качество изображения будет минимально
`imagegif($image [, $filename])`	Возвращает или сохраняет в файл `$filename` изображение в формате GIF

***Таблица 31.6** (окончание)*

Функция	Описание
imagepng($*image* [, $*filename*])	Возвращает или сохраняет в файл $*filename* изображение в формате PNG
imagegd($*image* [, $*filename*])	Возвращает или сохраняет в файл $*filename* изображение в формате GD
imagegd2($*image* [, $*filename*])	Возвращает или сохраняет в файл $*filename* изображение в формате GD2
imagexbm($*image*, $*filename* [, $*foreground*])	Возвращает или сохраняет в файл $*filename* изображение в формате XBM. Аргумент $*foreground* содержит идентификатор цвета переднего плана (по умолчанию — черный)
imagewbmp($*image* [, $*filename* [, $*foreground*]])	Возвращает или сохраняет в файл $*filename* изображение в формате WBMP. Аргумент $*foreground* содержит идентификатор цвета переднего плана (по умолчанию — черный)
image2wbmp($*image* [, $*filename*])	Возвращает или сохраняет в файл $filename изображение в формате WBMP
jpeg2wbmp($*jpegname*, $*wbmpname*, $*height*, $*width*, $*threshold*)	Конвертирует JPEG-файл $*jpegname* в файл WBMP-формата $*wbmpname*. Параметры $*height* и $*width* определяют высоту и ширину конечного WBMP-изображения
png2wbmp($*pngname*, $*wbmpname*, $*height*, $*width*, $*threshold*)	Конвертирует PNG-файл $*pngname* в файл WBMP-формата $*wbmpname*. Параметры $height и $width определяют высоту и ширину конечного WBMP-изображения

При выводе изображения в окно браузера при помощи функций из табл. 31.6 следует помнить, что перед отправкой бинарных данных необходимо сообщить браузеру MIME-тип изображения. В листинге 31.9 демонстрируется вывод в окно браузера JPEG-изображения.

> ***Замечание***
>
> Для создания файлов в формате progressive JPEG следует включить чересстрочное создание изображений функцией imageinterlace().

Листинг 31.9. Вывод JPEG-изображения

```
<?php
  // Создаем изображение на основе JPEG-файла
  $image = imagecreatefromjpeg("PICT0073.JPG");
  if($image)
  {
    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/jpeg");
    // Выводим изображение в браузер
```

```
    imagejpeg($image);
    // Уничтожаем изображение в памяти
    imagedestroy($image);
  }
?>
```

Следует учитывать, что документ не может содержать сразу несколько изображений. Попытка отправить HTTP-заголовки уже после вывода тела документа приведет к сообщению "Warning: Cannot modify header information - headers already sent by" ("Предупреждение: HTTP-заголовки не могут быть изменены, т. к. они уже отправлены"). Поэтому при использовании библиотеки GDLib скрипт не должен выводить ничего, кроме изображений. Если на странице необходимо вывести несколько динамических изображений, следует обратиться к HTML-тегу `<IMG>`.

31.4. Функции преобразования изображений

Данная группа функций осуществляет изменение размера изображений, поворот и т. п. В табл. 31.7 приводится список функций, относящихся к данной группе.

> ***ЗАМЕЧАНИЕ***
>
> Часть функций данной группы принимает два дескриптора отрытых изображения: источника `$src_im` и конечного изображения `$dst_im`.

Таблица 31.7. *Функции преобразования изображений*

Функция	Описание
`imagecopy($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $src_w, $src_h)`	Копирует область изображения `$src_im` на изображение `$dst_im`. Левый верхний угол копируемой области находится в точке с координатами `$src_x`, `$src_y`. Параметры `$src_w` и `$src_h` определяют ширину и высоту копируемой области соответственно. Скопированная область помещается на изображение `$dst_im` в точку с координатами `$dst_x`, `$dst_y`
`imagecopymerge($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $src_w, $src_h, $pct)`	Копирует область изображения `$src_im` и объединяет ее с изображением `$dst_im`. При этом можно задать степень прозрачности накладываемого изображения при помощи параметра `$pct`, который принимает значения в диапазоне от 0 до 100. При значении, равном 0, накладываемое изображение становится полностью прозрачным. При `$pct`, равном 100, функция работает аналогично функции `imagecopy()`. Остальные параметры функции идентичны параметрам функции `imagecopy()`
`imagecopymergegray($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $src_w, $src_h, $pct)`	Функция аналогична функции `imagecopymerge()`, однако при совмещении областей сохраняются оттенки изображения-источника `$src_im`. Сохранение оттенков происходит вследствие удаления цветовой информации из совмещаемых пикселов изображения-назначения `$dst_im` до операции совмещения

***Таблица 31.7** (окончание)*

Функция	Описание
`imagecopyresampled($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $dst_w, $dst_h, $src_w, $src_h)`	Копирует прямоугольную область изображения-источника `$src_im` на целевое изображение `$src_im`. При копировании пикселы интерполируются, чтобы сохранить максимальную четкость и яркость. Если размеры исходной области и области назначения различны, то происходит соответствующее сжатие или растяжение копируемой области. Параметры `$src_x` и `$src_y` определяют правый верхний угол прямоугольника на изображении-источнике. Параметры `$src_w` и `$src_h` соответственно ширину и высоту этого прямоугольника. Аналогичные параметры с префиксом `$dst` определяют параметры прямоугольника на конечном изображении
`imagecopyresized($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $dst_w, $dst_h, $src_w, $src_h)`	Функция аналогична функции `imagecopyresampled()`, но при копировании не происходит интерполяции копируемых пикселов для сохранения четкости и яркости
`imagerotate($src_im, $angle, $bgd_color)`	Осуществляет поворот изображения `$src_im` на угол `$angle`, заданный в градусах. В аргументе `$bgd_color` передается идентификатор фонового цвета для заливки пустых областей, которые будут получены в процессе поворота изображения
`imagefilter($src_im, $filtertype [, $args])`	Применяет фильтр к изображению `$src_im`

31.4.1. Создание уменьшенной копии изображения

Оформим процедуру уменьшения размера изображения в виде отдельной функции `resizeimg()` (листинг 31.10). Функция будет принимать четыре параметра:

```
resizeimg($filename, $smallimage, $w, $h)
```

Параметр `$filename` задает путь к исходному файлу, который будет подвергаться сжатию; `$smallimage` — путь к уменьшенной копии изображения; `$w` и `$h` — ширину и высоту уменьшенного изображения.

Листинг 31.10. Функция изменения размера изображения

```
<?php
  function resizeimg($filename, $smallimage, $w, $h)
  {
    // 1. Коррекция параметров $w и $h
    // Определим коэффициент сжатия изображения
    $ratio = $w / $h;
```

```
    // Получим размеры исходного изображения
    list($width, $height) = getimagesize($filename);
    // Если размеры меньше, то масштабирования не нужно
    if (($width < $w) && ($height < $h))
    {
      copy($filename, $smallimage);
      return true;
    }
    // Получаем коэффициент сжатия исходного изображения
    $src_ratio = $width/ $height;

    // Вычисляем размеры уменьшенной копии, чтобы при масштабировании
    // сохранились пропорции исходного изображения
    if ($ratio < $src_ratio) $h = $w/$src_ratio;
    else $w = $h*$src_ratio;

    // 2. Создание уменьшенной копии изображения
    // Создаем пустое изображение размером $w x $h пикселов
    $dest_img = imagecreatetruecolor($w, $h);
    // Открываем файл, который будет подвергаться сжатию
    $src_img = imagecreatefromjpeg($filename);

    // Масштабируем изображение
    imagecopyresampled($dest_img,
                       $src_img,
                       0,
                       0,
                       0,
                       0,
                       $w,
                       $h,
                       $width,
                       $height);
    // Сохраняем уменьшенную копию в файл
    imagejpeg($dest_img, $smallimage);
    // Чистим память от созданных изображений
    imagedestroy($dest_img);
    imagedestroy($src_img);
    return true;
  }
?>
```

Первая часть функции посвящена коррекции входных параметров `$w` и `$h`, задающих ширину и высоту уменьшенного изображения. Это необходимо для того, чтобы изображение не было искажено преобразованием. Вторая часть функции предназначена для создания уменьшенной копии `$smallimage` из исходного изображения `$filename`. Уменьшенное изображение создается при помощи функции `imagecreatetruecolor()`, которая возвращает дескриптор `$dest_img`. Дескриптор исходного изображения `$src_img` открывается при помощи функции `imagecreatefromjpeg()`. После этого исходное изображение переносится на уменьшенную копию при помощи функции `imagecopyresampled()`.

31.4.2. Поворот изображения

Для поворота изображения удобно воспользоваться функцией `imagerotate()` (листинг 31.11).

> ***Замечание***
>
> Третий параметр функции `imagerotate()` требует задания цвета. Функции для формирования цвета подробно обсуждаются в *разд. 28.5*. Можно было передать в качестве этого параметра значение 0, однако в этом случае для фонового цвета был бы использован цвет по умолчанию — черный, в то время как нам необходим белый.

Листинг 31.11. Поворот изображения при помощи функции `imagerotate()`

```
<?php
  // Создаем изображение на основе JPEG-файла
  $image = imagecreatefromjpeg("image.jpg");
  if($image)
  {
    // Задаем белый цвет
    $white = imagecolorallocate($image, 255, 255, 255);
    // Осуществляем поворот изображения на 45 градусов
    $rotate = imagerotate($image, 45, $white);
    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/jpeg");
    // Выводим изображение в браузер
    imagejpeg($rotate);
    // Уничтожаем изображения в памяти
    imagedestroy($image);
    imagedestroy($rotate);
  }
?>
```

Результатом работы скрипта из листинга 31.11 является новое изображение, сформированное поворотом старого изображения на 45° (рис. 31.1).



Рис. 31.1. Поворот изображения на 45°

31.5. Функции для работы с цветом

Данная группа функций позволяет работать с цветами: получать и уничтожать дескриптор цвета, используемый для заливки, определять цвета отдельных пикселов и т. п. В табл. 31.8 приводится список функций, относящихся к данной группе.

> ***ЗАМЕЧАНИЕ***
>
> Как и большинство функций библиотеки GDLib, функции данной группы принимают в качестве первого параметра дескриптор открытого изображения `$image`.

При работе с цветом используется RGB-представление: все цвета получаются смешением красного (`$red`), зеленого (`$green`) и синего (`$blue`) цветов. Каждый из параметров может принимать значения от 0 (полное отсутствие цвета) до 255 (100% присутствие цвета), обеспечивая тем самым возможность создания 16 777 216 оттенков. Если имя функции заканчивается суффиксом `alpha`, функция поддерживает работу с прозрачностью, которая задается параметром `$alpha`, способным принимать значения от 0 (полная непрозрачность) до 127 (полная прозрачность).

Таблица 31.8. *Функции для работы с цветом*

Функция	Описание
`imageistruecolor($image)`	Определяет, является ли изображение `$image` полноцветным
`imagetruecolortopalette($image, $dither, $ncolors)`	Конвертирует полноцветное изображение в изображение с ограниченной цветовой палитрой. Если параметр `$dither` принимает значение `true`, осуществляется размытие (смешение двух цветов для получения третьего), в противном случае, если параметр принимает значение `false`, размытия не происходит. Параметр `$ncolors` определяет количество цветов в конечном изображении

***Таблица 31.8** (продолжение)*

Функция	Описание
`imagecolorallocate($image, $red, $green, $blue)`	Возвращает дескриптор цвета, представленного RGB-компонентами: `$red` (красный), `$green` (зеленый) и `$blue` (синий)
`imagecolorallocatealpha($image, $red, $green, $blue, $alpha)`	Функция аналогична `imagecolorallocate()`, однако позволяет в четвертом параметре `$alpha` задать прозрачность
`imagecolordeallocate($image, $color)`	Уничтожает дескриптор цвета, выделенный ранее функцией `imagecolorallocate()` или `imagecolorallocatealpha()`
`imagecolorat($image, $x, $y)`	Возвращает дескриптор цвета для пиксела изображения в точке с координатами `$x` и `$y`
`imagecolorsforindex($image, $index)`	Для дескриптора цвета `$index` возвращает ассоциативный массив с ключами `'red'`, `'green'`, `'blue'`, соответствующими долям красного, зеленого и синего цветов
`imagecolorresolve($image, $red, $green, $blue)`	Ищет на изображении RBG-цвет, заданный компонентами `$red` (красный), `$green` (зеленый) и `$blue` (синий), и в случае успеха возвращает дескриптор цвета. Если цвет не найден, возвращается дескриптор ближайшего цвета
`imagecolorresolvealpha($image, $red, $green, $blue, $alpha)`	Функция аналогична `imagecolorresolve()`, однако помимо цветов учитывается степень прозрачности `$alpha`
`imagecolorexact($image, $red, $green, $blue)`	Ищет на изображении RGB-цвет, заданный компонентами `$red` (красный), `$green` (зеленый) и `$blue` (синий), и в случае успеха возвращает дескриптор цвета. Если цвет не найден, возвращается –1
`imagecolorexactalpha($image, $red, $green, $blue, $alpha)`	Функция аналогична `imagecolorexact()`, однако помимо цветов учитывается степень прозрачности `$alpha`
`imagecolorclosest($image, $red, $green, $blue)`	Ищет на изображении RGB-цвет, заданный компонентами `$red` (красный), `$green` (зеленый) и `$blue` (синий), и возвращает дескриптор наиболее близкого цвета
`imagecolorclosestalpha($image, $red, $green, $blue, $alpha)`	Функция аналогична `imagecolorclosest()`, однако помимо цветов учитывается степень прозрачности `$alpha`
`imagecolorclosesthwb($image, $red, $green, $blue)`	Возвращает дескриптор градации серого для цвета, заданного компонентами `$red` (красный), `$green` (зеленый) и `$blue` (синий)

***Таблица 31.8** (окончание)*

Функция	Описание
`imagecolorset($image, $index, $red, $green, $blue)`	Функция заменяет на изображении цвет с дескриптором `$index` новым цветом, заданным компонентами `$red` (красный), `$green` (зеленый) и `$blue` (синий)
`imagecolortransparent($image [, $color])`	Области изображения, прорисованные цветом `$color`, после применения этой функции станут прозрачными
`imagecolorstotal($image)`	Возвращает количество цветов в цветовой палитре изображения, если изображение создано функций `imagecreate()`, и 0, если изображение полноцветное
`imagecolormatch($image1, $image2)`	Сравнивает два изображения: полноцветное изображение `$image1` и изображение с ограниченной палитрой цветов `$image2`. Если изображения признаются похожими, то возвращается `true`, в противном случае — `false`
`imagepalettecopy($destination, $source)`	Копирует цветовую палитру из изображения-источника `$source` в изображение-приемник `$destination`
`imagegammacorrect($image, $inputgamma, $outputgamma)`	Осуществляет гамма-коррекцию изображения

31.5.1. Заливка изображения цветом

Создадим скрипт img.php (листинг 31.12), который принимает три GET-параметра: `red`, `green` и `blue`, соответствующих красному, зеленому и синему цветам в RGB-схеме. В качестве результата скрипт будет выводить квадрат размером 100×100 пикселов заданного цвета.

> ***Замечание***
>
> В работе скрипта из листинга 31.12 для заливки изображения используется функция `imagefill()`, которая более подробно рассматривается в *разд. 28.6*.

Листинг 31.12. Вывод цветного квадрата

```
<?php
  // Создаем пустое изображение размером 100x100 пикселов
  $image = imagecreatetruecolor(100, 100);
  if($image)
  {
    // Осуществляем проверку корректности компонентов RBG-цвета,
```

```
    // переданных через GET-параметры
    if($_GET['red'] < 0)     $_GET['red'] = 0;
    if($_GET['red'] > 255)   $_GET['red'] = 255;
    if($_GET['green'] < 0)   $_GET['green'] = 0;
    if($_GET['green'] > 255) $_GET['green'] = 255;
    if($_GET['blue'] < 0)    $_GET['blue'] = 0;
    if($_GET['blue'] > 255)  $_GET['blue'] = 255;
    // Получаем дескриптор цвета
    $color = imagecolorallocate($image,
                                $_GET['red'],
                                $_GET['green'],
                                $_GET['blue']);
    // Заливаем выбранным цветом изображение
    imagefill($image, 0, 0, $color);

    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/gif");
    // Выводим изображение в браузер
    imagegif($image);
    // Уничтожаем изображение в памяти
    imagedestroy($image);
  }
?>
```

На рис. 31.2 демонстрируется работа скрипта из листинга 31.12.



Рис. 31.2. Вывод цветного квадрата в окно браузера

В листинге 31.13 демонстрируется вывод трех цветных квадратов: красного, зеленого и синего.

Листинг 31.13. Вывод трех цветных квадратов на одной странице

```
<img src=img.php?red=127&green=0&blue=0>
<img src=img.php?red=0&green=127&blue=0>
<img src=img.php?red=0&green=0&blue=127>
```

Результат интерпретации HTML-кода из листинга 31.3 демонстрируется на рис. 31.3.

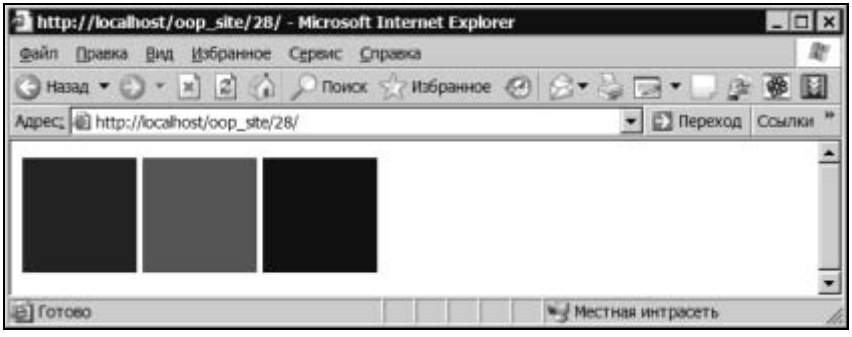


Рис. 31.3. Вывод трех цветных квадратов

31.5.2. Получение цвета заданного пиксела

Для получения дескриптора пиксела в заданной точке изображения удобно воспользоваться функцией `imagecolorat()`, далее компоненты RGB-схемы можно получить при помощи функции `imagecolorsforindex()` (листинг 31.14).

Листинг 31.14. Получение цвета заданного пиксела

```
<?php
  // Создаем изображение на основе существующего
  $image = imagecreatefromjpeg("PICT0073.JPG");
  if($image)
  {
    // Получаем дескриптор цвета для точки
    // с координатами x = 100, y = 100
    $color = imagecolorat($image, 100, 100);

    // Получаем компоненты цвета в виде массива
    $arr = imagecolorsforindex($image, $color);

    echo "<pre>";
    print_r($arr);
```

```
    echo "</pre>";

    // Уничтожаем изображение в памяти
    imagedestroy($image);
  }
?>
```

Результат работы скрипта из листинга 31.14 может выглядеть следующим образом:

```
Array
(
    [red] => 99
    [green] => 111
    [blue] => 39
    [alpha] => 0
)
```

31.5.3. Уменьшение количества цветов в изображении

Часто для оптимизации изображения или для достижения плакатного эффекта количество цветов в гамме сокращают. В библиотеке GDLib для этого предназначена функция `imagetruecolortopalette()`. Разработаем скрипт img.php, который будет принимать единственный GET-параметр `color`, равный количеству цветов в конечном изображении (листинг 31.15).

Листинг 31.15. Уменьшение количества цветов в изображении

```
<?php
  // Создаем изображение на основе существующего
  $image = imagecreatefromjpeg("image.jpg");
  if($image)
  {
    // Проверяем корректность параметра color
    if($_GET['color'] < 1) $_GET['color'] = 1;
    if($_GET['color'] > 256) $_GET['color'] = 256;

    // Уменьшаем количество цветов
    imagetruecolortopalette($image, true, $_GET['color']);

    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/gif");
    // Выводим изображение в браузер
    imagegif($image);
    // Уничтожаем изображение в памяти
```

```
    imagedestroy($image);
  }
?>
```

Для демонстрации работы скрипта из листинга 31.15 разработаем скрипт index.php, который наряду с исходным изображением выводит изображение, содержащее 20, 10 и 2 цвета (листинг 31.16).

Листинг 31.16. Вывод изображений с разным количеством цветов

```
<img src=image.jpg>
<img src=img.php?color=20>
<img src=img.php?color=10>
<img src=img.php?color=2>
```

Результат интерпретации HTML-кода из листинга 31.16 представлен на рис. 31.4.



Рис. 31.4. Вывод изображений с разным количеством цветов

31.5.4. Замена одного цвета другим

Графические пакеты часто предоставляют пользователю возможность замены одного цвета другим. Зачастую перед этой операцией удобно сократить количество цветов в изображении при помощи функции `imagetruecolortopalette()`, которая делает этот эффект более заметным. В листинге 31.17 приводится скрипт, который уменьшает количество цветов в фотографии до 10, затем выбирает цвет пиксела, отстоящего от левого верхнего угла, на 10 пикселов по оси абсцисс *x* и по оси ординат *y* и заменяет этот цвет на черный.

Листинг 31.17. Замена цвета

```
<?php
  // Создаем изображение на основе существующего
  $image = imagecreatefromjpeg("image.jpg");
```

```
  if($image)
  {
    // Уменьшаем количество цветов
    imagetruecolortopalette($image, true, 10);

    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/gif");
    // Выводим изображение в браузер
    imagegif($image);
    // Уничтожаем изображение в памяти
    imagedestroy($image);
  }
?>
```

На рис 31.5 демонстрируются три изображения: оригинал; изображение, количество цветов в котором уменьшено до 10; и изображение, сформированное скриптом из листинга 31.16.



Рис. 31.5. Замена цвета на изображении

31.6. Функции рисования

Данная группа функций позволяет рисовать точки, линии, геометрические фигуры. В табл. 31.9 представлен список функций данной группы.

ЗАМЕЧАНИЕ

Как и большинство функций библиотеки GDLib, функции данной группы принимают в качестве первого параметра дескриптор открытого изображения `$image`. Последний параметр `$color` всех функций данного раздела представляет собой дескриптор цвета, работа с которым подробно обсуждается в *разд. 31.5*.

Таблица 31.9. *Функции рисования*

Функция	Результат
imagesetpixel($*image*, $*x*, $*y*, $*color*)	Пиксел в точке с координатами $*x* и $*y*
imageline($*image*, $*x1*, $*y1*, $*x2*, $*y2*, $*color*)	Линия от точки с координатами $*x1* и $*y1* до точки с координатами $*x2* и $*y2*
imagedashedline($*image*, $*x1*, $*y1*, $*x2*, $*y2*, $*color*)	Штрихпунктирная линия от точки с координатами $*x1* и $*y1* до точки с координатами $*x2* и $*y2*
imagerectangle($*image*, $*x1*, $*y1*, $*x2*, $*y2*, $*color*)	Прямоугольник с левым верхним углом в точке с координатами $*x1* и $*y1* и правым нижним углом в точке с координатами $*x2* и $*y2*
imagepolygon($*image*, $*points*, $*num_points*, $*color*)	Многоугольник с координатами (*X*, *Y*) вершин, представленными в массиве точек $*points*
imagearc($*image*, $*cx*, $*cy*, $*w*, $*h*, $*s*, $*e*, $*color*)	Дуга эллипса. Параметры $*cx* и $*cy* задают координаты эллипса по оси абсцисс и ординат соответственно. Параметры $*w* и $*h* задают ширину и высоту эллипса, а $*s* и $*e* — начальный и конечный углы сектора в градусах (нулевая позиция соответствует 3 часам)
imageellipse($*image*, $*cx*, $*cy*, $*w*, $*h*, $*color*)	Эллипс. Центр эллипса находится в точке с координатами $*cx* и $*cy*. Параметры $*w* и $*h* задают ширину и высоту эллипса, а $*s* и $*e* — начальный и конечный углы сектора в градусах (нулевая позиция соответствует 3 часам)
imagefilledrectangle($*image*, $*x1*, $*y1*, $*x2*, $*y2*, $*color*)	Закрашенный прямоугольник с левым верхним углом в точке с координатами $*x1* и $*y1* и правым нижним углом в точке с координатами $*x2* и $*y2*
imagefilledpolygon($*image*, $*points*, $*num_points*, $*color*)	Закрашенный многоугольник, с количеством вершин $*num_points* с координатами (*X*, *Y*), содержащимися в массиве точек $*points*
imagefilledarc($*image*, $*cx*, $*cy*, $*w*, $*h*, $*s*, $*e*, $*color*, $*style*)	Сектор эллипса с заливкой. Параметры $*cx* и $*cy* задают координаты эллипса по оси абсцисс и ординат соответственно, $*w* и $*h* — ширину и высоту эллипса, $*s* и $*e* — начальный и конечный углы сектора в градусах (нулевая позиция соответствует 3 часам). Параметр $*style* определяет стиль формирования сектора и может принимать следующие константы: • IMG_ARC_PIE — соединяет начальную и конечную точки на окружности кривой линией, повторяющей окружность; • IMG_ARC_CHORD — соединяет начальную и конечную точки на окружности прямой линией;

***Таблица 31.9** (окончание)*

Функция	Результат
	• IMG_ARC_NOFILL — рисуемый сектор не закрашивается; • IMG_ARC_EDGED — начальная и конечная точки соединяются с центром окружности. Константы могут объединяться при помощи побитового ИЛИ, однако IMG_ARC_PIE и IMG_ARC_CHORD являются взаимоисключающими и не могут применяться совместно
imagefilledellipse($*image*, $*cx*, $*cy*, $*w*, $*h*, $*color*)	Закрашенный эллипс с центром в точке с координатами $*cx* и $*cy* шириной $*w* и высотой $*h*
imagefill($*image*, $*x*, $*y*, $*color*)	Заливка цветом $*color* области, ограниченной пикселами, цвет которых не совпадает с цветом точки с координатами $*x* и $*y*
imagefilltoborder($*image*, $*x*, $*y*, $*border*, $*color*)	Заливка, начиная с точки с координатами $*x* и $*y*, цветом $*color* области рисунка, включающей пикселы, чей цвет отличен от цвета, заданного дескриптором $*border*. $*border* — задает цвет контура, который необходимо залить цветом $*color*

31.6.1. Кривая Безье

Кривая Безье — это сглаженная кривая, построенная по координатам (X, Y) четырех точек при помощи системы параметрических уравнений. Система уравнений выглядит следующим образом:

$$X(t) = (1-t)^3X_0 + 3t(1-t)^2X_1 + 3t^2(1-t)X_2 + t^3X_3,$$
$$Y(t) = (1-t)^3Y_0 + 3t(1-t)^2Y_1 + 3t^2(1-t)Y_2 + t^3Y_3.$$

Параметр t принимает произвольное значение от 0 до 1, что позволяет получить координаты любой точки (X, Y) сглаженной кривой.

Создадим небольшое Web-приложение, запрашивающее у пользователя четыре точки с координатами X_1, Y_1, X_2, Y_2, X_3, Y_3, X_4, Y_4 и отображающее график сглаженной кривой с количеством точек, равным 1000 (листинг 31.18).

Листинг 31.18. Построение сглаженной кривой

```
<?php
  if(!empty($_POST))
  {
    $x1 = $_POST['x1'];
    $x2 = $_POST['x2'];
    $x3 = $_POST['x3'];
    $x4 = $_POST['x4'];
```

```
$y1 = $_POST['y1'];
$y2 = $_POST['y2'];
$y3 = $_POST['y3'];
$y4 = $_POST['y4'];
if(!empty($x1) && !empty($y1) &&
   !empty($x2) && !empty($y2) &&
   !empty($x3) && !empty($y3) &&
   !empty($x4) && !empty($y4))
for($t = 0.0; $t <= 1.0; $t += 0.001)
{
  $x[] = (1 - $t) * (1 - $t) * (1 - $t) * $x1 +
      3 * $t * (1 - $t) * (1 - $t) * $x2 +
      3 * $t * $t * (1 - $t) * $x3 +
      $t * $t * $t * $x4;
  $y[] = (1 - $t) * (1 - $t) * (1 - $t) * $y1 +
      3 * $t * (1 - $t) * (1 - $t) * $y2 +
      3 * $t * $t * (1 - $t) * $y3 +
      $t * $t * $t * $y4;
}
// Определяем максимальные и минимальные значения по осям x и y
$max_x = $min_x = $x[0];
$max_y = $min_y = $y[0];
for($i = 0; $i < count($x); $i++)
{
  if($max_x < $x[$i]) $max_x = $x[$i];
  if($min_x > $x[$i]) $min_x = $x[$i];
  if($max_y < $y[$i]) $max_y = $y[$i];
  if($min_y > $y[$i]) $min_y = $y[$i];
}

// Ширина изображения
$width = 100;
// Высота изображения
$height = 100;
// Создаем пустое изображение с 10-пиксельными отступами по краям
$img = imagecreatetruecolor($width + 20, $height + 20);
// Заливаем фон белым цветом
$white = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $white);
// Черный цвет для линии
$black = imagecolorallocate($img, 0, 0, 0);
```

```
    // В цикле отрисовываем линию
    for($i = 0; $i < count($x) - 1; $i++)
    {
      // Масштабируем координаты
      $x1 = 10 + intval(($x[$i] - $min_x)*$width/($max_x - $min_x));
      $y1 = 10 + intval(($max_y - $y[$i])*$height/($max_y - $min_y));
      $x2 = 10 + intval(($x[$i + 1] - $min_x)*$width/($max_x - $min_x));
      $y2 = 10 + intval(($max_y - $y[$i + 1])*$height/($max_y - $min_y));
      // Рисуем линию
      imageline($img, $x1, $y1, $x2, $y2, $black);
    }
    // Вычисляем координаты исходных точек
    $x1 = 10 + intval(($_POST['x1'] - $min_x)*$width/($max_x - $min_x));
    $y1 = 10 + intval(($max_y - $_POST['y1'])*$height/($max_y - $min_y));
    $x2 = 10 + intval(($_POST['x2'] - $min_x)*$width/($max_x - $min_x));
    $y2 = 10 + intval(($max_y - $_POST['y2'])*$height/($max_y - $min_y));
    $x3 = 10 + intval(($_POST['x3'] - $min_x)*$width/($max_x - $min_x));
    $y3 = 10 + intval(($max_y - $_POST['y3'])*$height/($max_y - $min_y));
    $x4 = 10 + intval(($_POST['x4'] - $min_x)*$width/($max_x - $min_x));
    $y4 = 10 + intval(($max_y - $_POST['y4'])*$height/($max_y - $min_y));
    // Отрисовываем точки
    imagefilledellipse($img, $x1, $y1, 7, 7, $black);
    imagefilledellipse($img, $x2, $y2, 7, 7, $black);
    imagefilledellipse($img, $x3, $y3, 7, 7, $black);
    imagefilledellipse($img, $x4, $y4, 7, 7, $black);

    // Выводим изображение в браузер
    header('Content-type: image/jpeg');
    imagejpeg($img);
    exit();
  }
?>
<table>
<form method='post'>
  <tr align='center'>
    <td>Координата (x1, y1)</td>
    <td><input type=text name=x1 size=3 value="<?= $_POST['x1'] ?>"></td>
    <td><input type=text name=y1 size=3 value="<?= $_POST['y1'] ?>"></td>
  </tr>
  <tr align=center>
    <td>Координата (x2, y2)</td>
```

```
    <td><input type=text name=x2 size=3 value="<?= $_POST['x2'] ?>"></td>
    <td><input type=text name=y2 size=3 value="<?= $_POST['y2'] ?>"></td>
  </tr>
  <tr align=center>
    <td>Координата (x3, y3)</td>
    <td><input type=text name=x3 size=3 value="<?= $_POST['x3'] ?>"></td>
    <td><input type=text name=y3 size=3 value="<?= $_POST['y3'] ?>"></td>
  </tr>
  <tr align=center>
    <td>Координата (x4, y4)</td>
    <td><input type=text name=x4 size=3 value="<?= $_POST['x4'] ?>"></td>
    <td><input type=text name=y4 size=3 value="<?= $_POST['y4'] ?>"></td>
  </tr>
  <tr>
    <td></td>
    <td cols=2><input type=submit value="Сгладить"></td>
  </tr>
</form>
</table>
```

При первой загрузке скрипта из листинга 31.18 выводится HTML-форма, позволяющая задать координаты четырех точек (рис. 31.6).

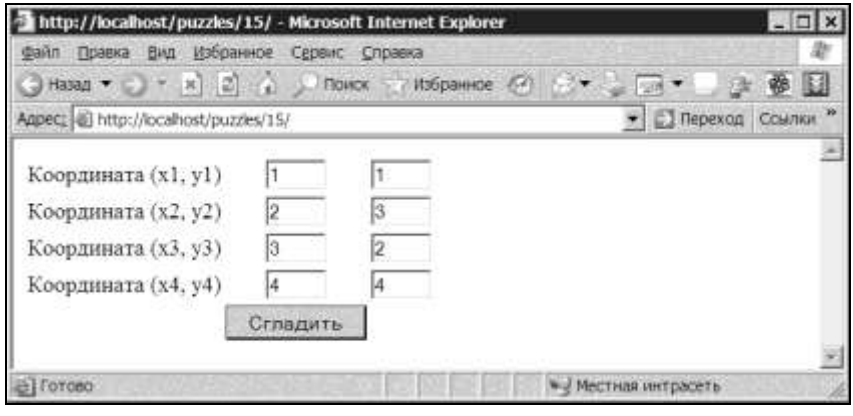


Рис. 31.6. Ввод координат четырех опорных точек

После того как пользователь заполняет поля и нажимает кнопку **Сгладить**, данные отправляются обработчику, который выполняет сглаживание по приведенным ранее формулам, отрисовывает точки при помощи функции `imagefilledellipse()` и формирует линию при помощи функции `imageline()`. Результат работы приложения с параметрами, приведенными на рис. 31.6, представлен на рис. 31.7.



Рис. 31.7. Кривая Безье, проведенная через четыре точки

31.6.2. Построение гистограммы средствами GDLib

Пусть значения столбцов задаются в процентах (от 0 до 100) в массиве `$rows` (листинг 31.19).

Листинг 31.19. Процентное задание столбцов в массиве `$rows`

```
<?php
  // Значение столбцов от 0 до 100
  $rows = array(80, 75, 53, 32, 20);
?>
```

В листинге 31.20 приводится пример построения гистограммы средствами библиотеки GDLib.

Листинг 31.20. Построение гистограммы средствами библиотеки GDLib

```
<?php
  // Значение столбцов от 0 до 100
  $rows = array(80, 75, 53, 32, 20);

  // Ширина изображения
  $width = 200;
  // Высота изображения
  $height = 200;
  // Ширина одного столбца
  $row_width = 30;
  // Ширина интервала между столбцами
  $row_interval = 5;

  // Создаем пустое изображение
```

```
  $img = imagecreatetruecolor($width, $height);

  // Заливаем изображение белым цветом
  $white = imagecolorallocate($img, 255, 255, 255);
  imagefill($img, 0, 0, $white);

  for($i = 0, $y1 = $height, $x1 = 0; $i < count($rows); $i++)
  {
    // Формируем случайный цвет для каждого из столбцов
    $color = imagecolorallocate($img,
                    rand(0, 255), rand(0, 255), rand(0, 255));

    // Нормирование высоты столбца
    $y2 = $y1 - $rows[$i]*$height/100;
    // Определение второй координаты столбца
    $x2 = $x1 + $row_width;

    // Отрисовываем столбец
    imagefilledrectangle($img, $x1, $y1, $x2, $y2, $color);

    // Между столбцами создаем интервал в $row_interval пикселов
    $x1 = $x2 + $row_interval;
  }

  // Выводим изображение в браузер, в формате GIF
  header ("Content-type: image/gif");
  imagegif($img);
?>
```

Как видно из листинга 31.20, скрипт использует значения элементов массива `$rows` как процентные величины для формирования в цикле `for` гистограммы. На каждой итерации цикла формируется случайный цвет, который используется для отрисовки очередного столбца гистограммы. Столбец формируется при помощи функции `imagefilledrectangle()`, которая рисует заполненный прямоугольник. На рис. 31.8 приводится результат работы скрипта из листинга 31.20.



Рис. 31.8. Гистограмма

31.6.3. Построение круговой диаграммы средствами GDLib

Пусть доли секторов заданы массивом `$row` из листинга 31.18. При построении круговой диаграммы следует помнить, что в круге 360°, и значения столбцов массива `$rows` следует нормировать таким образом, чтобы их сумма равнялась 360°. В листинге 31.21 приводится возможный вариант решения задачи создания круговой диаграммы.

Листинг 31.21. Построение круговой диаграммы

```
<?php
  // Значения столбцов от 0 до 100
  $rows = array(80, 75, 53, 32, 20);
  // Нормируем значения массива $rows таким образом,
  // чтобы их сумма составляла 360 градусов
  $summ = array_sum($rows);
  for($i = 0; $i < count($rows); $i++)
  {
    $rows[$i] = intval($rows[$i]*360/$summ);
  }

  // Создаем пустое изображение, размером 201x201 пикселов
  $img =  imagecreatetruecolor(201, 201);

  // Определение белого цвета на изображении
  $white = imagecolorallocate($img, 255, 255, 255);
  imagefill($img, 0, 0, $white);

  // Переменные $cy и $cy определяют центр круговой диаграммы
  $cx = $cy = 100;
  // Переменные $w и $h определяют ширину и высоту диаграммы
  $w = $h = 200;

  $start = 0;
  foreach ($rows as $value)
  {
    // Формируем случайный цвет для каждого сектора
    $color = imagecolorallocate($img,
                                rand(0, 255),
                                rand(0, 255),
                                rand(0, 255));
    // Определяем конечный угол сектора
    $angle_sector = $start + $value;
```

```
    // Отрисовываем сектор
    imagefilledarc($img, $cx, $cy, $w, $h, $start, $angle_sector,
                   $color, "IMG_ARC_PIE || IMG_ARC_EDGED");
    // Увеличиваем значение начального угла сектора
    $start += $value;
  }
  // Вывод изображения в окно браузера
  header ("Content-type: image/gif");
  imagegif($img);
?>
```

Результат работы скрипта из листинга 31.21 представлен на рис. 31.9.



Рис. 31.9. Круговая диаграмма

31.7. Функции настройки рисования

Данная группа функций позволяет настроить параметры рисования: кисти, толщину линии и т. п. В табл. 31.10 представлен список функций данной группы.

> ***ЗАМЕЧАНИЕ***
>
> Как и большинство функций библиотеки GDLib, функции данной группы принимают в качестве первого параметра дескриптор открытого изображения `$image`.

Таблица 31.10. *Функции настройки рисования*

Функция	Описание
`imagealphablending($image, $blendmode)`	Задает режим учета прозрачности при накладывании одного изображения на другое. Если параметр `$blendmode` принимает значение `false`, накладываемое изображение полностью заменяет собой лежащее под ним изображение. Если параметр `$blendmode` принимает значение `true`, то накладывание осуществляется с учетом прозрачности

Таблица 31.10 *(окончание)*

Функция	Описание
`imagesetthickness($image, $thickness)`	Устанавливает толщину `$thickness` линий при рисовании (в пикселах)
`imagesetstyle($image, $style)`	Устанавливает стиль `$style`, используемый для линий специального цвета `IMG_COLOR_STYLED` (например, в функциях `imageline()`, `imagepolygon()` и т. д.). Параметр `$style` представляет собой массив пикселов, определяющий стиль линии
`imagesetbrush($image, $brush)`	Устанавливает кисть, используемую для линий специального цвета `IMG_COLOR_STYLEDBRUSHED` (например, в функциях `imageline()`, `imagepolygon()` и т. п.). Параметр `$brush` представляет собой дескриптор изображения
`imagesettile($image, $tile)`	Устанавливает кисть `$tile`, используемую для заливки областей специального цвета `IMG_COLOR_TILED` (например, в функциях `imagefill()`, `imagefillpolygon()` и т. п.)
`imagelayereffect($image, $effect)`	Устанавливает флаг смешивания для использования в эффектах наложения изображений. Параметр `$effect` может принимать следующие значения: • `IMG_EFFECT_REPLACE` — пикселы накладываемых изображений заменяют собой существующие пикселы; • `IMG_EFFECT_ALPHABLEND` — пикселы накладываемых изображений заменяют собой существующие пикселы с учетом прозрачности; • `IMG_EFFECT_NORMAL` — данный режим эквивалентен режиму `IMG_EFFECT_ALPHABLEND`; • `IMG_EFFECT_OVERLAY` — при использовании данного режима белые пикселы изображения остаются белыми, черные — черными, а серые заменяются накладываемым изображением
`imageantialias($image, $on)`	Если параметр `$on` принимает значение `true`, включается сглаживание

31.7.1. Пунктирная линия

Функция `imagesetstyle()` позволяет задать стиль линии, который будет использоваться функциями библиотеки GDLib, если вместо цвета будет передана константа `IMG_COLOR_STYLED`. В листинге 31.22 задается пунктирная линия красного цвета.

Листинг 31.22. Пунктирная линия красного цвета

```
<?php
  // Создаем пустое изображение размером 100x100 пикселов
  $image = imagecreatetruecolor(100, 100);
  if($image)
  {
    // Определяем белый цвет
    $white = imagecolorallocate($image, 255, 255, 255);
    // Определяем красный цвет
    $red   = imagecolorallocate($image, 255, 0, 0);
    // Заполняем изображение белым цветом
    imagefill($image, 0, 0, $white);

    // Задаем массив из пяти красных и пяти белых пикселов
    $style = array($red, $red, $red, $red, $red,
    $white, $white, $white, $white, $white);
    // Устанавливаем стиль линии
    imagesetstyle($image, $style);

    // Рисуем линию
    imageline($image, 0, 0, 100, 100, IMG_COLOR_STYLED);

    // Отправляем HTTP-заголовок Content-type
    header("Content-type: image/gif");
    // Выводим изображение в браузер
    imagegif($image);
    // Уничтожаем изображение в памяти
    imagedestroy($image);
  }
?>
```

Для того чтобы нарисовать пунктирную линию, необходимо сформировать массив `$style`, каждый элемент которого соответствует отдельному пикселу. Результат работы скрипта из листинга 31.22 представлен на рис. 31.10.

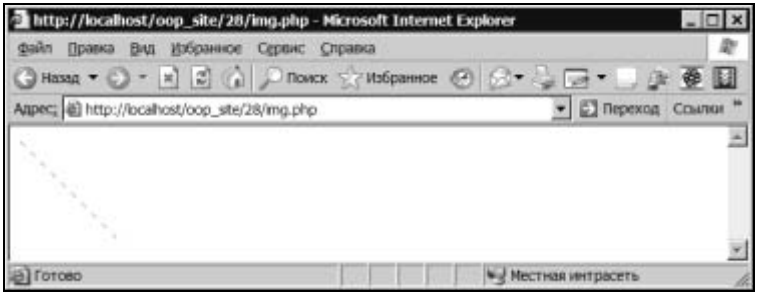


Рис. 31.10. Рисование пунктирной линии

31.8. Функции для работы с текстом

Данная группа функций позволяет размещать текст поверх изображения. В табл. 31.11 представлен список функций данной группы.

Таблица 31.11. *Функции для работы с текстом*

Функция	Описание
`imagefontheight($font)`	Возвращает высоту символов шрифта с дескриптором `$font`
`imagefontwidth($font)`	Возвращает ширину символов шрифта с дескриптором `$font`
`imageftbbox($size, $angle, $font_file, $text [, $extrainfo])`	Вычисляет размеры прямоугольника, который ограничивает строку `$text`, набранную шрифтом FreeType 2. Параметр `$size` определяет размер текста в пикселах, `$angle` — поворот в градусах против часовой стрелки, `$font_file` — путь к шрифту, массив `$extrainfo` содержит дополнительные параметры. В качестве результата функция возвращает массив из восьми элементов: • координата *X* левого нижнего угла; • координата *Y* левого нижнего угла; • координата *X* правого нижнего угла; • координата *Y* правого нижнего угла; • координата *X* правого верхнего угла; • координата *Y* правого верхнего угла; • координата *X* левого верхнего угла; • координата *Y* левого верхнего угла
`imagettfbbox($image, $size, $angle, $x, $y, $color, $font_file, $text [, $extrainfo])`	Вычисляет размеры прямоугольника, который ограничивает строку `$text`, набранную шрифтом FreeType. Параметры и возвращаемые значения совпадают с функцией `imageftbbox()`
`imagepsbbox($text, $font, $size [, $space [, $tightness [, $angle]]])`	Вычисляет размеры прямоугольника, который ограничивает строку `$text`, набранную шрифтом PostScript Type 1. Параметр `$size` определяет размер текста в пикселах, `$angle` — поворот в градусах против часовой стрелки, `$font` — дескриптор шрифта, `$tightness` — межсимвольное расстояние, измеряемое в тысячных долях дюйма, `$space` — размер пробела, также измеряемый в тысячных долях дюйма. В качестве результата функция возвращает массив из четырех элементов: • координата *X* левого нижнего угла; • координата *Y* левого нижнего угла; • координата *X* правого верхнего угла; • координата *Y* правого верхнего угла

Таблица 31.11 *(продолжение)*

Функция	Описание
`imagefttext($image, $size, $angle, $x, $y, $color, $font_file, $text [, $extrainfo])`	Выводит текст `$text` шрифтом FreeType 2 (путь к шрифту определяется параметром `$font_file`), цветом `$color`, размером `$size` и наклоном против часовой стрелки в `$angle` градусов в точке с координатами `$x` и `$y`
`imagettftext($image, $size, $angle, $x, $y, $color, $fontfile, $text)`	Выводит текст `$text` шрифтом FreeType. Параметры совпадают с функцией `imagefttext()`
`imagepstext($image, $text, $font, $size, $foreground, $background, $x, $y [, $space [, $tightness [, $angle [, $antialias_steps]]]])`	Выводит текст `$text` шрифтом PostScript Type1. Параметры совпадают с функциями `imagepsbbox()` и `imagefttext()`. Дополнительный параметр `$antialias_steps` определяет количество цветов, которые используются для сглаживания границ символов. Допустимы значения 4 и 16. Значение 16 имеет смысл использовать только для шрифтов большого размера (более 20 пикселов)
`imagepsloadfont($filename)`	Загружает шрифт в формате PostScript Type1 из файла `$filename`. В случае успеха функция возвращает дескриптор шрифта `$font`, который используется другими функциями этого семейства
`imageloadfont($file)`	Загружает пользовательский растровый шрифт из файла `$file`. В случае успеха функция возвращает дескриптор шрифта, значение которого всегда больше 5 и поэтому не конфликтует со встроенными шрифтами, дескрипторы которых лежат в диапазоне от 0 до 5
`imagepscopyfont($fontindex)`	Создает копию PostScript Type 1 шрифта, загруженного ранее функцией `imagepsloadfont()`. При удачном копировании возвращается дескриптор скопированного шрифта. В противном случае возвращает `false`. Функция копирования шрифтов используется для модификации шрифта без изменения оригинала, с помощью операций растяжения, сжатия, поворота, изменения кодировки и т. п.
`imagepsfreefont($fontindex)`	Освобождает память, занятую шрифтом с дескриптором `$fontindex`
`imagepsencodefont ($font_index, $encodingfile)`	Загружает вектор кодировки из файла `$encodingfile` и изменяет вектор кодировки PostScript Type 1 шрифта `$font_index`. Изменение вектора кодировки необходимо при использовании любого нелатинского шрифта
`imagepsextendfont ($font_index, $extend)`	Сжимает или растягивает шрифт PostScript Type 1 с дескриптором `$font_index` на величину `$extend`

Таблица 31.11 *(окончание)*

Функция	Описание
imagepsslantfont ($*font_index*, $*slant*)	Осуществляет наклон шрифта PostScript Type 1 с дескриптором $font_index на величину $slant
imagechar($*image*, $*font*, $*x*, $*y*, $*c*, $*color*)	Выводит первый символ из строки $*c* в горизонтальном режиме шрифтом с дескриптором $*font* цветом $*color* в точке с координатами $*x* и $*y*
imagecharup($*image*, $*font*, $*x*, $*y*, $*c*, $*color*)	Выводит символ $*c* в вертикальном режиме. Параметры аналогичны функции imagechar()
imagestring($*image*, $*font*, $*x*, $*y*, $*s*, $*color*)	Выводит строку $*s* в горизонтальном режиме шрифтом с дескриптором $*font* цветом $*color* в точке с координатами $*x* и $*y*
imagestringup ($image, $font, $x, $y, $s, $color)	Выводит строку $s в вертикальном режиме. Параметры аналогичны функции imagestring()

31.8.1. Защитное изображение для HTML-формы

Для защиты от автоматической регистрации в HTML-форму добавляют изображение со случайной последовательностью цифр и букв, которые выводятся пользователю в виде динамического изображения. Одновременно в сессию помещается случайная последовательность цифр. Пользователь должен ввести код, который он видит на изображении, в текстовое поле. Если код совпадает с сохраненным в сессии, можно с большой вероятностью сказать, что сайт просматривает человек, поскольку робот обычно не решает задачу распознавания образов (впрочем, настойчивый злоумышленник может попытаться распознать изображение, однако это случается крайне редко). В листинге 31.23 приводится один из возможных вариантов скрипта, формирующего случайную последовательность символов и чисел на изображении.

Листинг 31.23. Вывод случайного изображения

```
<?php
  // Ширина изображения
  $width = 150;
  // Высота изображения
  $height = 50;
  // Количество символов в коде
  $sign = 5;
  // Защитный код
  $code = "";

  // Инициируем сессию
  session_start();
```

```
// Символы, используемые в коде
$letters = array('a','b','c','d','e','f','g','h','j','k','m','n',
                 'p','q','r','s','t','u','v','w','x','y','z','2',
                 '3','4','5','6','7','8','9');
// Компоненты для RGB-цвета
$figures = array('50','70','90','110',
                 '130','150','170','190','210');

// Создаем пустое изображение
$img = imagecreatetruecolor($width, $height);

// Заливаем фон белым цветом
$fon = imagecolorallocate($img, 255, 255, 255);
imagefill($img, 0, 0, $fon);

// Заливаем фон точками
for($j=0; $j<$width; $j++)
{
  for($i=0; $i<($height*$width)/1000; $i++)
  {
    // Формируем случайный цвет
       $color = imagecolorallocatealpha(
                     $img,
                     $figures[rand(0,count($figures)-1)],
                     $figures[rand(0,count($figures)-1)],
                     $figures[rand(0,count($figures)-1)],
                  rand(10,30));
    // Устанавливаем случайную точку случайного цвета
    imagesetpixel($img,
                  rand(0,$width),
                  rand(0,$height),
                  $color);
  }
}

// Накладываем защитный код
for($i=0; $i<$sign; $i++)
{
  // Ориентир
  $h = 1;
  // Рисуем
  $color = imagecolorallocatealpha(
                  $img,
                  $figures[rand(0,count($figures)-1)],
                  $figures[rand(0,count($figures)-1)],
                  $figures[rand(0,count($figures)-1)],
```

```
                    rand(10,30));

    // Генерируем случайный символ
    $letter = $letters[rand(0,sizeof($letters)-1)];

    // Формируем координаты для вывода символа
    if(empty($x)) $x = $width*0.08;
    else $x = $x + ($width*0.8)/$sign+rand(0,$width*0.01);

    if($h == rand(1,2)) $y = (($height*1)/4) + rand(0,$height*0.1);
    else $y = (($height*1)/4) - rand(0,$height*0.1);

    // Запоминаем символ в переменной $code
    $code .= $letter;
    // Изменяем регистр символа
    if($h == rand(0,1)) $letter = strtoupper($letter);
    // Выводим символ на изображение
    imagestring($img, 6 ,$x, $y, $letter, $color);
  }

  // Помещаем защитный код в сессию
  $_SESSION['code'] = $code;

  // Выводим изображение
  header ("Content-type: image/jpeg");
  imagejpeg($img);
?>
```

Скрипт из листинга 31.23 позволяет создать изображение шириной `$width` и высотой `$height` пикселов, содержащее `$sign` случайных символов случайного цвета. Символы задаются массивом `$letters`, компоненты цвета — массивом `$figures`. После этого изображение заливается белым цветом, на котором размещаются в случайном порядке множество цветных пикселов, а затем генерируется и выводится случайный код `$code`. Переменная `$code` помимо всего прочего помещается в сессию, для того чтобы обработчик HTML-формы имел возможность сравнить введенный пользователем и сгенерированный коды. Так как данные из сессии не покидают сервер, злоумышленники не имеют возможности украсть защитный код и вынуждены решать сложную задачу распознавания образов, которые не всегда оправданны.

В листинге 31.24 приводится пример HTML-формы, защищенной при помощи изображения с защитным кодом (скрипт из листинга 31.23 помещен в файл с именем img.php).

Листинг 31.24. Защитное изображение для HTML-формы

```
<?php
  // Инициируем сессию
  session_start();
?>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>Пример</title>
</head>
<body>
<?php
  if(isset($_POST['code']) && isset($_SESSION['code']))
  {
    if(strtolower($_POST['code']) == $_SESSION['code'])
         echo '<font color="green">Защитный код верен!</font>';
       else
         echo '<font color="red">Неверный защитный код!</font>';
  }
  else
  {
    ?>
      <form method="post">
        <img src="img.php" border="0" alt="Введите защитный код"><br>
        <input type="text" name="code"><br>
        <input type="submit" value="Ввести">
      </form>
    <?php
  }
?>
</body>
</html>
```

На рис. 31.11 представлен результат работы скрипта из листинга 31.24.

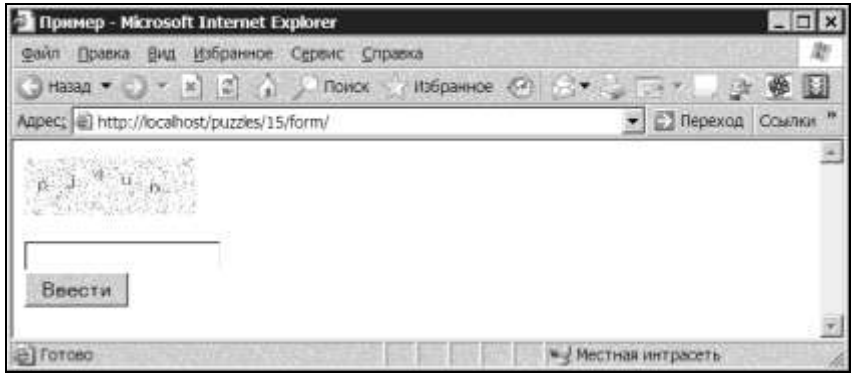


Рис. 31.11. Защита изображения защитным кодом

Заключение

Нам бы очень хотелось, чтобы после того, как вы перелистнули эту заключительную страницу, мы бы с вами не расстались, а встретились на наших форумах. Если у вас есть вопросы по программированию, то вы можете обсудить их на нашем форуме **http://www.softtime.ru/forum/**. Об общежизненных вопросах можно подискутировать на форуме по адресу **http://www.softtime.org/forum/**. Также сообщаем, что по адресу **http://www.softtime.org/blog-mk/** находится блог одного из авторов этой книги, директора IT-Студии Softtime, Кузнецова М. В. Приходите, читайте, обсуждайте.

А самое главное — всего вам доброго!

И, конечно, до новых встреч!

ПРИЛОЖЕНИЯ



Приложение 1

Установка и настройка PHP, Web-сервера Apache и MySQL-сервера

Программирование на PHP отличается от программирования на других языках достаточно сложной системой настройки среды. Для того чтобы создать удобную среду программирования, не достаточно установить инсталляционные пакеты — потребуется их тщательное конфигурирование. Так как и язык программирования PHP, и Web-сервер Apache, и MySQL-сервер пришли из операционной системы UNIX, их настройка и администрирование сводятся к редактированию конфигурационных файлов и работе в командной строке. Такой подход часто сбивает с толку программистов, не имеющих опыта работы в UNIX-подобных операционных системах. В данном приложении рассматривается процесс создания удобного рабочего места на локальной машине, которое позволит тестировать сайты без размещения их в Интернете на серверах хост-компаний.

> ***Замечание***
>
> Web-среда является динамичной средой, в которой часто происходят изменения, поэтому перед установкой сверьтесь со статьями, расположенными по адресу **http://www.softtime.ru/article/index.php?id_page=9**, которые авторы регулярно обновляют. По ссылке вы также можете загрузить готовые конфигурационные файлы httpd.conf (для Apache) и php.ini (для PHP). Если что-то не устанавливается или вам что-то не понятно, вы можете смело обращаться на форум **http://www.softtime.ru/forum/index.php?id_forum=5**, где получите ответы на все вопросы.

П1.1. Где взять дистрибутивы?

Прежде чем предпринять какие-либо действия, необходимо обзавестись дистрибутивами интерпретатора PHP, Web-сервера Apache и MySQL-сервера. Поиски дистрибутивов следует начинать с официальных сайтов данных программных продуктов (табл. П1.1).

Таблица П1.1. *Официальные сайты PHP, Apache и MySQL*

Программный продукт	Web-сайт
Интерпретатор PHP	**http://www.php.net**
Web-сервер Apache	**http://www.apache.org**
СУБД MySQL	**http://dev.mysql.com**

Поскольку версии программных продуктов постоянно меняются и указать постоянную ссылку невозможно, приведем алгоритм загрузки свежей версии каждого из программных продуктов.

Замечание

Свежие версии программных продуктов всегда доступны на нашем сайте **http://www.softtime.ru/**.

П1.1.1. Дистрибутив PHP

Загрузив страницу **http://www.php.net**, необходимо перейти на страницу **downloads**, которая на момент написания книги имела адрес **http://www.php.net/downloads.php**. На данной странице PHP доступен в двух форматах: исходных кодах (**Complete Source Code**) и предкомпилированном варианте (**Windows Binaries**). Нас будет интересовать предкомпилированная версия, которая также распространяется в двух вариантах: в виде автоматического установщика (например, php-5.2.5-installer.exe) и в виде zip-архива (например, php-5.2.5-Win32.zip). Автоматический установщик удобен, но содержит лишь ограниченную версию PHP (для сравнения: инсталлятор занимает 2 Мбайт, а zip-архив 8 Мбайт). В состав полной версии входят расширения (например, для работы с СУБД MySQL, для создания динамических изображений и т. п.). Кроме того, использование автоматического инсталлятора не избавляет от необходимости настройки конфигурационного файла Web-сервера Apache — httpd.conf, поэтому рекомендуется загружать zip-архив.

В разделе **Windows Binaries** следует выбрать ссылку **PHP 5.x.x zip package**, которая приведет на страницу со списком зеркал, откуда можно загрузить текущую версию PHP. На данной странице можно выбрать любую ссылку; разница в них заключается лишь в географическом расположении серверов. Разумнее выбрать сервер, который расположен в Российской Федерации (**Russian Federation**). Ссылки на такие серверы помечены значком российского флага.

Замечание

С сайта **http://www.php.net** можно загрузить справочник по языку PHP, в том числе частично переведенный на русский язык. Для этого необходимо перейти на страницу **documentation** (**http://www.php.net/docs.php**) и там выбрать ссылку **downloads** (**http://www.php.net/download-docs.php**). На странице загрузки будет представлена таблица, позволяющая загрузить справочник в трех форматах для каждого из доступных языков. Для пользователей Windows лучше воспользоваться chm-форматом, который позволяет осуществлять поиск в справочнике.

П1.1.2. Дистрибутив Apache

Для того чтобы тестировать и просматривать сайты на локальной машине, необходимо установить Web-сервер Apache. Поиск дистрибутива следует начать с официальной страницы **http://www.apache.org**, выбрав ссылку **Download from a mirror** (**http://www.apache.org/dyn/closer.cgi**). На открывшейся странице будет представлен список HTTP- и FTP-серверов, откуда можно загрузить Web-сервер Apache.

При поиске следует помнить, что Apache может также называться httpd, по имени его демона в UNIX. На зеркалах обычно много различных файлов, например:

- httpd-2.0.58-win32-src.zip — архив с исходными кодами (src) для Windows (win32) Web-сервера Apache (httpd) версии 2.0.58;
- httpd-2.0.58.tar.gz — то же самое, но для Linux, в котором программы принято распространять в исходных кодах;
- apache_2.0.58-win32-x86-no_ssl.exe — а вот это откомпилированный под архитектуру (*x*86) для Windows (win32) без поддержки SSL (no_ssl) сервер Apache (apache) версии 2.0.58 — именно его и следует загрузить.

> ***ЗАМЕЧАНИЕ***
>
> Бинарные коды дистрибутивов Apache для Windows распространяются в нескольких вариантах, с расширением как exe, так и msi и имеют название вида httpd_версия_win32_*_.msi.

> ***ЗАМЕЧАНИЕ***
>
> Скачать дистрибутивы Apache для Windows можно по адресу **http://apache.rinet.ru/dist/httpd/binaries/win32/**.

П1.1.3. Дистрибутив MySQL

Дистрибутив MySQL можно загрузить со страницы **http://dev.mysql.com/downloads/**. На момент написания книги для загрузки были доступны три версии СУБД MySQL:

- MySQL 5.0 — рекомендуемая версия MySQL;
- MySQL 5.1 — разрабатываемая версия, находящаяся в стадии бета-тестирования;
- MySQL 4.1 — устаревшая, но поддерживаемая версия.

> ***ЗАМЕЧАНИЕ***
>
> На официальном сайте MySQL для загрузки всегда доступны три версии. Как только разрабатываемая версия переходит в стадию релиза (рекомендуемой версии), в нее прекращают добавлять нововведения и лишь исправляют найденные ошибки. Все нововведения добавляются в новую разрабатываемую версию. При этом рекомендуемая ранее версия получает статус устаревшей, и ее поддержка прекращается.

> ***ЗАМЕЧАНИЕ***
>
> Полное справочное руководство по MySQL на русском языке можно найти на официальном сайте MySQL по адресу **http://dev.mysql.com/doc/mysql/ru/index.html**.

Из трех доступных дистрибутивов следует выбрать рекомендуемый. На открывшейся странице будет представлен список дистрибутивов, скомпилированных под разные операционные системы. Для Linux имеются дистрибутивы как в gzip-архивах (tar.gz), так и в rpm-пакетах (rpm), откомпилированные под 32- и 64-битную архитектуры. Для Windows дистрибутивы представлены в трех вариантах:

- Windows Essentials (*x*86) — урезанная версия дистрибутива, из которой удалены все вспомогательные утилиты, по сути, это "голый" сервер MySQL;

- Windows (*x*86) — полная версия дистрибутива MySQL, включающая автоматический установщик;
- Without installer (unzip in C:\) — полная версия MySQL без автоматического установщика.

Рекомендуется выбрать дистрибутив Windows (*x*86). Несмотря на то, что его объем превышает все остальные дистрибутивы из Windows-серии, он наиболее удобен в работе. Дальнейшее изложение материала будет основано именно на этом дистрибутиве.

Помимо самого дистрибутива MySQL имеет смысл загрузить графические клиенты для работы с MySQL-сервером, которые свободно распространяются на сайте **http://dev.mysql.com**. (табл. П1.2).

Таблица П1.2. *Графические клиенты MySQL*

Программа	Ссылка
MySQL Administrator	**http://dev.mysql.com/downloads/administrator/1.0.html**
MySQL Query Browser	**http://dev.mysql.com/downloads/query-browser/1.1.html**
MySQL Control Center	**http://dev.mysql.com/downloads/other/mysqlcc.html**

П1.2. Установка Web-сервера Apache под Windows

Apache под Windows доступен как в виде исходных кодов, так и виде откомпилированного пакета. Исходные коды могут понадобиться в том случае, если вы решили при установке перекомпилировать Apache. Перекомпилирование исходных кодов необходимо, когда нужно исключить из исполняемой версии неиспользуемые функции или включить поддержку функций, не входящих в стандартную поставку. В этом случае необходимо наличие установленной на машине среды разработки Microsoft Visual Studio. Если стандартная компиляция сервера вас устраивает, можно приступить к установке.

> ***Примечание***
>
> Несмотря на то, что Web-сервер Apache уже давно перенесен и успешно функционирует под Windows, только с версии Apache 2.0.0 произведена оптимизация его исходных кодов для использования системных вызовов Windows. До этого использовался уровень POSIX, что приводило к недостаточной производительности при работе с Apache под Windows.

После двойного щелчка на файле httpd_версия_win32_*_.msi запустится Microsoft Installer.

Для начала установки нажмите кнопку **Next**, после чего появится окно с лицензионным соглашением. После принятия лицензионного соглашения следует перейти к следующему окну с краткой информацией о нововведениях во второй версии Apache. Следующее окно, показанное на рис. П1.1, позволяет ввести информацию о сервере: доменное имя сервера, имя сервера и адрес электронной почты администратора.

Если установка происходит на локальную машину, то в поля для доменного имени и имени сервера следует ввести `localhost`. Очень важно заполнить эти реквизиты, иначе вместо IP-адреса 127.0.0.1 в конфигурационный Apache-файл httpd.conf будет записан IP-адрес 0.0.0.0, и сервер не сможет запуститься. В нижней части окна предлагается выбрать номер порта, по которому сервер будет принимать запросы (80 или 8080).

ЗАМЕЧАНИЕ

Обычно используется стандартный для протокола HTTP 80-й порт, однако если для запуска сервера Apache по 80 порту не достаточно прав, следует выбрать порт 8080.

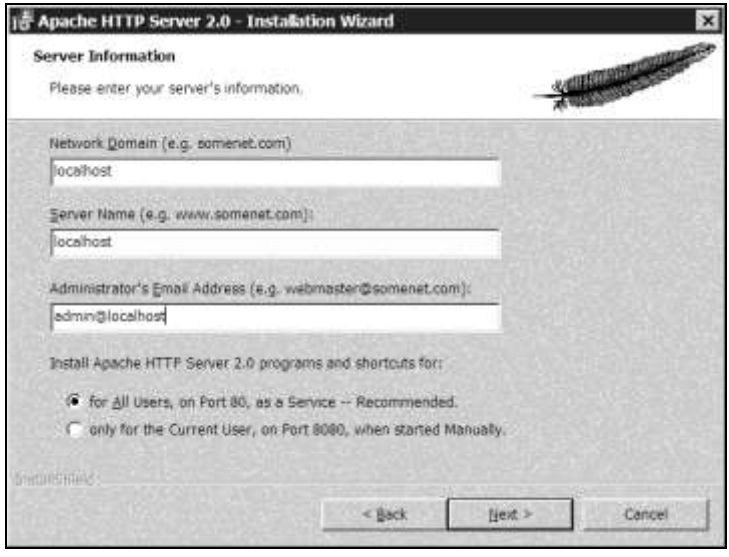


Рис. П1.1. Настройка параметров сервера

Далее будет предложен способ установки (рис. П1.2): типичный (**Typical**) или выборочный (**Custom**), позволяющий выбрать компоненты сервера вручную. Следующее окно позволяет выбрать каталог установки сервера, по умолчанию это C:\Program Files\Apache Group.

Затем мастер установки сообщит о готовности к процессу установки, и после нажатия кнопки **Install** будет произведено копирование файлов сервера. Если установка прошла успешно, Windows автоматически запустит Apache. По умолчанию вместе с сервером запускается утилита мониторинга работы сервера ApachtMonitor, значок которой помещается в системном триере.

ЗАМЕЧАНИЕ

Работа утилиты ApachtMonitor не влияет на работу сервера, и ее можно отключить. Если далее она потребуется, ее можно найти в подкаталоге bin каталога установки Apache.

Проверить работоспособность сервера можно, набрав в браузере **http://localhost**. В случае успешной установки вы увидите приветственную страницу Apache (рис. П1.3).

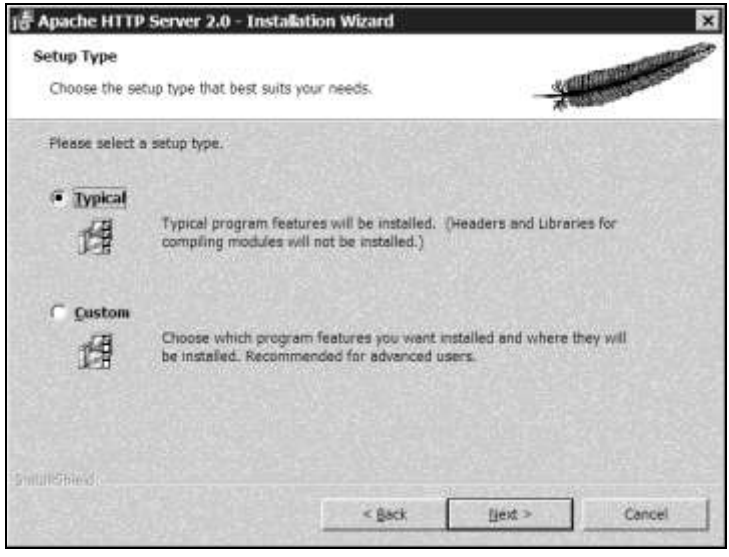


Рис. П1.2. Выбор способа установки

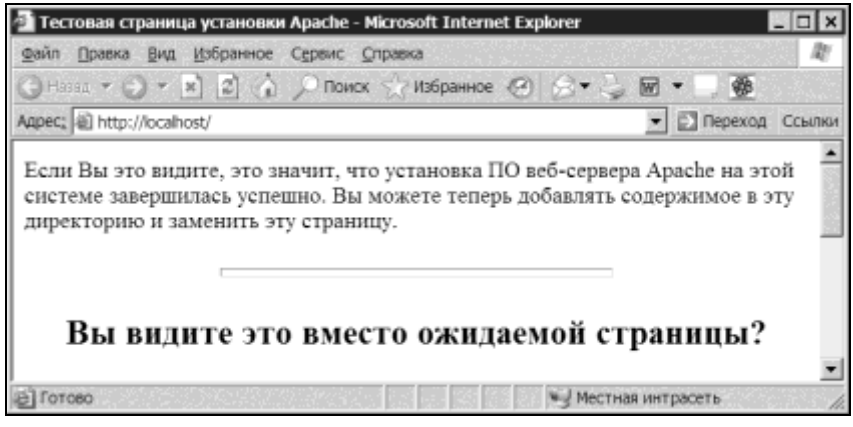


Рис. П1.3. Приветственная страница Web-сервера Apache

П1.3. Установка Web-сервера Apache под Linux

Разархивируйте дистрибутив при помощи команды:

```
tar –xzf apache_x.x.xx.xx.tar.gz
```

Перейдите в каталог исходного кода и выполните настройку:

```
cd apache_x.x.xx.xx
./configure –prefix=/usr/local/apache/\ --enable-chared=max\
--enable-module=most
```

Далее выполните следующие команды:

```
make
make install
```

Для запуска Apache нужно выполнить команду:

```
/usr/local/apache/bin/apachectl start
```

Теперь вы можете попасть на ваш Web-сайт с помощью графического браузера (если вы работаете в X Window) или через команду:

```
lynx http://localhost/
```

Проверить, выполняется ли Apache, можно командой:

```
ps auxwww | grep httpd
```

Если все нормально, эта команда покажет пять рабочих процессов с одинаковым именем `httpd`.

Если вы хотите, чтобы Apache запускался автоматически при загрузке компьютера, можно поместить в каталог /etc/rc.d/init.d/ следующий сценарий с именем apache:

```
cp /usr/local/apache/bin/apachectl /etc/rc.d/init.d/apache
chmod 755 /etc/rc.d/init.d/apache
```

Эти команды создают исполняемый сценарий оболочки, доступный на всех уровнях загрузки. Затем в /etc/rc.d/rc3.d/ нужно выполнить команду:

```
ln –s ../init.d/apache S99apache
```

Эта команда создает символическую ссылку в каталоге rc3.d. Во время загрузки команды из этого каталога выполняются в алфавитном порядке, начиная с "S".

П1.4. Настройка виртуальных хостов

После того как установка Web-сервера Apache будет успешно завершена, необходимо создать виртуальный хост. Виртуальный хост позволяет указать каталог, где будут располагаться HTML- и PHP-файлы. Кроме этого, виртуальные хосты позволяют перенести HTML- и PHP-файлы в другой раздел диска, что может быть удобно при резервном копировании данных.

Создадим виртуальный хост **localhost**, файлы которого будут храниться в каталоге D:\data\. Перечень виртуальных узлов обычно расположен в конце файла httpd.conf. Сначала при помощи директивы `NameVirtualHost` требуется указать, какой IP-адрес и порт используются для виртуального хоста:

```
NameVirtualHost 127.0.0.1:80
```

> ***Замечание***
>
> 80 в конце IP-адреса указывает порт, который будет прослушивать сервер Apache. Так, если необходимо назначить Web-серверу альтернативный порт, можно указать вместо 80 порт 8080. Однако этого лучше избегать, т. к. в этом случае при обращении к серверу придется явно указывать порт в адресе: **http://localhost:8080/index.html**, поскольку если порт не указывается, браузер автоматически обращается по порту 80.

Далее необходимо создать контейнер `<VirtualHost>`, который будет определять конфигурацию виртуального хоста (листинг П1.1).

Листинг П1.1. Контейнер `<VirtualHost>`

```
<VirtualHost 127.0.0.1:80>
    ServerAdmin webmaster@may_domain.ru
    DocumentRoot d:/data
    ServerName localhost
    ErrorLog logs/dummy-host.example.com-error_log
    CustomLog logs/dummy-host.example.com-access_log common
</VirtualHost>
```

Адрес в контейнере виртуального хоста должен совпадать с адресом, указанным в директиве `NameVirtualHost`. В листинге П1.1 директива `ServerAdmin` определяет адрес электронной почты администратора. Именно этот адрес будет выводиться при генерации Web-сервером страниц с сообщениями об ошибках.

Директива `DocumentRoot` определяет физическое расположение виртуального хоста на жестком диске. Теперь файлы можно размещать в каталоге D:\data\, и они будут доступны при обращении к адресу **http://localhost/**.

> ***Замечание***
>
> Если в пути к каталогу виртуального хоста встречаются пробелы, путь следует заключать в двойные кавычки.

Директива `ServerName` задает имя сервера, которое в данном случае может быть любым. Директивы `ErrorLog` и `CustomLog` определяют путь к файлам журналов (log-файлам), в которые Web-сервер Apache помещает ошибки и фиксирует обращения к страницам сайта.

Для проверки работоспособности виртуального хоста необходимо создать каталог D:\data\ и разместить в нем файл index.html, содержащий фразу `"Hello, world!"`.

Для того чтобы изменения конфигурационного файла вступили в действие, следует перезапустить Web-сервер Apache (данные из конфигурационного файла читаются один раз при загрузке сервера). Если конфигурационный файл не содержит ошибок, после перезапуска сервера и обращения по адресу **http://localhost/** браузер должен выглядеть так, как это представлено на рис. П1.4.

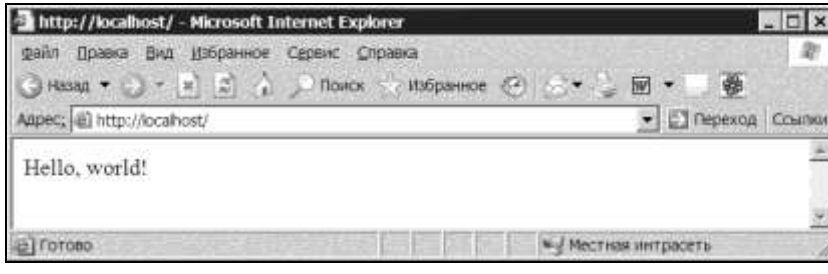


Рис. П1.4. Настройка виртуального хоста

Можно настроить сразу несколько виртуальных хостов. Для этого необходимо для каждого виртуального хоста прописать директиву `NameVirtualHost` и контейнер `VirtualHost`. Создадим дополнительный виртуальный хост **www.site.ru** таким образом, чтобы при обращении по адресу **http://www.site.ru** загружался сайт, расположенный в каталоге D:\www.site.ru\.

> ***Замечание***
>
> Для владельцев операционной системы Windows XP SP2 для настройки нескольких виртуальных хостов потребуется загрузить исправление WindowsXP-KB884020-x86-rus.exe (**http://www.microsoft.com/downloads/details.aspx?FamilyID=17d997d2-5034-4bbb-b74d-ad8430a1f7c8&DisplayLang=ru**). В противном случае настроить несколько виртуальных хостов не удастся.

Поместим виртуальный хост на IP-адрес 127.0.0.2, прописав вторую директиву `NameVirtualHost`:

```
NameVirtualHost 127.0.0.1:80
NameVirtualHost 127.0.0.2:80
```

К уже имеющемуся контейнеру `<VirtualHost>` для `localhost` добавим второй контейнер виртуального хоста **www.site.ru** (листинг П1.2).

> ***Замечание***
>
> Файлы, определенные в директивах `ErrorLog` и `CustomLog`, обязательно должны существовать, иначе Web-сервер не запустится.

Листинг П1.2. Контейнер `<VirtualHost>`

```
<VirtualHost 127.0.0.1:80>
    ServerAdmin webmaster@may_domain.ru
    DocumentRoot d:/data
    ServerName localhost
    ErrorLog logs/dummy-host.example.com-error_log
    CustomLog logs/dummy-host.example.com-access_log common
</VirtualHost>
<VirtualHost 127.0.0.2:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot "D:/www.site.ru"
    ServerName www.site.ru
    ErrorLog logs/site-error_log
    CustomLog logs/site-access_log common
</VirtualHost>
```

Помимо этого, может потребоваться исправить значение директивы `Listen 127.0.0.2:80` и `BindAddress 127.0.0.2:80` для Apache версии 1.3.*x*.

В Windows соответствие между IP-адресами и именами, вводимыми в адресную строку браузера, устанавливается в файле C:\Windows\system32\drivers\etc\hosts. В данном

файле необходимо установить соответствие между адресом **www.site.ru** и IP-адресом 127.0.0.1 (листинг П1.3).

Листинг П1.3. Файл hosts

```
# (C) Корпорация Майкрософт (Microsoft Corp.), 1993-1999
#
# Это образец файла HOSTS, используемый Microsoft TCP/IP для Windows.
#
# Этот файл содержит сопоставления IP-адресов именам узлов.
# Каждый элемент должен располагаться в отдельной строке. IP-адрес должен
# находиться в первом столбце, за ним должно следовать соответствующее
# имя. IP-адрес и имя узла должны разделяться хотя бы одним пробелом.
#
# Кроме того, в некоторых строках могут быть вставлены комментарии
# (такие, как эта строка), они должны следовать за именем узла и
# отделяться от него символом '#'.
#
# Например:
#
#      102.54.94.97     rhino.acme.com          # исходный сервер
#       38.25.63.10     x.acme.com              # узел клиента x

127.0.0.1       localhost
127.0.0.2       www.site.ru
```

Создадим каталог D:\www.site.ru и поместим в него файл index.html следующего содержания (листинг П1.4).

Листинг П1.4. Содержимое файла index.html

```
Это файл index.html виртуального хоста www.site.ru
```

Теперь если перезагрузить сервер и набрать в адресной строке адрес **http://www.site.ru**, можно увидеть страницу, представленную на рис. П1.5.

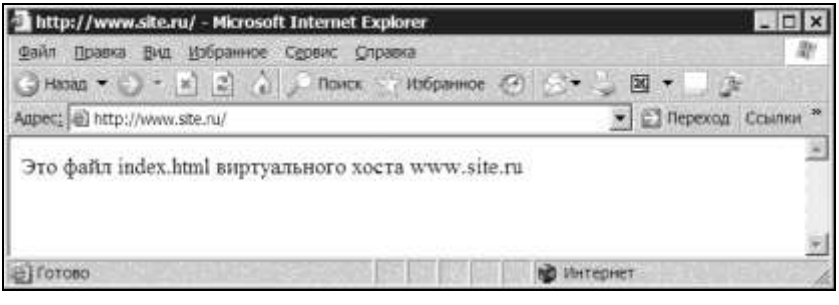


Рис. П1.5. Страница виртуального хоста **www.site.ru**

Следует быть осторожным в выборе имени виртуального хоста, т. к. если оно совпадет с реальным сайтом, открыть его уже не удастся.

П1.5. Настройка кодировки по умолчанию

Для того чтобы настроить кодировку по умолчанию, необходимо отредактировать конфигурационный файл httpd.conf Web-сервера Apache, который можно обнаружить в каталоге C:\Program files\Apache Group\Apache2\conf. В первую очередь следует исправить кодировку документов. Для того чтобы документы воспринимались в кодировке cp1251 (кодировке Windows), необходимо исправить директиву `AddDefaultCharset` так, как это представлено ниже:

```
AddDefaultCharset windows-1251
```

Клиенту будет сообщаться, что он имеет кодировку cp1251, если кодировка не установлена в заголовке HTML-документа.

П1.6. Управление запуском и остановкой Web-сервера Apache

Если при установке сервера в качестве порта, по которому Apache принимает запросы, был выбран 80-й порт (см. рис. П1.1), допускается запуск Apache в качестве сервиса. Для запуска консоли управления выполните команду **Пуск | Настройка | Панель управления | Администрирование | Службы**. В появившемся окне консоли, приведенном на рис. П1.6, следует выбрать сервис Apache2. Контекстное меню позволяет осуществлять запуск, остановку и перезапуск сервиса.

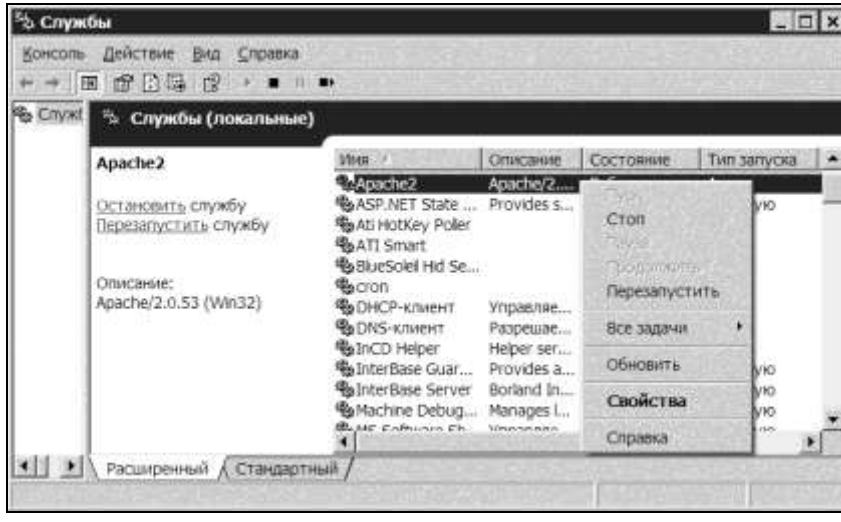


Рис. П1.6. Службы Windows

Службы Windows позволяют производить запуск фоновых приложений при старте системы. Для этого необходимо перейти в окно **Свойства**, выбрав в контекстном меню сервиса пункт **Свойства** (рис. П1.6), и в появившемся окне (рис. П1.7) в выпадающем списке **Тип запуска** выбрать пункт **Авто.**

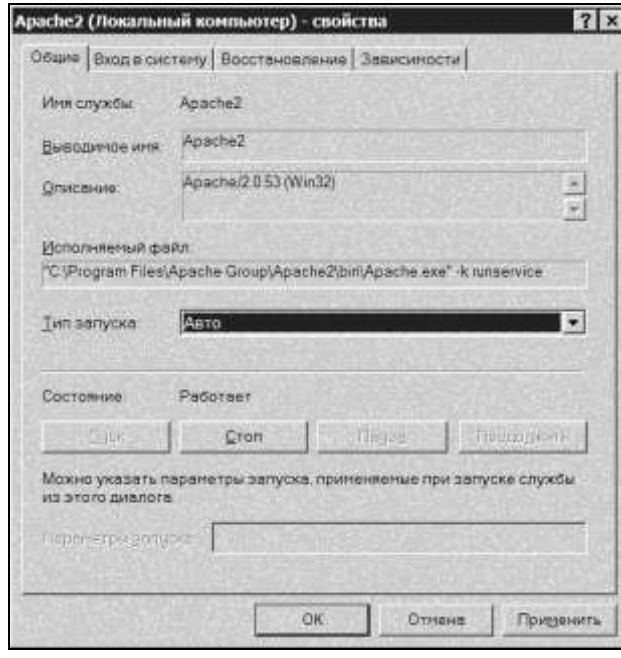


Рис. П1.7. Окно свойства сервиса Apache2

П1.7. Управление Apache из командной строки

В Windows запускать, останавливать и перезапускать сервер Apache из командной строки можно при помощи следующих команд:

- `Apache -k start` (запуск);
- `Apache -k restart` (перезапуск);
- `Apache -k stop` или `Apache -k shutdown` (остановка).

Все команды следует выполнять из каталога bin сервера Apache (C:\Program Files\Apache Group\Apache2\bin\).

> ***Замечание***
>
> Команды с ключом `-k` являются управляющими командами сервера Apache. Так, команды `Apache -k install` и `Apache -k uninstall` позволяют установить и удалить сервис Apache2.

ЗАМЕЧАНИЕ

Получить полный список команд управления с их кратким описанием можно, воспользовавшись командой `Apache -help` или обратившись к документации к серверу Apache.

Команда `Apache -t` позволяет проверить конфигурационные файлы Apache на предмет наличия синтаксических ошибок. В случае их отсутствия выдается строка `"Syntax OK"`. Если же в конфигурационных файлах имеются ошибки, то в результате тестирования программа выдаст сообщение об ошибке, например:

```
Syntax error on line 57 of C:/Program Files/Apache Group/Apache2/conf/
httpd.conf:
ServerRoot takes one argument, Common directory of server-related files
```

Для управления Apache в Linux предназначена утилита `apachectl`. В дистрибутивах, основанных на Red Hat, данная утилита находится в каталоге /usr/sbin/. Для старта, перезапуска или остановки сервера используются следующие команды:

- `/usr/sbin/apachectl start` (запуск);
- `/usr/sbin/apachectl restart` (перезапуск);
- `/usr/sbin/apachectl stop` (остановка).

Допускается также использование параметров сервера `-k start`, `-k restart` и `-k stop`, описанных ранее, только в качестве исполняемого модуля в Linux выступает не Apache, а файл httpd, который находится в каталоге /usr/sbin/.

В дистрибутивах, основанных на Red Hat, можно также использовать сервисы для управления Web-сервером Apache. Для этого необходимо выполнить следующие команды:

- `/etc/rc.d/init.d/httpd start` (запуск);
- `/etc/rc.d/init.d/httpd restart` (перезапуск);
- `/etc/rc.d/init.d/httpd stop` (остановка).

П1.8. Установка PHP под Windows

Для установки PHP следует создать каталог C:\php и разместить в нем файлы из zip-архива дистрибутива. После этого нужно переименовать конфигурационный файл php.ini-dist в php.ini и скопировать его в каталог C:\Windows.

Далее необходимо сообщить Web-серверу о наличии установленного PHP. Установка PHP возможна двумя способами: как модуль Apache и как внешнее CGI-приложение. Далее будут рассмотрены оба варианта установки.

П1.8.1. Установка PHP в качестве модуля

Установка PHP в качестве модуля немного повышает быстродействие, т. к. модуль PHP загружается один раз при запуске Web-сервера.

Замечание

При установке PHP в качестве модуля настройки из php.ini читаются один раз при запуске Web-сервера. Поэтому при изменении файла php.ini необходимо перезагрузить Apache для того, чтобы внесенные изменения вступили в силу.

Для установки PHP откройте конфигурационный Apache-файл httpd.conf и удалите символы комментариев напротив строк, указанных в листинге П1.5, при необходимости изменив их.

Замечание

Строки, представленные в листинге П1.5, должны располагаться в районе других директив `AddType`. Если их поместить ближе к концу, например, непосредственно перед виртуальными хостами, они не будут восприняты Web-сервером Apache.

Листинг П1.5. Подключение PHP, как модуль Apache

```
AddType application/x-httpd-php phtml php
LoadModule php5_module c:/php/php5apache2.dll
```

В Apache версии 2.2.*x* появилась возможность указать путь к конфигурационному файлу при помощи директивы `PHPIniDir`.

В корневом каталоге C:/php имеется несколько динамических библиотек, каждая из которых предназначена для своего случая. Так php5apache2.dll используется для подключения к Apache 2.0.*x*, php5apache2_2.dll — для подключения к Apache 2.2.*x*, php5apache.dll — для подключения к Apache 1.3.*x*, php5isapi.dll применяется для подключения к Web-серверу IIS.

П1.8.2. Установка PHP как CGI-приложения

При установке PHP как CGI-приложения интерпретатор PHP будет загружаться каждый раз при вызове PHP-сценария. В связи с этим возможно некоторое ухудшение быстродействия. Впрочем, это справедливо лишь для загруженных серверов и практически незаметно на локальной машине.

Если PHP установлен как CGI-приложение, то при внесении изменений в файл php.ini Web-сервер Apache перезагружать не следует, т. к. установки читаются каждый раз при выполнении PHP-сценария.

Примечание

При установке PHP как CGI перестанут работать некоторые заголовки, например, нельзя будет организовать авторизацию пользователей средствами HTTP Basic Authentification. Авторизацию можно будет реализовать только средствами самого Apache с помощью файлов .htaccess.

Для установки PHP необходимо добавить в конфигурационный файл httpd.conf строки, представленные в листинге П1.6.

ЗАМЕЧАНИЕ

В версии PHP 4 вместо php-cgi.exe следует указывать php.exe.

ЗАМЕЧАНИЕ

Строки, представленные в листинге П1.6, должны располагаться в районе других директив `AddType`. Если их поместить ближе к концу, например, непосредственно перед виртуальными хостами, они не будут восприняты Web-сервером Apache.

Листинг П1.6. Файл httpd.conf. Подключение PHP, как CGI-приложение

```
AddType application/x-httpd-php phtml php
<Directory "C:/php">
  Options ExecCGI
</Directory>
ScriptAlias "/php_dir/" "C:/php/"
Action application/x-httpd-php "/php_dir/php-cgi.exe"
```

Кроме этого, может оказаться полезной настройка директивы `DirectoryIndex`, которая отвечает за индексный файл каталога: надо добавить к индексным файлам значение index.php.

```
DirectoryIndex index.php index.html index.html.var
```

Теперь если в адресе не указывается страница (например, пользователь набирает адрес **http://localhost/** вместо **http://localhost/index.php**), Web-сервер автоматически будет искать в текущем каталоге файл index.php. Если такой файл не найден, будет выполнен поиск файла index.html. Если и этот файл не обнаружится, будет произведен поиск index.html.var. В директиве `DirectoryIndex` можно указать любые другие индексные файлы, например, index.htm, index.php3, index.phtml и т. п.

Для проверки работоспособности PHP-файла в каталоге виртуального хоста необходимо создать PHP-файл index.php следующего содержания (листинг П1.7).

Листинг П1.7. Проверочный PHP-скрипт

```
<?php
  phpinfo();
?>
```

Функция `phpinfo()` выводит в окно браузера отчет о конфигурации PHP. Если PHP успешно связан с Web-сервером, то результат может выглядеть так, как это представлено на рис. П1.8.

Следует отметить, что отчет, предоставляемый функцией `phpinfo()`, указывает на путь к конфигурационному файлу php.ini. Его можно выяснить в строке `Configuration File (php.ini) Path`. Этот путь может быть полезным, если имеется подозрение, что редактируется не та копия, которая используется PHP (например, внесенные изменения не вступают в силу).

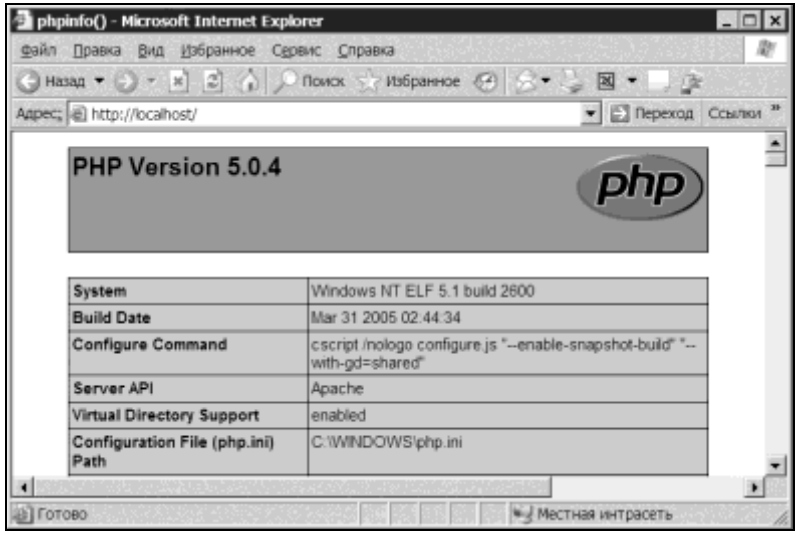


Рис. П1.8. Отчет функции `phpinfo()`

П1.9. Установка PHP под Linux

После того как дистрибутив PHP загружен с сайта **http:/www.php.net** в каталог /usr/local/src, следует разархивировать файлы дистрибутива:

```
tar –xzf php-5.x.x.tar.gz
```

Затем перейти в каталог PHP:

```
cd php-5.x.x
```

Теперь необходимо создать сценарий для настройки PHP. Для этого создайте файл с именем config.sh и поместите в него следующий код:

```
./configure\
--with-apxs=/usr/local/apache/bin/apxs\
--with-mysql=/var/lib/mysql/
```

Все другие необходимые расширения можно поместить в отдельные строки этого файла. Как можно видеть, в этом сценарии задан параметр `--with-apxs` вместо `--with-apache`. Это сделано для того, чтобы PHP устанавливался не как статический модуль, а как динамически разделяемый объект. В этом случае можно обновлять PHP, не компилируя Apache заново. Если устанавливать PHP как статический модуль, то при обновлении PHP нужно будет вновь компилировать Apache.

Далее следует сохранить файл config.sh и выполнить команду:

```
chmod 755 config.sh
```

Запуск сценария производится при помощи команды:

```
./config.sh
```

После чего можно скомпилировать и установить PHP:

```
make
make install
```



Приложение 2

Установка MySQL

Как и любая другая СУБД (система управления базами данных), MySQL является сложным программным комплексом, от установки и настройки которого зависит производительность, устойчивость и безопасность. Данная глава посвящена установке и первичной настройке СУБД MySQL. Будут рассмотрены вопросы, которые возникают практически сразу при работе с любой СУБД: как установить и наиболее эффективно настроить СУБД, а также как переносить базы данных с одного сервера на другой.

П2.1. Установка MySQL под Windows

П2.1.1. Процесс установки

При работе в Windows NT/2000/XP необходимо зарегистрироваться в системе с привилегиями администратора, разархивировать дистрибутив во временный каталог, после чего запустить файл setup.exe. В результате появится окно мастера установки, показанное на рис. П2.1.

> ***Замечание***
>
> Перед установкой сервера MySQL рекомендуется отключить FireWall, если он имеется в системе, и включить его только после того, как сервер MySQL успешно установлен. Если после этого сервер MySQL прекратит принимать запросы, в FireWall следует открыть порт 3306 (или другой порт, если значение порта будет изменено во время установки).

> ***Замечание***
>
> Установка дистрибутива MySQL описывается на примере полного дистрибутива Windows (*x*86).

Для продолжения установки следует нажать кнопку **Next**, после чего откроется окно, показанное на рис. П2.2, в котором предлагается выбрать способ установки:

- **Typical** (Стандартный);
- **Complete** (Полный);
- **Custom** (Выборочный).

В первом случае устанавливаются стандартные компоненты, во втором — все возможные компоненты, в третьем — по выбору пользователя.



Рис. П2.1. Стартовое окно автоматического установщика



Рис. П2.2. Выбор режима установки MySQL

В режиме **Custom** нажмите кнопку **Next** и в появившемся окне (рис. П2.3) выберите необходимые компоненты. Компоненты, которые по умолчанию отключены, перечеркнуты красным крестиком. Для того чтобы установить предлагаемые компоненты, необходимо выделить нужный компонент и выбрать в выпадающем меню пункт **This feature will be installed on local hard drive**. Если у вас отсутствует опыт установки MySQL, при первой установке лучше ничего не менять.

После того как все компоненты выбраны, в диалоговом окне, показанном на рис. П2.3, можно изменить каталог установки, нажав кнопку **Change**.

ЗАМЕЧАНИЕ

По умолчанию установка производится в каталог C:\Program Files\MySQL\MySQL Server 5.0\.

Рекомендуется изменить путь по умолчанию на более короткий, например на C:\mysql5\. Это необходимо для более комфортной работы с утилитами MySQL в командной строке. Нажав кнопку **Change**, введите путь C:\mysql5\ (рис. П2.4).

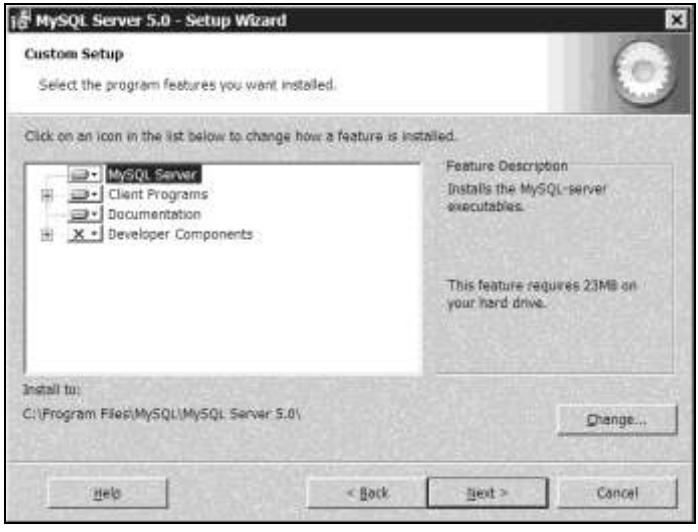


Рис. П2.3. Выбор компонентов для установки

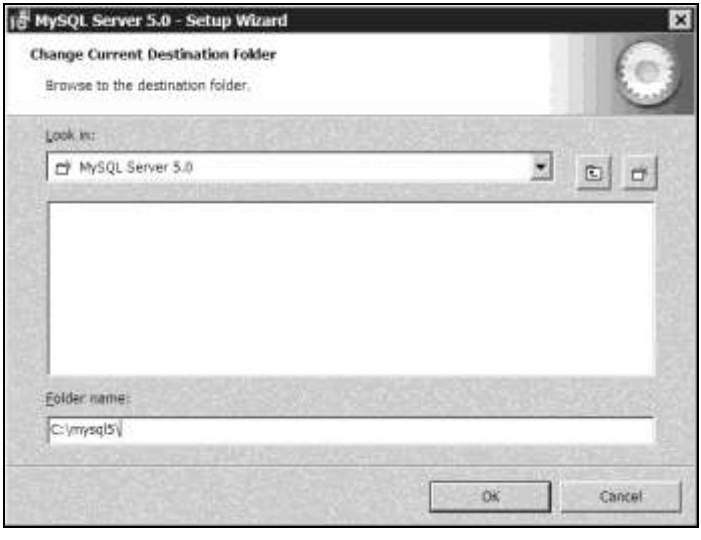


Рис. П2.4. Выбор каталога, в который будет установлен MySQL-сервер

Если вы устанавливаете MySQL поверх старой копии, например, MySQL 4.1, 4.0 или 3.23 — будьте осторожны. Такой способ вполне допустим и часто применяется для того, чтобы базы данных старой версии перешли в новую инсталляцию. Но следует помнить, что системная база данных `mysql` в настоящий момент несовместима с предыдущими версиями. Поэтому если вы производите обновление MySQL, помимо остановки сервера следует удалить каталог C:\mysql\data\mysql системной базы данных `mysql`. При первой инсталляции проблем, связанных с несовместимостью, возникать не должно.

После завершения предынсталляционной настройки выводится окно, представленное на рис. П2.5. Нажатие кнопки **Install** начинает процесс копирования фалов (рис. П2.6).

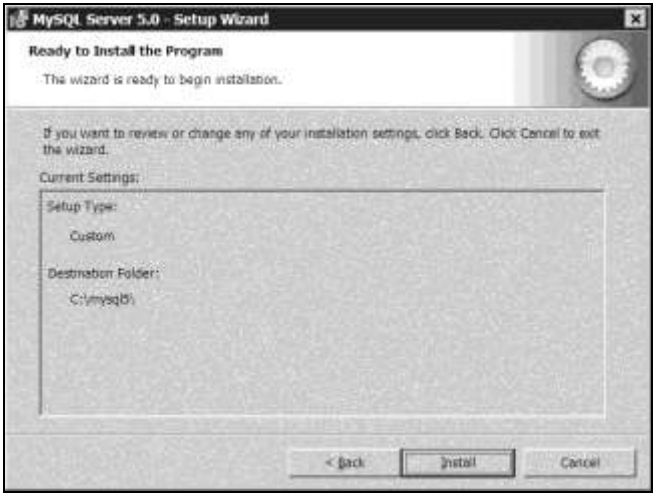


Рис. П2.5. Завершающее окно перед процессом инсталляции

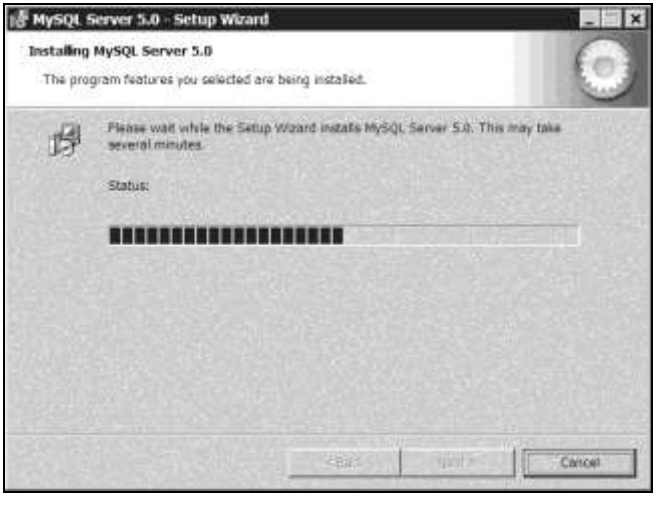


Рис. П2.6. Копирование файлов

После копирования файлов мастер установки предлагает зарегистрироваться на сайте **http://www.mysql.com**. Для того чтобы пропустить эту процедуру, выберите пункт **Skip Sign-Up** (рис. П2.7). Следующее окно (рис. П2.8) сообщает о завершении основного этапа установки MySQL.

Если в данном окне не отключать возможность немедленной настройки сервера, то после нажатия кнопки **Finish** запустится утилита MySQL Server Instance Config Wizard, которая подробно рассматривается в следующем разделе.



Рис. П2.7. Предложение зарегистрироваться на сайте **http://www.mysql.com**



Рис. П2.8. Завершение основного этапа установки MySQL

П2.1.2. Постинсталляционная настройка

Сразу после установки MySQL запускается утилита постинсталляционной настройки. Данный этап можно пропустить, учитывая, что к настройке всегда можно вернуться, выбрав пункт системного меню **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Server Instance Config Wizard**. Тем не менее, рекомендуется сразу произвести настройку системы.

Настройка начинается со стартового окна (рис. П2.9). После нажатия кнопки **Next** открывается окно (рис. П2.10), в котором предлагается выбрать режим настройки:

- **Detailed Configuration** (Подробный);
- **Standard Configuration** (Стандартный).

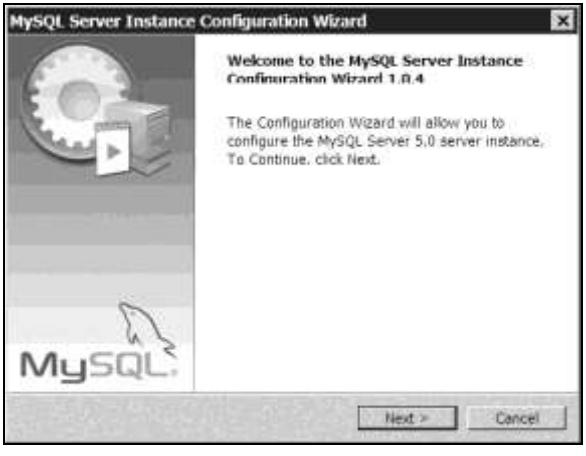


Рис. П2.9. Настройка MySQL при помощи мастера MySQL Server Instance Configuration Wizard



Рис. П2.10. Выбор режима настройки

Для более гибкой настройки системы следует выбрать первый пункт (**Detailed Configuration**). После нажатия кнопки **Next** открывается окно настройки производительности MySQL (рис. П2.11) со следующими опциями:

- **Developer Machine** (Машина разработчика);
- **Server Machine** (Сервер);
- **Dedicated MySQL Server Machine** (Выделенный сервер).



Рис. П2.11. Настройка производительности сервера MySQL

Все три опции различаются по интенсивности использования процессора, объему требуемой оперативной памяти и жесткого диска. Следует выбрать первый пункт, т. к. в этом случае MySQL занимает наименьший объем оперативной памяти, не мешая работе других приложений. Второй пункт предназначен для сервера, на котором, помимо MySQL, будут работать другие серверы, например, Web-сервер или транспортный почтовый агент. Третий пункт предназначен для выделенного сервера MySQL, на котором не будут выполняться никакие другие приложения.

Окно, представленное на рис. П2.12, позволяет выбрать предпочтительный тип для таблиц, который назначается по умолчанию. Следует оставить первый пункт.

> ***Замечание***
>
> Результатом работы утилиты MySQL Server Instance Configuration Wizard является конфигурационный файл C:\mysql5\my.ini, который позже можно отредактировать вручную.

В окне, показанном на рис. П2.12, предлагается выбор, по сути, между двумя типами таблиц: MyISAM и InnoDB. Главное различие заключается в том, что под каждую таблицу MyISAM создается отдельный файл, который хранится в соответствующей базе данных каталога, а все таблицы InnoDB хранятся в едином табличном пространстве, под которое выделяется один файл. Кроме этого, скорость работы с MyISAM в несколько раз превышает скорость работы с таблицами типа InnoDB. Однако таблицы InnoDB поддерживают транзакции на уровне строк, более надежны и позволяют работать

с таблицами большого объема. При первом знакомстве рекомендуется выбрать более простые в обслуживании и настройке таблицы типа MyISAM.

Выбор диска для хранения этого файла и пути, куда он будет помещен, осуществляется в следующем окне, представленном на рис. П2.13.



Рис. П2.12. Выбор типа таблиц по умолчанию



Рис. П2.13. Выбор пути для хранения файла под таблицы InnoDB

Следующее окно (рис. П2.14) предлагает указать максимальное количество клиентов, которые смогут одновременно подключаться к серверу. Первый пункт (рекомендуется выбрать именно его) предполагает, что количество таких соединений не будет превышать 20, второй пункт допускает 500 соединений с сервером, третий позволяет назначить собственное ограничение количества активных соединений.



Рис. П2.14. Выбор максимального количества активных соединений с сервером

На реальных серверах не следует устанавливать это число более 200, обычно если оперативной памяти выделено достаточно и производительность процессора (процессоров) высока, запросы выполняются очень быстро, не достигая этого предела. Как только один из компонентов сервера начинает потреблять лишние ресурсы, все остальные процессы ожидают выполнения и накапливаются в очереди. Так как в памяти накапливается множество невыполненных запросов, ситуация еще больше усугубляется. Ограничение количества одновременных запросов позволяет при такой критической ситуации отбрасывать все новые запросы. В результате после того, как пиковая нагрузка минует, сервер выполняет все накопившиеся в очереди запросы, и ситуация нормализуется. При отсутствии такого ограничения накапливающиеся запросы приведут к зависанию сервера и необходимости ручного вмешательства для его перезапуска.

Замечание

Если ситуация с превышением максимального количества одновременных соединений воспроизводится регулярно и вызвана не утечками памяти или перегрузкой процессора, следует увеличить ограничение на количество одновременно выполняющихся запросов.

В следующем окне (рис. П2.15) устанавливается номер порта, по которому будет происходить соединение клиентов с MySQL-сервером (по умолчанию — 3306). Если на компьютере не имеется других версий MySQL, рекомендуется сохранить значение по умолчанию, т. к. порт 3306 является стандартным для MySQL.

Примечание

В том случае, если необходимо запускать MySQL 5 совместно с другими версиями MySQL, можно выбрать другой номер порта, отличный от тех, по которым происходит обращение к прочим MySQL-серверам.

В следующем окне предлагается указать кодировку по умолчанию (рис. П2.16). Необходимо выбрать третий пункт (ручной выбор кодировки) и в выпадающем списке выделить тип **cp1251**, соответствующий русской Windows-кодировке.

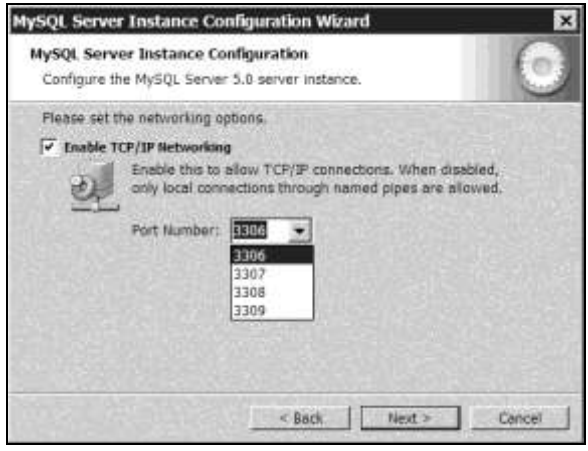


Рис. П2.15. Выбор порта, по которому сервер MySQL будет слушать клиентские запросы



Рис. П2.16. Выбор кодировки на MySQL-сервере по умолчанию

При работе в среде Windows NT/2000/XP можно установить MySQL в качестве сервиса, что обеспечит запуск сервера mysqld.exe при старте системы и корректное завершение работы сервера при выключении компьютера.

Окно, представленное на рис. П2.17, предназначено для настройки такого сервиса. Установка флажка **Install As Windows Service** позволяет установить сервис с именем, которое следует выбрать в выпадающем списке **Service Name**. Можно изменить имя сервиса, особенно если в системе присутствует инсталлированный MySQL-сервер более ранней версии — это позволит избежать конфликтов при запуске серверов как MySQL 5, так и более ранней версии. Отметка флажка **Launch the MySQL Server automatic** позволяет настроить сервис на автоматический режим работы, когда запуск сервера производится со стартом системы, а остановка — при завершении работы с системой. В противном случае запуск и остановку сервера придется выполнять вруч-

ную. Как будет показано далее, можно изменить режим работы сервиса в любой момент в консоли управления сервисами.

Флажок **Include Bin Directory in Windows PATH** позволяет прописать путь к каталогу C:\mysql5\bin в системной переменной `PATH`, что может быть удобным при частом использовании утилит из этого каталога.



Рис. П2.17. Настройка сервиса для автоматического запуска MySQL-сервера



Рис. П2.18. Настройка учетных записей

В следующем окне (рис. П2.18) производится настройка учетных записей. Если вы не знакомы с системой авторизации MySQL и осуществляете установку первый раз, рекомендуется снять флажок **Modify Security Settings** и оставить данные настройки по умолчанию. Два текстовых поля позволяют задать пароль для суперпользователя `root` (в противном случае в качестве пароля будет выступать пустая строка). Флажок **Create**

An Anonymous Account позволяет создать анонимного пользователя. Такой пользователь обладает минимальными правами, а в качестве имени и пароля у него выступает пустая строка.

Последняя страница утилиты настройки MySQL-сервера MySQL Server Instance Configuration Wizard представлена на рис. П2.19.



Рис. П2.19. Последняя страница утилиты MySQL Server Instance Configuration Wizard

После нажатия кнопки **Execute** будет создан конфигурационный файл C:\mysql5\my.ini и запущен сервер MySQL.

> ***Замечание***
>
> Если при установке сервера MySQL у вас возникли затруднения, вы можете обратиться на форум по адресу **http://www.softtime.ru/forum/index.php?id_forum=3**. Авторы присутствуют на форуме каждый день и помогают читателям с установкой MySQL, Web-сервера Apache и PHP уже на протяжении двух лет. На данном форуме можно также задавать любые вопросы, связанные как с администрированием, так и с программированием в среде MySQL.

П2.1.3. Проверка работоспособности MySQL

После того как установка и конфигурирование MySQL завершены, необходимо убедиться в работоспособности сервера MySQL. Для этого следует открыть окно для работы с командной строкой, выбрав в системном меню **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Command Line Client**. Открывшееся окно может выглядеть так, как это показано на рис. П2.20. На предложение ввести пароль нажмите клавишу <Enter>. После того как появилось приглашение `mysql>`, введите команду:

```
SELECT VERSION();
```

В результате сервер должен показать текущую версию сервера, в нашем случае это `5.0.26-community-nt` (рис. П2.21).



Рис. П2.20. Командная строка утилиты MySQL Command Line Client

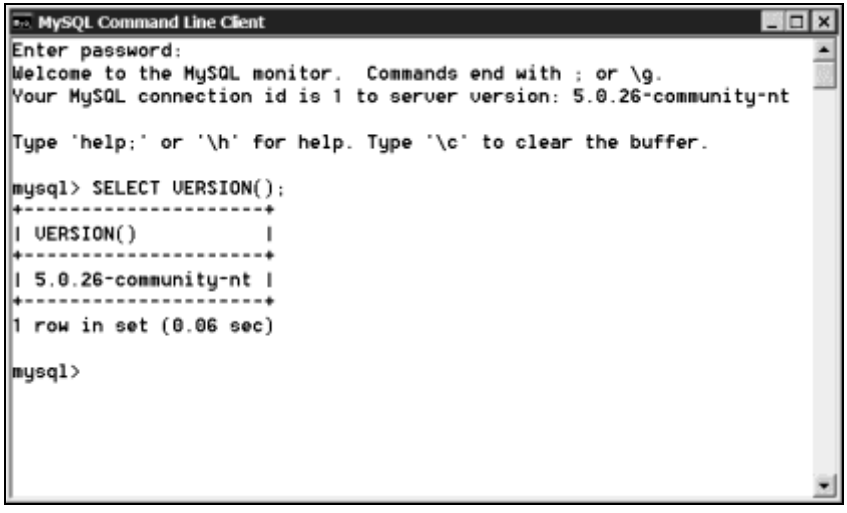


Рис. П2.21. Результат выполнения команды `SELECT VERSION()`

Замечание

Для выхода введите команду `EXIT`.

Замечание

Возможно, при обращении к утилите MySQL Command Line Client она издаст писк и закроется. В этом случае следует отредактировать файл my.ini таким образом, чтобы директива `default-character-set` присутствовала только в секции `[mysqld]` и отсутствовала в секции `[mysql]` и всех других секциях.

Если сервер не запущен, то после нажатия клавиши <Enter> окно будет закрыто. Для запуска сервера следует перейти в консоль управления сервисами, выполнив команду **Пуск | Настройка | Панель управления | Администрирование | Службы**. В результате этого откроется окно, представленное на рис. П2.22.

Замечание

Проверить, запущен MySQL-сервер или нет, можно также по наличию процесса mysql-nt.exe в Диспетчере задач.

В консоли управления сервисами необходимо найти сервис MySQL. Если поле **Состояние** данного сервиса пусто, то он не запущен. Для его запуска следует выбрать в контекстном меню пункт **Пуск**. Для остановки сервиса необходимо выбрать пункт **Стоп**. Обратите внимание на столбец **Тип запуска**: значение **Авто** сообщает Windows

о необходимости запускать сервис при старте операционной системы, значение **Вручную** означает, что запуск выполняется пользователем через консоль управления сервисами. Можно изменить режим запуска сервиса, выбрав в контекстном меню сервиса пункт **Свойства**.

Замечание

Для того чтобы запустить MySQL-сервер в обход сервисов, необходимо воспользоваться параметром `--standalone`.

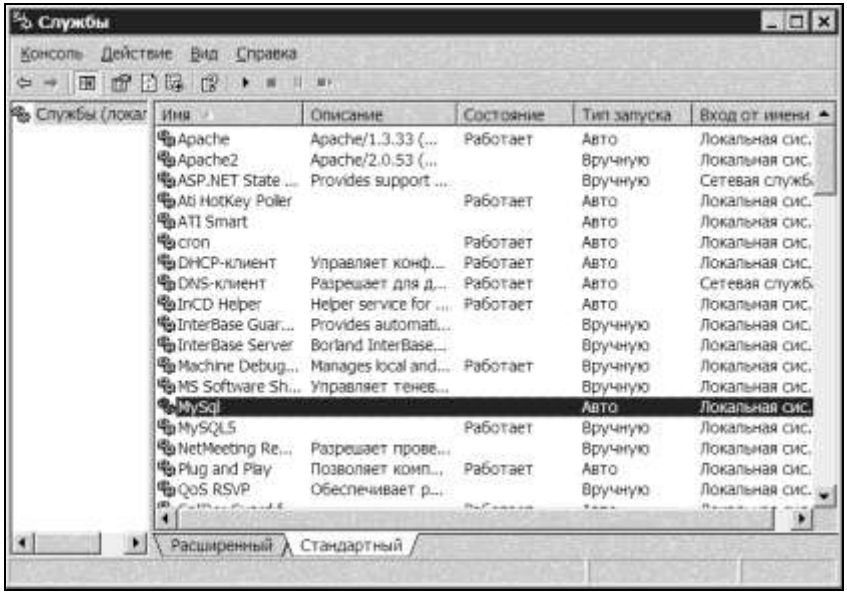


Рис. П2.22. Консоль управления сервисами

П2.2. Установка MySQL под Linux

Большинство дистрибутивов Linux содержит в своем составе СУБД MySQL. В данном разделе будет рассмотрена установка сервера и клиента применительно к Red Hat Linux (Fedora Core).

Несмотря на то что в среде Linux принято устанавливать программное обеспечение из исходных кодов, а MySQL, как база данных с открытым кодом, позволяет это сделать, официально рекомендуется ставить MySQL из бинарных пакетов, т. к. они оптимизированы с учетом использования конкретной архитектуры для достижения максимальной эффективности.

Замечание

Перед установкой MySQL необходимо остановить сервер, если он запущен. Для этого в системах, основанных на дистрибутиве Rad Hat, как правило, требуется остановить демон mysql, выполнив команду из-под `root`: `/sbin/service mysql stop`.

Чтобы просмотреть все файлы в пакете RPM, выполните команду:

```
rpm -qpl MySQL-VERSION.i386.rpm
```

Для того чтобы выяснить требования пакета, необходимо выполнить команду:

```
rpm -qRp MySQL-VERSION.i386.rpm
```

Для выполнения стандартной установки выполните команды, предварительно войдя в систему как `root`:

```
rpm -ihv MySQL-server-VERSION.i386.rpm
rpm -Uhv MySQL-client-VERSION.i386.rpm
```

> ***ЗАМЕЧАНИЕ***
>
> Если пакет не ставится из-за неудовлетворенных зависимостей, можно потребовать игнорировать зависимости при помощи параметра `--nodeps`. Часть зависимостей необходимо удовлетворять обязательно, часть — нет: так модуль DBI для Perl требуется только для запуска тестовых скриптов, а без соответствующей версии библиотеки glibc MySQL не заработает.

Дистрибутивы Red Hat и Fedora Core до версии 3 поставлялись с MySQL версии 3.23.58. Это связано с тем, что большое число приложений ориентировалось именно на эту версию, и ее удаление приводило к нарушению зависимостей. Поэтому для установки более новых версий необходимо дополнительно ставить пакет MySQL-shared-compact, который включает в себя две динамические библиотеки для обратной совместимости: libmysqlclint.so.12 для MySQL 4.0 и libmysqlclient.so.10 для MySQL 3.23. В Fedora Core 4 этот пакет входит в состав дистрибутива, для более ранних версий необходимо загрузить пакет MySQL-shared-compat-VERSION.i386.rpm с официального сайта MySQL.

На загрузочной странице **http://dev.mysql.com/downloads/mysql/5.0.html** для дистрибутива, основанного на Red Hat, лучше ориентироваться на раздел **Linux x86 generic RPM (statically linked against glibc 2.2.5) downloads**. В отличие от Windows-раздела, MySQL поставляется в виде отдельных компонентов, а не в виде единого rpm-пакета. Для установки сервера потребуется пакет MySQL-server-VERSION.i386.rpm из раздела **Server**. Для установки клиентского программного обеспечения нужен пакет MySQL-client-VERSION.i386.rpm из раздела **Client programs**. По желанию могут быть установлены тесты производительности **Benchmark/test suites**, библиотеки и заголовочные файлы для разработки программ **Libraries and header files**, динамические библиотеки клиентского ПО для поддержки клиентов MySQL — **Dynamic client libraries**. Раздел **Dynamic client libraries (including 3.23.x libraries)** позволяет загрузить упоминавшуюся ранее библиотеку MySQL-shared-compat-VERSION.i386.rpm, предназначенную для обратной совместимости.

RPM помещает данные в каталог /var/lib/mysql/ и создает соответствующие вхождения в каталоге /etc/init.d/ для автоматического запуска сервера при загрузке.

> ***ЗАМЕЧАНИЕ***
>
> После установки основных файлов автоматически запускается скрипт mysql_intall_db, который разворачивает системные базы данных. Если он в силу каких-либо причин не срабатывает (например, из-за неправильно сконфигурированного файла /etc/my.cnf), его можно запустить позже.

При обновлении уже существующей версии MySQL можно воспользоваться командами:

```
rpm -Uhv MySQL-server-VERSION.i386.rpm
rpm -Uhv MySQL-client-VERSION.i386.rpm
```

После этого можно запустить сервер при помощи команды `/usr/bin/mysqld_safe`. Для проверки работоспособности далее следует набрать команду:

```
mysql -u root
```

в ответ на которую должно открыться приглашение вида:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18 to server version: 5.0.18-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

При помощи команды `SELECT VERSION()` можно запросить версию сервера, как это описывалось в разделе, посвященном установке MySQL под Windows.

> ***Замечание***
>
> Более подробно утилита mysql описывается в следующей главе. Здесь стоит лишь упомянуть, что покинуть утилиту mysql можно не только при помощи команды `EXIT`, но и команды `QUIT`.

После установки новой версии инсталлятор перезаписывает скрипт автоматического запуска и остановки сервера /etc/rc.d/init.d/mysql сценарием mysql.server из дистрибутива, в результате чего он может не работать, как это, например, происходит в системе Fedora Core. Для того чтобы восстановить его работу, необходимо переписать функцию `parse_server_arguments()`, указав пути к каталогу данных и серверу, как это представлено в листинге П2.1.

> ***Замечание***
>
> Если при установке скрипт mysql.server не был развернут в каталог /etc/rc.d/init.d/, его можно найти в каталоге /usr/share/mysql/.

Листинг П2.1. Функция `parse_server_arguments()` скрипта /etc/rc.d/init.d/mysql

```
parse_server_arguments() {
  for arg do
    case "$arg" in
      --basedir=*)  basedir=`echo "$arg" | sed -e 's/^[^=]*=//'`
                    bindir="/usr/bin"
                    datadir="/var/lib/mysql"
                    sbindir="/usr/sbin"
                    libexecdir="$basedir/libexec"
        ;;
      --datadir=*)  datadir=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
```

```
      --user=*)  user=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
      --pid-file=*) server_pid_file=`echo "$arg" | sed -e 's/^[^=]*=//'` ;;
      --use-mysqld_safe) use_mysqld_safe=1;;
      --use-manager)     use_mysqld_safe=0;;
    esac
  done
}
```

Скрипт /etc/rc.d/init.d/mysql в первую очередь предназначен для автоматического запуска и остановки сервера MySQL при старте и завершении работы операционной системы. Однако он вполне допускает старт и остановку сервера в ручном режиме. Для запуска mysqld необходимо войти в систему с правами суперпользователя и выполнить команду:

```
/etc/rc.d/init.d/mysql start
```

Следующая команда выполнит остановку сервера:

```
/etc/rc.d/init.d/mysql start
```

Помимо этого можно осуществить перезапуск сервера (это особенно полезно при изменении параметров конфигурационного файла my.ini):

```
/etc/rc.d/init.d/mysql restart
```

Для того чтобы проверить, запущен MySQL-сервер в UNIX-среде или нет, можно использовать команду:

```
ps -Af | grep mysql
```

Если сервер запущен, команда выдаст строки, соответствующие процессу mysqld.

П2.3. Конфигурационный файл

Сервер MySQL имеет переменные состояния, которые определяют его настройки. Некоторые из переменных состояния могут быть изменены при помощи параметров запуска. При просмотре команды запуска сервера MySQL в свойствах сервиса или при помощи команды `ps -Af` можно заметить, что им передаются параметры запуска. *Параметром* называют ключевое слово, которое следует после имени процесса. Например, параметр `--defaults-file` в листинге П2.2 сообщает MySQL-серверу, что конфигурационный файл my.ini следует искать по адресу C:\mysql5\my.ini.

> ***Замечание***
>
> В Windows, если не указывается параметр `--defaults-file`, конфигурационный файл ищется в корне диска C:, в UNIX-подобных операционных системах конфигурационные файлы традиционно помещаются в каталог /etc, поэтому путем по умолчанию будет /etc/my.cnf.

Листинг П2.2. Параметр запуска `--defaults-file`

```
C:\mysql5\bin\mysqld-nt.exe --defaults-file="C:\mysql5\my.ini"
```

Параметров сервера MySQL достаточно много, поэтому их может накапливаться изрядное количество. Просмотреть их полный список можно, передав серверу MySQL параметры `--verbose --help` (до версии MySQL 4.1 можно указать только параметр `--help`). Некоторые параметры имеют короткие сокращения и начинаются с одного дефиса, а не с двух. Например, параметр `--user` имеет сокращение `-u`.

Параметры запуска можно не прописывать непосредственно в строке запуска сервера, а поместить в конфигурационный файл my.ini или my.cnf, из которого сервер прочитает их при старте.

Следует отметить, что в конфигурационном файле my.ini параметры указываются без предваряющих дефисов, т. к. каждый параметр располагается на отдельной строке. Если строка начинается с символа диеза `#` или точки с запятой `;`, то строка является комментарием. Параметры в конфигурационном файле называют *директивами*.

В операционной системе Windows конфигурационный файл может иметь как расширение ini (такой файл может располагаться в каталогах C:\Windows\ и в C:\mysql5\), так и расширение cnf (обычно располагается в корне диска C:).

В UNIX-подобных операционных системах конфигурационный файл имеет, как правило, расширение cnf.

Конфигурационный файл влияет на работу не только сервера MySQL, но и вспомогательных утилит, таких как консольный клиент mysql, утилита создания SQL-дампов mysqldump и т. п., более того, один конфигурационный файл может управлять работой нескольких серверов MySQL. Поэтому содержимое конфигурационного файла разделено на секции, которые имеют вид `[`*`имя_секции`*`]`. Имя секции определяет утилиту или сервер, к которым будут относиться перечисленные далее директивы, до тех пор, пока не встретится новая секция или конец файла. В табл. П2.1 перечислены наиболее типичные секции.

Таблица П2.1. *Секции конфигурационного файла my.ini или my.cnf*

Имя секции	Описание
`[mysqld]`	Сервер MySQL
`[server]`	Сервер MySQL
`[mysqld-4.0]`	Сервер MySQL версии 4.0
`[mysqld-4.1]`	Сервер MySQL версии 4.1
`[mysqld-5.0]`	Сервер MySQL версии 5.0
`[mysqld-5.1]`	Сервер MySQL версии 5.1
`[mysqld_safe]`	Утилита запуска `mysqld_safe`
`[safe_mysqld]`	Утилита запуска `mysqld_safe`
`[mysql.server]`	Скрипт запуска `mysql.server`
`[mysqld_multi]`	Утилита запуска `mysqld_multi`
`[client]`	Любая клиентская утилита, обращающаяся к серверу

Таблица П2.1 *(окончание)*

Имя секции	Описание
[mysql]	Консольный клиент mysql
[mysqldump]	Утилита создания SQL-дампов mysqldump
[mysqlhotcopy]	Утилита горячего копирования бинарных файлов базы данных

Наличие секции [mysqld] и специальных секций для разных версий обусловлено тем, что с каждой новой версией появляется все больше и больше параметров запуска, и если конфигурационный файл управляет несколькими серверами, то некоторые директивы будут одинаковыми для всех серверов, а другие уникальны для каждой из версий.

Точно такая же ситуация сложилась с секцией [client] и секциями для каждой из утилит. Дело в том, что все утилиты обладают сходными параметрами (например, параметры соединения с серверами у всех одинаковые), и в то же время каждая из них имеет уникальные параметры, характерные только для ее функциональных возможностей.

В листинге П2.3 приводится пример конфигурационного файла my.ini, характерного для операционной системы Windows.

Листинг П2.3. Типичный конфигурационный файл my.ini

```
# Секция MySQL-сервера
[mysqld]
# Порт TCP/IP, который прослушивает MySQL-сервер, порт 3306 является
# стандартным, однако можно назначить любой другой не занятый порт
port=3306

# Путь установки MySQL-сервера, все остальные пути,
# если не указано другое, вычисляются относительно этого пути
basedir="C:/mysql5/"

# Путь к каталогу данных, в случае данного конфигурационного файла
# эту директиву можно не указывать, т. к. путь будет правильно
# вычислен относительно директивы basedir
datadir="C:/mysql5/Data/"

# В качестве кодировки по умолчанию для новых таблиц и баз данных
# будет выступать русская кодировка Windows-1251,
# в случае отсутствия этой директивы по умолчанию будет назначена
# кодировка latin1 (русский текст будет сортироваться неправильно)
default-character-set=cp1251

# Объем оперативной памяти, которая отводится на кэш ключей
```

```
key_buffer_size = 128M
# Максимальный размер запроса, который посылает клиент серверу
max_allowed_packet = 8M
# Объем оперативной памяти, которая отводится на кэш запросов
query_cache_size = 64M
# Максимальное количество одновременных подключений
max_connections = 200

# Секция консольного клиента mysql
[mysql]
# Пользователь по умолчанию, если клиенту mysql через параметр -u
# не будет передан текущий пользователь, будет использован root
user = root
```

В конфигурационном файле my.ini, приведенном в листинге П2.3, представлена лишь небольшая часть директив. Полный список можно найти в официальной документации.

П2.4. Утилита mysql

Консольный клиент mysql часто называют "терминальным монитором" или просто "монитором". Пользователи операционной системы Linux могут набирать имя данной программы с параметрами из любой точки системы. Владельцам Windows для запуска mysql необходимо выбрать следующий пункт меню: **Пуск | Программы | MySQL | MySQL Server 5.0 | MySQL Command Line Client**.

> ***Замечание***
>
> Возможно, при обращении к утилите MySQL Command Line Client она издаст писк и закроется. В этом случае следует отредактировать файл my.ini таким образом, чтобы директива `default-character-set` присутствовала только в секции `[mysqld]` и отсутствовала в секции `[mysql]` и всех других секциях.

П2.4.1. Командная строка

При запуске утилиты mysql с помощью MySQL Command Line Client нет возможности вводить какие-либо параметры. Кроме того, для работы с другими утилитами, входящими в поставку MySQL, необходим доступ к командной строке. Получить его можно, обратившись к следующему пункту: **Пуск | Программы | Стандартные | Командная строка**. В результате будет открыто окно, представленное на рис. П2.23.

> ***Замечание***
>
> В данной книге цвет окна и его размеры выбраны отличными от тех, что выставляются системой по умолчанию. Можно самостоятельно изменить цветовую схему командной строки и ее размеры в свойствах системного меню окна.

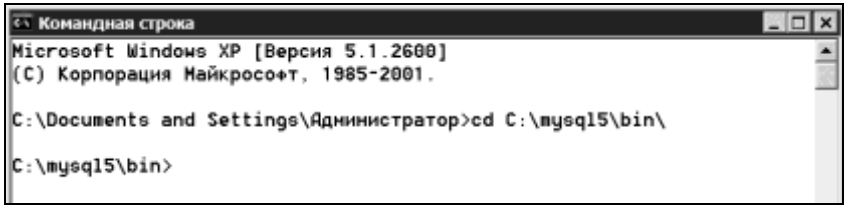


Рис. П2.23. Командная строка

После запуска командной строки необходимо перейти в подкаталог bin корневого каталога MySQL. Для этого следует набрать команду:

```
cd C:\mysql5\bin
```

В ответ будет выведено приглашение командной строки:

```
C:\mysql5\bin>
```

> ***Замечание***
>
> Если корневой каталог MySQL расположен на другом диске, например, D:, необходимо перед выполнением команды `cd` сменить диск при помощи команды `D:`. Кроме этого, создав ярлык для командной строки, в свойствах ярлыка можно выставить в качестве рабочего каталога путь к каталогу bin, для того чтобы не вводить всякий раз команду `cd`.

Теперь мы находимся в каталоге bin и можем запускать расположенные в нем утилиты. Для этого достаточно набрать в командной строке имя утилиты и, если необходимо, параметры. Параметры — это символы, начинающиеся с дефиса, например, `-u`, за которыми следует их значение. Использование различных параметров, которые будут рассмотрены в данной главе, позволяет изменять режим работы утилит.

Каждый раз набирать команды перехода в каталог bin достаточно утомительно, особенно, если сервер MySQL был установлен в каталог C:\Program Files. Имеется несколько возможностей для автоматизации загрузки консольного клиента mysql. Первая из них заключается в создании ярлыка командной строки на рабочем столе. Далее в контекстном меню ярлыка следует выбрать пункт **Свойства**, а в открывшемся диалоге вкладку **Ярлык**. В поле **Рабочая папка** следует ввести путь к каталогу bin (на рис. П2.24 это путь C:/mysql5/bin), после чего нажать кнопку **OK** или **Применить**.

Теперь при вызове командной строки при помощи ярлыка в качестве текущего каталога изначально будет выставлен каталог C:\mysql5\bin.

Второй путь автоматизации процесса запуска утилиты mysql заключается в том, чтобы прописать каталог C:\mysql5\bin в переменной окружения `PATH`. Это позволит запускать утилиты из каталога bin из любого другого каталога компьютера.

> ***Замечание***
>
> В UNIX-подобных операционных системах все утилиты помещаются либо в каталог /bin, либо в каталог /usr/sbin, которые указаны в переменной окружения `PATH` по умолчанию. Если по какой-то причине утилиты MySQL расположены в других каталогах, следует либо указывать полный путь к ним, либо прописать каталог в переменной окружения `PATH`.

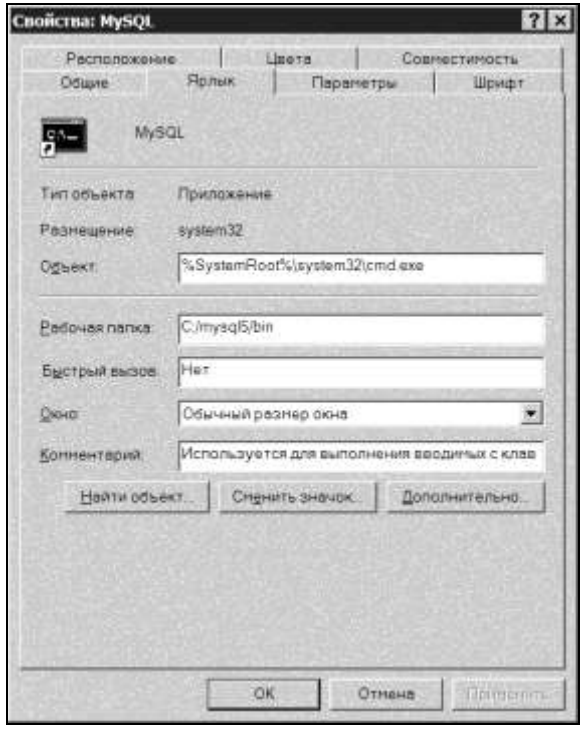


Рис. П2.24. Свойства ярлыка для командной строки

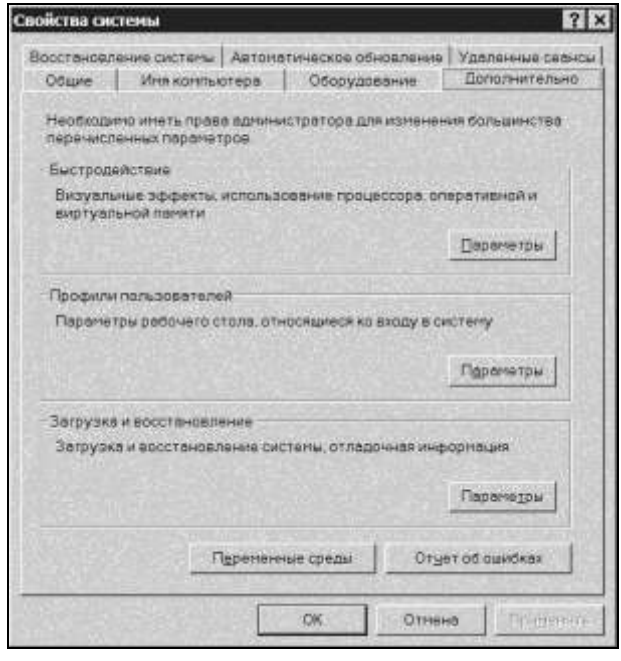


Рис. П2.25. Диалоговое окно **Свойства системы**

Переменная окружения — это параметр, который позволяет настроить поведение операционной системы: где искать в первую очередь исполняемые файлы, где хранить временные файлы, где расположен системный каталог и т. п. На рис. П2.24 в поле **Объект** используется конструкция %SystemRoot% — это один из примеров переменной окружения, которая хранит путь к каталогу Windows. Отдельные программы могут создавать собственные переменные окружения, однако некоторые из переменных окружения являются стандартными. К стандартным переменным окружения относится переменная `PATH`, которая сообщает операционной системе, в каких каталогах следует искать исполняемые файлы.

В операционной системе Windows XP для доступа к переменным окружения следует открыть диалог **Свойства системы**, выбрав в контекстом меню значка **Мой компьютер** пункт **Свойства** или выбрать следующий пункт системного меню: **Пуск | Настройка | Панель управления | Система**. В результате будет открыто диалоговое окно, изображенное на рис. П2.25.

На вкладке **Дополнительно** необходимо нажать кнопку **Переменные среды**, что приведет к открытию диалогового окна, представленного на рис. П2.26.

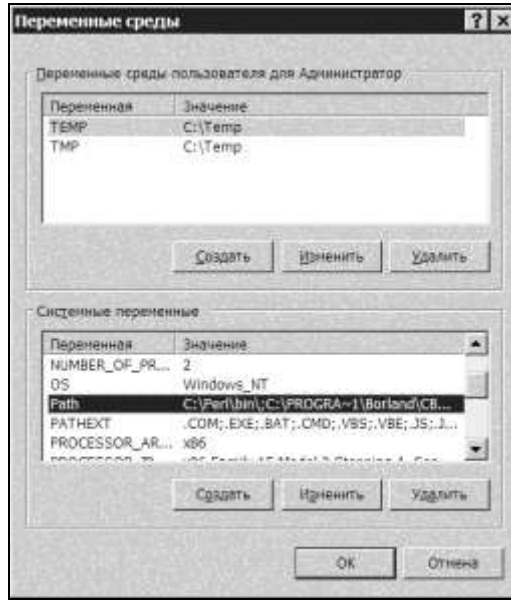


Рис. П2.26. Диалоговое окно **Переменные среды**

В разделе **Системные переменные** следует отыскать переменную окружения `PATH` и дополнить ее путем к каталогу bin СУБД MySQL. Отдельные пути в значении переменной `PATH` разделяются точкой с запятой (в конце всей строки точка с запятой не требуется). Новое значение переменной окружения `PATH` вступает в силу после перезагрузки компьютера.

Еще одним важным параметром командной строки является кодировка. По умолчанию набранный в консоли русский текст будет помещен в базу данных MySQL в кодировке

cp866 (DOS), что может быть очень не удобным. Для смены кодировки в консоли используется команда `chcp`, запуск которой без параметров сообщает текущую кодировку, передача команде в качестве параметра номера кодировки приводит к смене кодировки консоли. Например, для того чтобы установить кодировку Windows-1251, следует выполнить команду:

```
chcp 1251
```

> ***Замечание***
>
> Если в результате выполнения команды `chcp 1251` набираемый русский текст отображается в искаженном виде, следует в свойствах окна консоли поменять шрифт с точечного на Lucida Console.

П2.4.2. Работа с утилитой mysql в диалоговом режиме

Для соединения с сервером базы данных в параметрах mysql необходимо указать имя пользователя и его пароль. В только что установленной системе существует один пользователь `root`, наделенный правами администратора, а его паролем по умолчанию выступает пустая строка. Поэтому для получения доступа к серверу достаточно набрать команду:

```
mysql –u root
```

Результат выполнения команды представлен на рис. П2.27. После номера версии сервера и информации о справочных ключах выводится приглашение `mysql>`, разрешающее ввод команд.

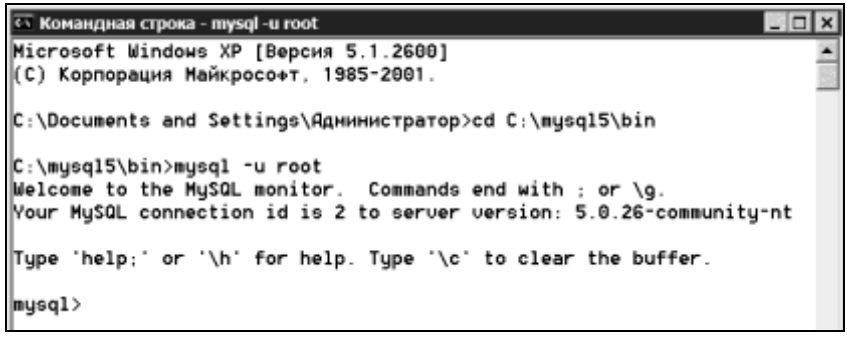


Рис. П2.27. Приглашение `mysql>` консольного клиента mysql

Для демонстрации полноценной авторизации создадим нового пользователя `user` с паролем `hello` при помощи оператора `CREATE USER`, представленного в листинге П2.4. Эту инструкцию следует набрать сразу после приглашения `mysql>`, как это показано на рис. П2.28.

Листинг П2.4. Создание нового пользователя `user`

```
CREATE USER user IDENTIFIED BY 'hello';
```

После этого выйдем из оболочки mysql и попробуем зайти от имени нового пользователя. Выход из оболочки mysql производится при помощи команды `exit` или `quit` (рис. П2.28).

Теперь команда `mysql -u user` не проходит — сервер отказывается предоставить доступ без подтверждения полномочий паролем (рис. П2.29). Для того чтобы соединиться с сервером с правами пользователя `user`, необходимо передать пароль. Для этого используется параметр `-p`, сразу после которого без пробела вводится пароль. Для удобства в других параметрах допускается использование пробела между параметром и его значением. Например, следующие записи эквивалентны:

```
mysql -uuser -phello
mysql -u user -phello
```

Параметр `-p` является исключением, т. к. все следующие за ним символы воспринимаются как часть пароля.

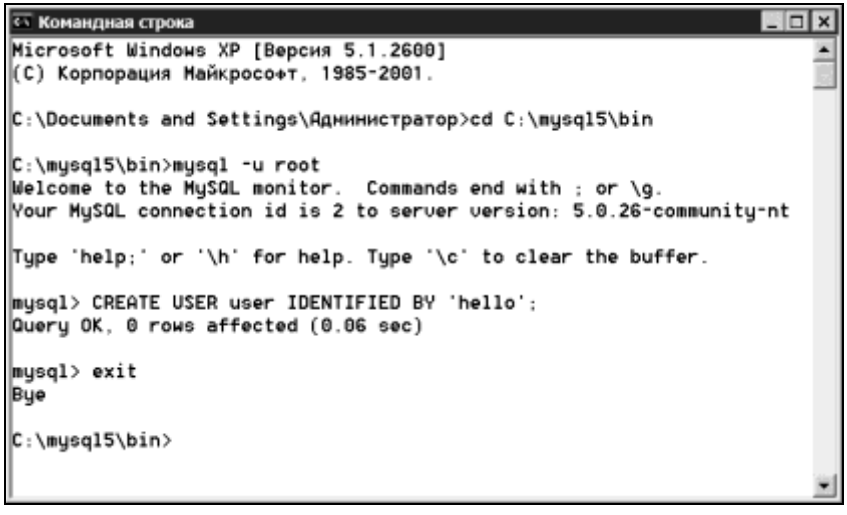


Рис. П2.28. Создание пользователя `user` с паролем `hello` и выход из оболочки

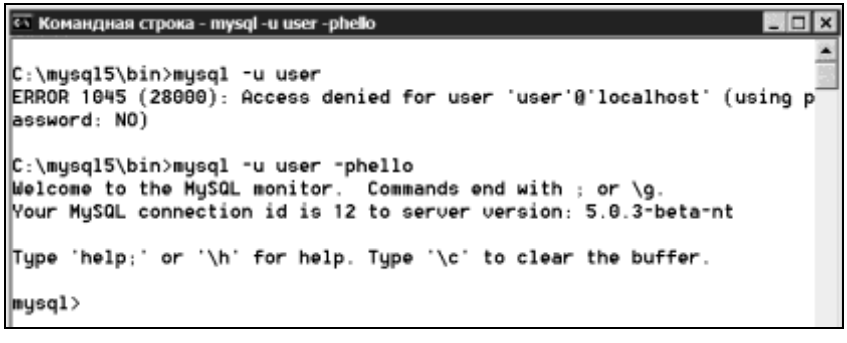


Рис. П2.29. Авторизация с использованием пароля

В некоторых ситуациях требуется скрыть пароль символами звездочки, в этом случае допускается использование параметра `-p` без значения. Утилита mysql запросит пароль при помощи отдельной строки `Enter password:`, в которой вводимые символы будут скрыты символом звездочки (рис. П2.30).

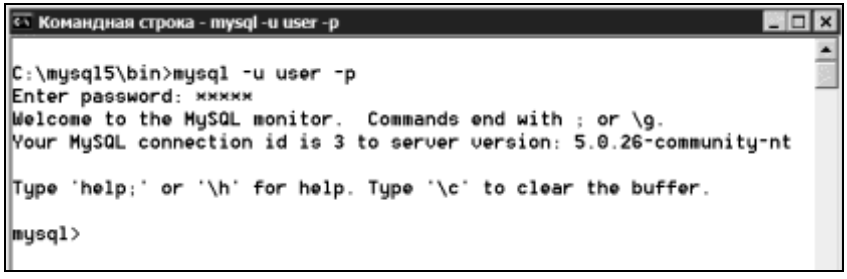


Рис. П2.30. Альтернативный способ авторизации

Команды и SQL-инструкции за редким исключением (`exit`, `quit`, `use`) должны заканчиваться точкой с запятой. Например, на рис. П2.31 приведена SQL-инструкция, запрашивающая у сервера MySQL его версию и текущую дату.

Замечание

Начиная с MySQL 5.0, изменить символ завершения запроса с точки запятой на новый символ, например `//`, можно либо при помощи команды `DELIMITER //`, либо указав разделитель при помощи параметра `--delimiter=name`.

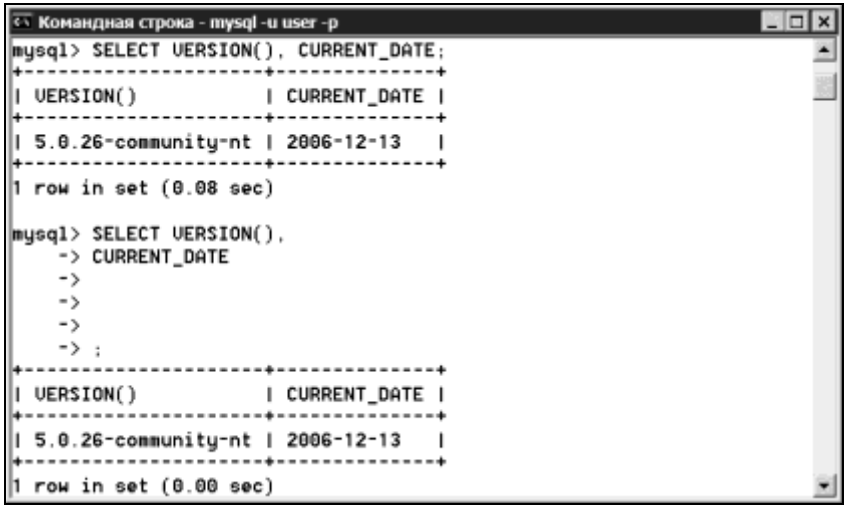


Рис. П2.31. Выполнение команд

После ввода пользователем команды она отправляется на сервер для выполнения, и, если нет ошибок в синтаксисе, на экран выводится результат в виде результирующей

таблицы, а на новой строке приглашение `mysql>`, разрешающее ввод следующей команды.

В первой строке таблицы с результатами содержатся заголовки столбцов, а в следующих строках — ответ сервера на запрос. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном ранее примере), mysql дает столбцу имя запрашиваемого выражения. После этого сообщается количество возвращаемых строк (`1 row in set` — одна строка в результате) и время выполнения запроса.

Для ввода ключевых слов можно использовать любой регистр символов. Так приведенные в листинге П2.5 запросы абсолютно идентичны.

Листинг П2.5. Имена инструкций и ключевые слова не зависят от регистра символов

```
SELECT VERSION(), CURRENT_DATE;
select version(), current_date;
SeLeCt vErSiOn(), current_DATE;
```

Если команда не помещается на одной строке, возможен переход на другую строку после нажатия клавиши <Enter> — запрос отправляется серверу только после того, как консольный клиент mysql встретит символ точки с запятой. Приглашение командной строки после ввода первой строки этого запроса меняется с `mysql>` на `->` (рис. П2.31). Таким образом, программа mysql показывает, что завершенного выражения она пока что не получила, и ожидает его полного ввода. Точно так же утилита mysql ведет себя, когда ожидает завершение строки, заключенной в двойные (`"`) или одинарные (`'`) кавычки (рис. П2.32).

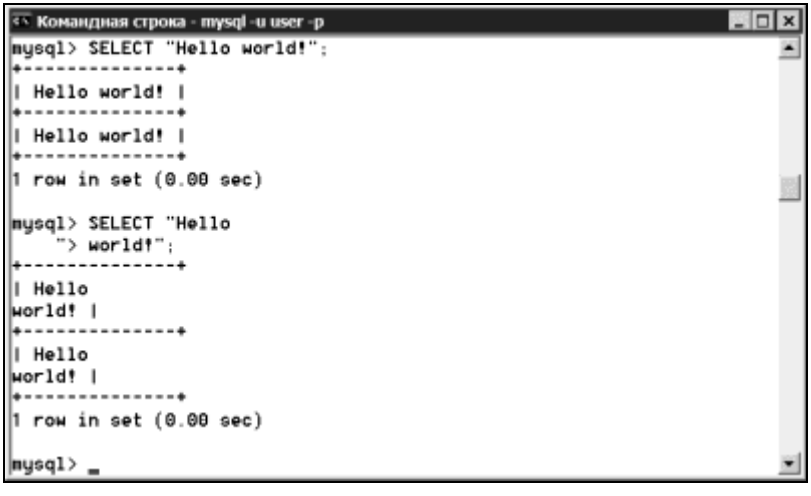


Рис. П2.32. Многострочный ввод текста, заключенного в двойные кавычки

Как видно из рис. П2.32, при переходе на следующую строку приглашение командной строки меняется с `mysql>` на `">`. Если строка заключена в одинарные кавычки, при-

глашение меняется на `'>`. Для того чтобы отменить текущий запрос, нужно ввести последовательность `\c`.

Уже введенные ранее команды не обязательно вводить снова, для этого достаточно их вызвать клавишами <↑> и <↓> (стрелка вверх и стрелка вниз); очистить строку запроса можно при помощи клавиши <Esc>. Полный список комбинаций клавиш, которые применяются в редакторе утилиты mysql, приведен в табл. П2.2.

Таблица П2.2. *Комбинации клавиш утилиты mysql*

Комбинация клавиш (UNIX)	Комбинация клавиш (Windows)	Значение
<↑>, <Ctrl>+<P>	<↑>	Вызов предыдущей команды из буфера команд
<↓>, <Ctrl>+<N>	<↓>	Вызов следующей команды из буфера команд
<←>, <Ctrl>+<B>	<←>	Перемещение курсора влево (назад)
<→>, <Ctrl>+<F>	<→>	Перемещение курсора вправо (вперед)
<Esc>+<Ctrl>+<B>	<Ctrl>+<←>	Перемещение курсора влево на одно слово
<Esc>+<Ctrl>+<F>	<Ctrl>+<→>	Перемещение курсора вправо на одно слово
<Ctrl>+<A>	<Home>	Перемещение курсора в начало строки
<Ctrl>+<E>	<End>	Перемещение курсора в конец строки
<Ctrl>+<D>	—	Удаление символа, выделенного курсором
<Delete>	<Delete>	Удаление символа справа от курсора
<Backspace>	<Backspace>	Удаление символа слева от курсора
<Esc>+<D>	—	Удаление слова
<Esc>+<Backspace>	—	Удаление слова слева от курсора
—	<Esc>	Удаление всего запроса
<Ctrl>+<K>	—	Удаление части строки от курсора до конца строки
<Ctrl>+<_>	—	Отмена последнего изменения

Параметры в утилитах MySQL могут иметь две формы: полную, начинающуюся с двух дефисов `--user`, и краткую, которая начинается с одного дефиса `-u`. Более подробно параметры утилиты mysql описываются в справочном руководстве.

П2.5. Перенос баз данных с одного сервера на другой

Одной из часто встречающихся задач является перенос баз данных с одного сервера на другой или просто создание резервной копии баз данных. СУБД MySQL предоставляет

для этого множество инструментов, однако чаще всего используется одно из двух решений:

- копирование бинарных файлов баз данных;
- создание SQL-дампа.

П2.5.1. Копирование бинарных файлов

Перенос бинарных файлов основан на том, что таблицы типа MyISAM (основной тип таблиц в MySQL) являются платформо-независимыми, и их можно перемещать с одного сервера на другой независимо от версии сервера и операционной системы, под управлением которой он работает.

> ***Замечание***
>
> Создание в каталоге данных нового каталога аналогично созданию новой базы данных.

В каталоге данных для каждой базы данных заводится свой подкаталог, а каждая таблица представлена тремя файлами, имена которых совпадают с именем таблицы, а расширения имеют следующий смысл:

- frm — определяет структуру таблицы, имена полей, их типы, параметры таблицы и т. п.;
- MYD — содержит данные таблицы (расширение образовано от сокращения MYData);
- MYI — содержит индексную информацию (расширение образовано от сокращения MYIndex).

Однако копирование данных непосредственно из каталога данных работающего сервера может привести к повреждению копий таблиц, причем это может случиться даже в том случае, если MySQL-сервер не обращался к копируемым таблицам в текущий момент. Поэтому перед копированием бинарных данных необходимо либо остановить сервер, либо блокировать таблицы на запись. Блокировку таблиц удобно осуществить при помощи оператора `FLUSH TABLES` (листинг П2.6).

Листинг П2.6. Блокировка таблиц на запись

```
FLUSH TABLES WITH READ LOCK;
```

Для того чтобы блокировка осталась в силе на время копирования, следует оставить клиент включенным и не выходить из него до тех пор, пока копирование каталога данных не будет завершено.

Для того чтобы снять блокировку на запись, следует выполнить запрос `UNLOCK TABLES` (листинг П2.7).

Листинг П2.7. Снятие блокировки

```
UNLOCK TABLES;
```

В дистрибутив MySQL входит скрипт горячего копирования бинарных файлов баз данных `mysqlhotcopy`. Данный скрипт действует по той же схеме, что описана ранее: блокирует таблицы базы данных `base` и копирует их бинарное представление по указанному пути /to/new/path (листинг П2.8).

Листинг П2.8. Использование скрипта `mysqlhotcopy`

```
mysqlhotcopy base /to/new/path
```

П2.5.2. Создание SQL-дампа

Копирование бинарных файлов не всегда удобно, иногда требуется развернуть базу данных в другой СУБД или в СУБД MySQL более ранней версии, не поддерживающей нововведения поздних версий. В этом случае часто прибегают к созданию SQL-дампов. SQL-дамп — это текстовый файл с SQL-инструкциями, выполнение которых воссоздает базу данных.

Основным инструментом для создания SQL-дампов служит утилита mysqldump. Для того чтобы создать резервную копию базы данных, например `base`, необходимо выполнить команду, представленную в листинге П2.9.

Листинг П2.9. Создание дампа базы данных `base`

```
C:\mysql5\bin> mysqldump –u root base > base.sql
```

Как видно из листинга П2.9, утилита `mysqldump` принимает имя пользователя при помощи параметра `-u`. Если учетная запись защищена паролем, следует добавить параметр `-p`. После всех параметров указывается имя базы данных `base`, для которой производится создание дампа. Так как вывод данных осуществляется в стандартный поток (за которым по умолчанию закреплен экран монитора), его следует перенаправить в файл (в листинге П2.9 — это base.sql). Перенаправление данных осуществляется при помощи оператора >. Если вместо оператора > использовать >>, то данные не будут перезаписывать уже существующий файл, а будут добавлены в конец файла.

При помощи параметра `--databases` (в сокращенной форме `-B`) можно создать дамп сразу нескольких баз данных, которые перечисляются через пробел (листинг П2.10).

Листинг П2.10. Создание дампа нескольких баз данных

```
C:\mysql5\bin> mysqldump –u root -B base mysql > base_mysql.sql
```

Так, команда, представленная в листинге П2.10, сохраняет дампы баз данных `base` и `mysql` в файл base_mysql.sql.

Если необходимо сохранить дамп всех баз данных MySQL-сервера, следует воспользоваться параметром `--all-databases` или в сокращенной форме `-A` (листинг П2.11).

Листинг П2.11. Создание дампа всех баз данных MySQL-сервера

```
mysqldump –u root --all-databases > all_databases.sql
```

Развернуть SQL-дамп на другом сервере можно при помощи утилиты mysql в пакетном режиме (листинг П2.12).

Листинг П2.12. Развертывание дампа базы данных с использованием mysql

```
mysql –u root test < base.sql
```

В листинге П2.12 данные из дампа base.sql перенаправляются на стандартный вход утилиты mysql, которая размещает таблицы базы данных `base` в базе данных `test`.

Развернуть SQL-дамп можно не только в пакетном режиме, но и в диалоговом. Самый простой способ — это воспользоваться командой `SOURCE`, которая выполняет несколько SQL-инструкций, перечисленных в SQL-дампе (листинг П2.13).

> ***Замечание***
>
> Следует указывать абсолютный путь к файлу в команде `SOURCE` или помещать SQL-дамп в каталог bin. В последнем случае можно использовать лишь имя файла.

Листинг П2.13. Использование команды SOURCE

```
SOURCE base.sql;
```

Объемные текстовые файлы и многочисленные бинарные файлы таблиц зачастую не удобны для транспортировки, и их обычно упаковывают в архив. В Windows не представляет труда создать архив при помощи специализированных пакетов WinZip или WinRar. В UNIX-подобных операционных системах для создания архива каталога обычно используют утилиту tar. В листинге П2.14 первая команда создает архив t.tgz каталога /dir, вторая — распаковывает содержимое архива t.tgz в каталог /dir.

> ***Замечание***
>
> Параметр `--same-owner` позволяет сохранить владельцев файлов и каталогов при распаковке их из архива. Без использования параметра `--same-owner` владельцем будет назначен пользователь, от чьего имени производится разархивация.

Листинг П2.14. Упаковка каталога в архив

```
tar czvf t.tgz /dir
tar xjvf --same-owner t.tgz /dir
```

Рекомендуемая литература

Технология Web-разработки включает в себя множество дисциплин, детальное изучение каждой из которых не под силу одному человеку. Однако знакомство только с одной областью, например, HTML или PHP, даже на уровне эксперта, не позволяет разработчику создавать профессиональные сайты высокой сложности. Ценность имеет не отдельный язык программирования, а их совокупность — технология. Чем большим количеством языков программирования и инструментов владеет разработчик, тем он ценнее. Для эффективной работы в Web-области придется приложить значительные усилия, т. к. Web-разработчик высокого уровня должен владеть следующими языками программирования и технологиями:

- язык разметки HTML;
- каскадные таблицы стилей CSS;
- основы дизайна (включая инструменты работы с векторной и растровой графикой);
- клиентский язык JavaScript;
- XML;
- серверный язык PHP;
- серверный язык Perl;
- Web-сервер Apache;
- устройство Интернета и основные интернет-протоколы (хотя бы HTTP, IP и DNS);
- сокеты и CURL;
- основы работы с электронной почтой;
- регулярные выражения;
- MySQL и язык SQL, желательно на уровне стандартов и различий между его диалектами для различных систем управления базами данных;
- Flash;
- основы операционной системы UNIX.

> ***Замечание***
>
> Перечисленные выше пункты касаются только технологии разработки сайтов для связки Apache, PHP и MySQL, помимо которых существуют технологии ASP.NET, Java и CouldFusion, которые в данной книге не затрагиваются.

Как уже упоминалось, охватить все области Web-разработки одному человеку невозможно, и, как правило, профессиональный Web-разработчик специализируется в нескольких наиболее интересных ему областях. Для создания сайта объединяется несколько разработчиков или даже организаций:

- хост-провайдеры, сдающие в аренду свои серверы, берут на себя организацию хост-площадки, требующую глубоких знаний операционной системы UNIX, администрирование Web-сервера Apache, DNS-сервера bind и MySQL-сервера;
- разработчики, специализирующиеся на Web-дизайне, уделяют большое внимание языку разметки HTML, XML, каскадным таблицам стилей, дизайну, инструментам работы с растровой (Photoshop) и векторной (3ds Max, CorelDRAW) графикой, технологиям JavaScript и Flash;
- разработчики, обычно специализирующиеся на одном или нескольких серверных языках (PHP и Perl), Web-протоколах (HTTP, IMAP, SMTP, FTP и т. п.), базах данных (MySQL) и регулярных выражениях, занимаются созданием бизнес-логики сайта.

Такое условное разделение вовсе не значит, что, специализируясь в одной области, разработчик может даже не знакомиться с соседними. Далее приводится список литературы, который, не претендуя на универсальность и объективность, позволит сориентироваться в порядке и объеме знаний, необходимых для профессиональной работы.

> ***Замечание***
>
> Охватить весь спектр представленной на рынке литературы мы не можем, более того, выпуск ряда книг со временем может быть прекращен, и читателю потребуется самостоятельно подбирать аналоги. Однако список позволит составить представление об объеме и характере знаний, которыми должен обладать Web-разработчик, специализирующийся на серверных языках программирования, т. е. не занимающийся самостоятельной разработкой Web-дизайна и администрированием серверов, однако представляющий соседние области и способный к переквалификации.

Книги в области программирования делятся на два типа: последовательные, излагающие материал от простого к сложному, и рецептурные, предлагающие ценные советы, разделы в которых, как правило, не зависят друг от друга и читать которые можно в произвольном порядке. В приводимом далее списке такие книги будут помечены отдельно.

> ***Совет***
>
> Если вы только начинаете изучать технологию, ориентируйтесь на книгу объемом 400—600 страниц; книга, содержащая 1000—1200 страниц, больше подходит в том случае, когда вы уже знакомы с технологией и хотите детально в ней разобраться, в противном случае велика вероятность "захлебнуться", не изучив книгу до конца.

Важно понимать, что техническая литература, в отличие от художественной, более насыщена информацией. Поэтому нельзя просто пропустить несколько страниц текста или приступать к следующей главе, не до конца понимая ранее изложенный материал. Возможно, одну и ту же главу придется прочитать несколько раз, а саму книгу отложить на полгода. Если разработчик утверждает, что на изучение технологии или языка программирования у него ушла пара недель — он либо знает язык очень поверхностно,

либо изучил большое количество языков до этого, поэтому многие сложные моменты оказались проработаны заранее. Чудес не бывает: любое искусство требует упорной работы и длительной практики. Об этом не любят вспоминать, когда цель достигнута, поэтому у начинающего программиста может сложиться ошибочное впечатление, что он топчется на месте, в то время как все вокруг буквально за пару недель способны к освоению приведенных ранее технологий.

HTML, XML, CSS, JavaScript и Flash

Для разработчика бизнес-логики не требуется экспертное знание языка разметки HTML, XML, каскадных таблиц стилей и клиентского языка JavaScript. Однако знакомство с данными технологиями крайне желательно, поскольку они описывают основы предметной области: сайт можно построить без использования PHP, однако без HTML и CSS создать более или менее приличный сайт просто не удастся.

- Мержевич В. В. HTML и CSS на примерах. — СПб.: БХВ-Петербург, 2005. — 488 с.

  Изучение языка разметки HTML в отрыве от каскадных таблиц стилей CSS в современных условиях не рационально. Представленная книга является идеальным введением в HTML, CSS, а также описывает их взаимодействие. Книга прекрасно сочетает в себе последовательное изложение от простого к сложному и множество советов, которые можно сразу применить на практике.

- Электронная документация по HTML.

  Книги являются своеобразным комментарием к официальной документации (как комментарии к уголовному или гражданскому кодексам). Как и любой комментарий, книги позволяют читателям разобраться в технологии быстрее, чем если бы они пользовались электронной документацией на английском языке, т. е. приобретая книгу, читатель экономит время в обмен на деньги. Тем не менее, электронной документаций не следует пренебрегать, особенно если вы знакомы с технологией, но вам требуется справочник для быстрого поиска.

  > ***Замечание***
  >
  > Знание английского языка не указывается в качестве необходимого для Web-разработчика, однако позволяет значительно повысить конкурентоспособность. Документация появляется в первую очередь на английском языке, часть ее вообще не переводится на национальные языки, переводится с опозданием или не в полном объеме.

  Если у вас установлена одна из версий Microsoft Office, вы можете найти CHM-справочник (в его имени будет присутствовать подстрока html) по HTML и DHTML в его справочных файлах в каталоге Program Files. Например, на компьютере авторов установлен Microsoft Office XP, и файл называется htmltag.chm.

- Мейер Э. CSS — каскадные таблицы стилей. Подробное руководство, 3-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2008. — 576 с.

  Книга представляет полное изложение каскадных таблиц стилей вплоть до спецификации CSS 2.1 с комментариями, позволяющими составить представление о возможностях, реализованных в современных браузерах. Книга читается легко, однако требует предварительного знакомства с HTML.

❒ Валентайн Ч., Минник К. XHTML.: Пер. с англ. — М.: Издательский дом "Вильямс", 2001. — 480 с.

На смену HTML должен прийти язык разметки XHTML, так Web-приложения для браузеров мобильных устройств (сотовые телефоны, КПК) уже сегодня разрабатываются с использованием этого языка разметки.

Так получилось, что помимо функций структурирования документов HTML стал выполнять и оформительские задачи. Для того чтобы разделить структуру и оформление документов, консорциумом W3C были введен язык разметки XML и каскадные таблицы стилей CSS, при помощи которых можно оформить XML-код. Язык XML настолько гибок, что позволяет самостоятельно определить нужный XML-формат путем разработки собственного словаря. В настоящий момент уже разработано огромное число словарей XML, позволяющих описывать любую информацию — от химических реакций (CML, Chemical Markup Language) до финансовых (OFX, Open Financial Exchange). По сравнению с HTML XML является более гибким форматом; повсеместного применения XML еще не получил, но это лишь вопрос времени.

❒ Питц-Моултис Н., Кирк Ч. XML.: Пер. с англ. — СПб.: БХВ-Петербург, 2001. — 736 с.

Полное и последовательное изложение языка разметки XML.

❒ Гудман Д. JavaScript и DHTML. Сборник рецептов. Для профессионалов. — СПб.: Питер, 2004. — 523 с.

Книга представляет множество рецептов для создания динамических сайтов с использованием клиентских технологий. Последовательное изложение языка программирования JavaScript в книге отсутствует, однако охватываются практически все области его применения.

❒ Гудман Д., Моррисон М. JavaScript. Библия пользователя, 5-е издание.: Пер. с англ. — М.: ООО "И. Д. Вильямс", 2006. — 1184 с.

Приводится полное и последовательное изложение JavaScript и DHTML. Большое внимание уделяется различиям в реализации JavaScript в современных браузерах.

❒ Электронная документация по JavaScript.

Точно так же, как и в случае HTML, в документации к Microsoft Office можно обнаружить CHM-описание для версии JavaScript, реализованной в Internet Explorer. У авторов данный файл называется jscript5.chm. За документацией по JavaScript, реализованному в движке Mozilla, следует обратиться к сайту **http://www.mozilla.org**.

❒ Бибо Б., Кац И. jQuery. Подробное описание по продвинутому JavaScript. — Пер. с англ. — СПб.: Символ-Плюс, 2009. — 384 с.

Краткое и насыщенное руководство по JavaScript-библиотеке jQuery, одной из самых популярных библиотек на сегодняшний момент. Компактный синтаксис библиотеки позволяет реализовывать сложные конструкции DHTML буквально в одну строку. Немаловажной особенностью библиотеки является тот факт, что она нивелирует различия в поведении разных браузеров, позволяя использовать особенности каскадных таблиц версии CSS 3, даже в тех браузерах, которые не поддерживают данный стандарт.

- Рейнхардт Р., Дауд С. Macromedia Flash 8. Библия пользователя.: Пер. с англ. — М.: "И. Д. Вильямс", 2006. — 1328 с.

  В книге приводится подробное и последовательное изложение основ работы в среде Macromedia Flash и создание Flash-роликов.

- Мук К. ActionScript 3.0 для Flash. Подробное руководство. — Пер. с англ. — СПб.: Символ-Плюс, 2009. — 992 с.

  В технологии программирования Flash используется язык программирования ActionScript. Данная книга представляет одно из полных его описаний.

PHP и Perl

Для работы приложения на стороне сервера используются серверные языки программирования, такие как PHP и Perl. Perl здесь рассматривается не случайно: разработчики PHP заимствовали из него множество концепций, и для более полного изучения PHP зачастую неплохо познакомиться с Perl, т. к. многие положения в документации по PHP описаны туманно или не описаны вообще.

PHP разрабатывался как более удобное средство для создания Web-приложений, и это удалось на славу — он стал одним из самых быстрых средств разработки. Разработчики PHP преследовали цель создать не контекстный язык с элементами лингвистического языка, как в случае Perl, а язык для быстрой разработки со строгим синтаксисом. Приложение, созданное одним разработчиком, должно было без проблем подхватываться другим, т. е. он планировался как прагматический язык. В свою очередь Perl создавался как произведение искусства — это целая философия. Его последователи зачастую фанатичны (дай им волю, они и операционную систему разработают на Perl), пишут стихи на Perl, устраивают конкурсы на самую непонятную программу, т. е. это достаточно романтический язык — в нем множество соблазнов для программиста, которые мешают созданию промышленного кода. Это язык для души. Perl более красив, чем PHP, но менее удобен для бизнеса и Web.

- Кузнецов М., Симдянов И. Самоучитель PHP 5/6. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 672 с.

  Приводится последовательное изложение языка, идеально подходящее для знакомства с PHP и его расширениями. Она начинает нашу серию, посвященную Web-разработке.

- Кузнецов М., Симдянов И. PHP. Практика создания Web-сайтов. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 1264 с.

  Книга рассматривает построение большого количества Web-приложений, входящих в состав полнофункционального Web-сайта. Цель книги — помочь разработчикам перейти от начальных знаний языка программирования PHP к созданию больших программных систем. Рассматривается создание CMS, форума, системы сбора статистики, рассылка, FTP-менеджер, каталоги продукции и т. п. Книга ориентирована на читателя, уже знакомого с основами HTML и PHP.

- Кузнецов М., Симдянов И. Объектно-ориентированное программирование на PHP. — СПб.: БХВ-Петербург, 2007. — 608 с.

Подробно рассматриваются объектно-ориентированные возможности PHP. Приводится множество примеров реального использования объектных возможностей при построении Web-приложений.

- Кузнецов М., Симдянов И., Голышев С. PHP 5 на примерах. — СПб.: БХВ-Петербург, 2005. — 576 с.

  В отличие от предыдущей книги, рассматриваются не готовые примеры, а сборник коротких и эффективных приемов, взятых из реальной практики: загрузка курса валют, создание динамического изображения, вывод случайного изображения из массива, создание PDF-документа и т. п. Книга носит рецептурный характер, и ее главы могут читаться в произвольном порядке. Несмотря на то, что книга не является последовательным изложением языка программирования PHP, в ней затрагиваются все основные моменты его использования.

- Скляр Д., Трахтенберг А. PHP. Сборник рецептов. 2-е изд. — Пер. с англ. — БХВ-Петербург, 2007. — 736 с.

  Книга с изложением полезных рецептов, главы можно читать в произвольном порядке.

- Кузнецов М., Симдянов И. Головоломки на PHP для хакера. — СПб.: БХВ-Петербург, 2006. — 464 с.

  Книга представляет собой задачник по Web-технологиям с уклоном в защиту Web-приложений от злоумышленников. Цель книги — помочь Web-разработчику научиться самостоятельно обнаруживать и устранять уязвимости в своем коде.

- Низамутдинов М. Ф. Тактика защиты и нападения на Web-приложения. — СПб.: БХВ-Петербург, 2005. — 432 с.

  Введение в проблему безопасности Web-приложений, созданных с использованием PHP и MySQL.

- Электронная документация по PHP.

  Электронную документацию по PHP можно загрузить с официального сайта **http://www.php.net**; здесь же находится перевод части документации на русский язык. Однако последним следует пользоваться крайне осторожно, т. к. информация на русском языке сильно устарела.

- Леки-Томпсон Э., Коув А., Новицки С., Айде-Гудман Х. PHP 5 для профессионалов.: Пер. с англ. — М.: ООО "И. Д. Вильямс", 2006. — 608 с.

  Книга содержит описание паттернов объектно-ориентированного программирования и применение их к PHP. К сожалению, в ней приведено описание объектно-ориентированного подхода для PHP 4, затрагивая ООП PHP 5 лишь на двух страницах, посвященных описанию нововведений. Книга читается тяжело.

- Уолл Л., Кристиансен Т., Орвант Д. Программирование на Perl. — Пер. с англ. — СПб.: Символ-Плюс, 2004. — 1152 с.

  Знаменитая Camel-book ("Верблюжья книга") от создателя языка Ларри Уолла — замечательное введение в язык, с юмором раскрывающее тонкости и философию языка Perl. Книга представляет собой последовательное изложение языка от простого к сложному.

- Кристиансен Т., Торкингтон Н. Perl. Сборник рецептов. Для профессионалов. 2-е изд. — Пер. с англ. — СПб.: Питер, 2004. — 928 с.

  Книга носит рецептурный характер и является своеобразным дополнением книги Ларри Уолла "Программирование на Perl".

СУБД MySQL

Ни один современный сайт не может обойтись без использования баз данных. В Российской Федерации широкое распространение получила СУБД MySQL, выступающая стандартом де-факто для российских хост-провайдеров и Web-разработчиков.

> ***Замечание***
>
> Несмотря на то, что существует стандарт языка запросов SQL, каждая СУБД реализует собственный диалект, иногда значительно отличающийся от диалектов других СУБД. Поэтому процесс изучения SQL сводится сначала к изучению общих основ SQL, а затем диалекта конкретной базы данных.

- Грофф Дж., Вайнберг П. Энциклопедия SQL, 3-е изд. — СПб.: Питер, 2003. — 896 с.

  Добротное и последовательное изложение языка запросов SQL с указанием особенностей отдельных диалектов. Очень ясное и прозрачное изложение, снабженное большим количеством поясняющих схем и рисунков.

- Кузнецов М., Симдянов И. Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2005. — 560 с.

  Введение в диалект SQL для СУБД MySQL 5.0. Изложение ведется последовательно от простого к сложному. Затрагиваются нововведения MySQL 5.0: хранимые процедуры и функции, представления, триггеры, курсоры и информационная схема.

- Кузнецов М., Симдянов И. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024 с.

  Последовательное и полное изложение MySQL 5.0. В отличие от книги "Самоучитель MySQL 5", подробно рассматривается администрирование MySQL и взаимодействие с языками программирования C++, Perl и PHP под операционными системами Windows и UNIX.

- Дюбуа П. MySQL, 2-е изд. — М.: Вильямс, 2007. — 1168 с.

  Изложение MySQL, на которое стоит обратить внимание. Очень подробно описывается администрирование СУБД в условиях UNIX-подобной операционной системы. К сожалению, книга лишь вскользь описывает нововведения MySQL 4.1 и 5.0.

- Кузнецов М., Симдянов И. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592 с.

  Книга посвящена MySQL версии 5.1 и помимо указанных выше нововведений затрагивает планировщик заданий и сегментирование. Большое внимание уделяется объектно-ориентированной библиотеке php_mysqli. Книга носит рецептурный характер, однако может использоваться в том числе и для последовательного изучения материала.

- Дюбуа П. MySQL. Сборник рецептов. — Пер. с англ. — СПб.: Символ-Плюс, 2004. — 1056 с.

  Книга носит рецептурный характер и может читаться с любой главы, однако, как и указанная ранее книга этого же автора, описывает лишь MySQL версии 4.0.

- Электронная документация по MySQL.

  В разделе официального сайта MySQL, посвященного разработчикам **http://dev.mysql.com**, можно найти ссылки на документацию в электронном формате, которая обновляется ежедневно. На этом же сайте присутствует и русскоязычный вариант документации, но только для версии MySQL 4.0; для более поздних версий она доступна только на английском языке.

  Замечание

  Недавно компания MySQL AB выпустила перевод документации на русский язык в виде книг: MySQL. Руководство администратора.: Пер. с англ. — М.: "И. Д. Вильямс, 2005. — 624 с.; MySQL. Справочник по языку.: Пер. с англ. — М.: "И. Д. Вильямс", 2005. — 432 с. Однако следует иметь в виду, что официальная документация, подготавливаемая разработчиками, зачастую просто нечитаема. Книги содержат зачастую совершенно несвязанные участки текста, понять смысл которых можно только в результате длительных экспериментов. Их можно рекомендовать, если чтение английской документации у вас вызывает затруднение.

Интернет и Web-сервер Apache

Web-приложения — это всегда распределенные приложения, в работе которых принимает участие множество серверов и клиентов. Для того чтобы создавать эффективные Web-приложения, необходимо очень четко представлять себе принципы работы Интернета и его протоколов.

- Камер Д. Э. Сети TCP/IP. Том 1. Принципы, протоколы и структура, 4-е изд.: Пер. с англ. — М.: "И. Д. Вильямс", 2003. — 880 с.

  Прекрасное введение в историю и современное устройство Интернета. Подробно описывается физическая и логическая архитектура, протоколы IP, ICMP, UDP и TCP. К сожалению, мало внимания уделяется протоколам прикладного уровня (HTTP, SMTP, FTP, IMAP и т. п.), которые наиболее интересны для Web-разработчика.

- Кришнамурти Б., Рексфорд Дж. Web-протоколы. Теория и практика. — М.: ЗАО "Издательство БИНОМ", 2002. — 592 с.

  Описание протокола HTTP и приемов работы с ним. Книга не очень эффективная и содержит мало примеров, однако из бумажных книг по протоколу HTTP посоветовать практически нечего. Книга читается тяжело.

- Электронная документация по протоколам Интернета.

  Все протоколы и спецификации Интернета описываются в так называемых RFC-документах, получить которые можно с большого количества сайтов, стоит лишь ввести в поисковую систему аббревиатуру RFC. Практически все RFC-документы на английском языке, и только часть из них переведена на русский.

- Альбитц П., Ли К. DNS и BIND. — Пер. с англ. — СПб.: Символ-Плюс, 2004. — 688 с.

  Подробно описываются система доменных имен Интернета и конфигурирование сервера BIND.

- Кабир М. Дж. Сервер Apache 2. Библия пользователя.: Пер с англ. — М.: "И. Д. Вильямс", 2002. — 672 с.

  Прекрасное изложение конфигурирования Web-сервера Apache. К сожалению, не затрагивает последнюю версию Apache 2.2, а также ряд новых модулей, позволяющих более эффективно управлять сервером в условиях виртуального хостинга. Однако, несмотря на все недостатки, представляет собой одно из лучших описаний Web-сервера Apache на сегодняшний день.

Регулярные выражения

Регулярные выражения являются специализированным языком программирования, позволяющим манипулировать текстом. В свободном виде регулярные выражения практически не используются и применяются совместно с другими языками программирования. Регулярные выражения можно найти в JavaScript, PHP, Perl, MySQL, Apache и многих других языках программирования и технологиях. Будучи специализированным языком программирования, регулярные выражения имеют множество диалектов, поэтому в каждом конкретном случае потребуется исследование возможностей данной реализации регулярных выражений.

> ***ЗАМЕЧАНИЕ***
>
> Существуют две развитые реализации регулярных выражений: регулярные выражения Perl, реализованные и развиваемые в рамках языка Perl, и регулярные выражения POSIX, реализуемые в рамках стандарта переносимых операционных систем. Более удобными и распространенными являются регулярные выражения Perl.

- Фридл Дж. Регулярные выражения, 2-е изд. — СПб.: Питер, 2003. — 464 с.

  Книга представляет собой единственное наиболее полное издание, посвященное регулярным выражениям. Несмотря на то, что в ней не описываются регулярные выражения применительно к PHP, это не мешает восприятию и эффективному применению полученных знаний на практике. Рассматриваются все распространенные диалекты регулярных выражений. Издательство "Питер" приняло решение не переиздавать книгу, однако, к радости ее поклонников, она выложена издательством в свободный доступ. Ее также можно загрузить с нашего сайта по ссылке **http://www.softtime.ru/info/fridl.php**.

UNIX-подобные операционные системы

UNIX-подобные операционные системы, в частности Linux и FreeBSD, интенсивно используются для построения Интернета. Только 20% серверов работают под управлением операционных систем, не являющихся UNIX-подобными. С другой стороны,

клиентские машины в основном работают под управлением операционной системы Windows. В результате часто возникают конфликты, когда Web-приложение, разработанное в условиях Windows, перестает работать в UNIX. Вопрос, насколько досконально следует изучать UNIX-подобные операционные системы, является делом Web-разработчика, однако иметь представление о них просто необходимо.

- Тейнсли Д. Linux и UNIX: программирование в shell. Руководство разработчика.: Пер. с англ. — К.: Издательская группа BHV, 2001. — 464 с.

  Добротное изложение стандартного командного интерпретатора UNIX — bash. Подробно рассматривается система прав доступа UNIX и создание собственных скриптов.

- Немеет Э., Снайдер Г., Сибасс С., Хейн Т. UNIX: руководство системного администратора. Для профессионалов. — СПб.: Питер. К.: Издательская группа BHV, 2003. — 925 с.

  Данная книга является настоящей энциклопедией UNIX, рассказывающей о тонкостях работы системного администратора этой операционной системы. Повествование опирается на операционные системы Solaris, HP-UX, Red Hat и FreeBSD. Книга прекрасно подойдет как профессионалу, так и новичку, который захочет окунуться в мир UNIX. Изложение материала идет от простого к сложному, рассматриваются процессы пуска и останова системы, привилегии, управление процессами, файловая система и т. д. Не обделены вниманием и сетевые настройки: DNS, BIND, sendmail, NIS, Apache, Usenet, FTP-сервер. Книга читается легко и интересно, приводится множество реальных примеров из практики системного администрирования.

- Реймонд Э. С. Искусство программирования для UNIX.: Пер. с англ. — М.: "И. Д. Вильямс", 2005. — 544 с.

  Данная книга описывает философию программирования под UNIX. В ней вы не найдете листингов, описания системных вызовов, но зато сможете понять дух операционной системы, принципы построения интерфейсов и архитектуры UNIX-приложений.

  В книге затрагиваются вопросы истории развития легендарной операционной системы и раскрываются секреты ее долгожительства. Вы узнаете, почему ООП более популярен в Windows, чем в UNIX, а также сможете познакомиться с культурой и фольклором UNIX.

  Если вы никогда не сталкивались с операционной системой UNIX — это именно та книга, с которой нужно начать знакомство. Вы получите ясное представление о том, куда следует идти и что делать, и вообще нужен вам UNIX или нет. Если вы хорошо знаете данную операционную систему, иметь эту книгу просто необходимо: создававшаяся в течение 5 лет, она содержит мудрость проектирования под UNIX, накопленную UNIX-сообществом на протяжении 30 лет.

- Костромин В. Самоучитель Linux для пользователя. — СПб.: БХВ-Петербург, 2003. — 672 с.

  Добротный самоучитель по операционной системе Linux, последовательно вводящий читателя в обсуждаемую предметную область.

- Стаханов А. Linux. — СПб.: БХВ-Петербург, 2003. — 912 с.

  Полное и последовательное изложение основ операционной системы Linux. Подробно описывается установка, конфигурирование и работа в операционной системе Linux.

- Скловская С. Л. Команды Linux. Справочник, 3-е изд., перераб. и доп. — СПб.: ООО "ДиаСофтЮП", 2004. — 848 с.

  Операционная система Linux имеет подробную электронную документацию, вызвать которую можно при помощи команды `man`. Однако зачастую удобнее воспользоваться бумажным справочником, особенно если у вас вызывает затруднения английский язык.

Методология программирования

Программирование — это не только код, спецификации, операционные системы, это еще и методология. Существуют книги, которые не связаны непосредственно ни с одним языком программирования, но помогают чувствовать направление и основные тенденции развития индустрии.

- Кузнецов М., Симдянов И. Программирование: ступени успешной карьеры. — СПб.: БХВ-Петербург, 2006. — 320 с.

  Процесс программирования очень увлекателен сам по себе. Однако начинающим программистам следует иметь в виду, что если создаваемый ими код не выдерживает конкуренции или его не удается продать, рано или поздно придется уходить из любимой области. Данная книга посвящена тому, как утвердиться и успешно развиваться в сфере программирования.

- Жарков С. В. Shareware: профессиональная разработка и продвижение программ. — СПб.: БХВ-Петербург, 2002. — 320 с.

  Прекрасное руководство продвижения программных продуктов в условиях российской действительности и современного Запада. Книга написана с юмором и читается очень легко.

- Макконнелл С. Совершенный код. Мастер-класс.: Пер. с англ. — М.: Издательско-торговый дом "Русская Редакция"; СПб.: Питер, 2005. — 896 с.

  Это настоящая библия разработчика, содержащая самое полное собрание эффективных методик современного программирования, от правил именования переменных и функций и принципов структурного программирования до объектно-ориентированного программирования, которая поможет вам достичь высшего мастерства в создании эффективного, читабельного и легко сопровождаемого кода. Книга не содержит "воды" и недомолвок — вы получите настолько подробные сведения, что никогда не сможете использовать их в полном объеме. Автору удалось добиться практически невозможного: не привязываясь к конкретному языку программирования или платформе, донести до читателей опыт, накопленный за 40 лет существования программирования. Приводимые им рекомендации не голословны, а подтверждаются исследованиями специалистов в области computer science,

а также отчетами таких известных корпораций как IBM и Microsoft. Более того, в отличие от всех подобных книг, вы не найдете философских рассуждений, которые невозможно реализовать на практике: все описываемые в книге приемы взяты из реальных проектов и реального кода.

- Брукс Ф. Мифический человеко-месяц, или как создаются программные системы. — Пер. с англ. — СПб.: Символ-Плюс, 1999. — 304 с.

  Классическое произведение по проектированию больших программных продуктов, прочитать которое — долг любого программиста. Пожалуй, ни на одну книгу не ссылаются так часто и, пожалуй, ни одна книга не описывает более честно реальные сложности управления программными проектами.

Предметный указатель

A

Advanced Encryption Standard (AES) 814
ARPANET 5
AUTO_INCREMENT, атрибут 856

C

chcp 978
Cookies 468
CSS 16

D

DELIMITER, команда 980

F

FTP 17

H

HTML 16
HTTP-заголовок 454
  Cache-Control 465
  Connection 506
  Content-Disposition 485
  Content-Length 483, 519
  Content-Type 517
  Expires 465
  Host 506
  Last-Modified 465
  Set-Cookie 471
  Transfer-Encoding 517
  User-Agent 526
  получение с сервера 515

I

IP-адрес 827, 828
  клиента 182
IP-дейтограмма 8

M

Message-Digest Algorithm (MD5) 815
MILNET 6
MIME-тип 899

S

Secure Hash Algorithm (SHA1) 814
SSH 17
Structured Query Language (SQL) 15, 743

T

$this 573
TCP/IP 6

U

Universal Unique Identifier (UUID) 828

W

Web-сервер 13

X

XML 16

А

Абстрактные типы данных 567
Аксессор 586
Алгоритм:
  3DES 501
  DEC 495, 496
  MD5 495
  ROT13 495
  SHA1 495
  Оливера 296

Б

База данных:
  information_schema 748
  mysql 748
  реляционная 744, 746
  система управления 743
  создание 747
  удаление 748
Безопасный режим 539
Блокировка:
  получение 827
  проверка 827
  снятие 828

В

Виртуальный хост 945
Выражение 22
  составное 24

Г

Гистограмма 922

Д

Дамп 984
Деструктор 584
Директива 972
  AddDefaultCharset 949
  allow_url_fopen 558
  allow_url_include 558
  arg_separator.input 552
  arg_separator.output 552
  asp_tags 536
  auto_append_file 553, 556
  auto_globals_jit 553
  auto_prepend_file 553, 556
  CustomLog 946
  date.default_latitude 561
  date.default_longitude 561
  date.sunrise_zenith 561
  date.sunset_zenith 561
  date.timezone 561
  default_charset 554
  default_mimetype 554
  default_socket_timeout 558
  disable_classes 544
  disable_functions 544
  display_errors 549
  display_startup_errors 549
  DocumentRoot 946
  engine 536
  error_log 550
  error_reporting 549, 650
  ErrorLog 946
  expose_php 547
  extension 559
  extension_dir 559
  file_uploads 558
  from 558
  highlight.bg 547
  highlight.comment 547
  highlight.default 547
  highlight.html 547
  highlight.keyword 547
  highlight.string 547
  ignore_repeated_errors 549
  ignore_user_abort 547
  log_errors 549
  log_errors_max_len 549
  magic_quotes_gpc 553, 555, 556
  magic_quotes_runtime 553
  magic_quotes_sybase 553
  max_execution_time 548
  max_input_time 548
  memory_limit 503, 548, 565
  NameVirtualHost 946
  open_basedir 544, 562
  output_buffering 481, 536, 537, 539
  php_admin_flag 562
  php_admin_value 562
  php_flag 562
  php_value 562
  PHPIniDir 535, 952
  post_max_size 553
  precision 536, 537
  realpath_cache_size 547
  realpath_cache_ttl 547
  register_argc_argv 553
  register_globals 552, 554
  register_long_arrays 552
  request_order 552
  safe_mode 541
  safe_mode_allowed_env_vars 541, 542
  safe_mode_exec_dir 542
  safe_mode_gid 539, 542
  safe_mode_include_dir 540, 542
  safe_mode_protected_env_vars 542
  ServerName 946
  session.auto_start 560
  session.cookie_domain 560
  session.cookie_lifetime 473, 560
  session.cookie_path 560
  session.cookie_secure 560
  session.gc_divisor 561
  session.gc_maxlifetime 561
  session.gc_probability 560
  session.name 560
  session.save_handler 560
  session.save_path 473, 560
  session.serialize_handler 560
  session.use_cookies 560
  session.use_only_cookies 560
  short_open_tag 21, 536
  track_errors 550
  upload_max_filesize 558
  upload_tmp_dir 558
  user_agent 558
  variables_order 552
Директива php.ini
  error_reporting 36
  extension 559
  extension_dir 559
  magic_quotes_gpc 265
  memory_limit 31
  precision 30
  upload_max_filesize 236
Доменное имя 10

Ж

Жесткая ссылка 392

З

Замечание (Notice) 36
Запрос, выборка данных 775

И

Изображение:
  вывод в браузер 903
  замена одного цвета другим 915
  защита HTML-формы 930
  поворот 908
  создание 901
  уменьшение количества цветов 914
  уменьшенная копия 906
Имя сервера 182
Индексы:
  PRIMARY KEY 773
  UNIQUE 773
  внешний ключ 746
  логический ключ 746
  первичный ключ 746, 765
  суррогатный ключ 746
Инкапсуляция 567, 571
Инструкция:
  include 569
  include_once 569
  require 569
  require_once 569
Интерполяция 575
Интерфейс 635
  наследование 641
  проверка существования 640
Исключение 660
Итератор 731

К

Каталог 392
данных 747
Класс 568
DateInterval 727—729
DateTime 723
dir 721
DirectoryInterator 736
Exception 662
FilterIterator 738
LimitIterator 739
mysqli 853
mysqli_result 858
mysqli_stmt 856—858, 873
Reflection 694, 719
ReflectionClass 695, 702, 709
ReflectionException 694, 715
ReflectionExtension 695, 716
ReflectionFunction 694, 695
ReflectionMethod 695, 710
ReflectionObject 695
ReflectionParameter 694, 700
ReflectionProperty 695, 712
абстрактный 630
базовый 621
производный 621
Клиент 11
Кодировка 749, 810, 977
Коды состояния 458
Комментарий 25
в конфигурационном файле 972
Константа:
__CLASS__ 60, 605
__FILE__ 60, 605
__FUNCTION__ 60, 605
__LINE__ 60, 605
__METHOD__ 60, 605
FALSE 38, 56
M_1_PI 362
M_2_PI 362
M_2_SQRTPI 362
M_E 362
M_EULER 363
M_LN10 362
M_LN2 362
M_LNPI 363
M_LOG10E 362
M_LOG2E 362
M_PI 362
M_PI_2 362
M_PI_4 362
M_SQRT1_2 363
M_SQRT2 362
M_SQRT3 363
M_SQRTPI 362
NULL 56
RAND_MAX 366
TRUE 38, 56
динамическая 58
объявление 54
предопределенная 60
проверка существования 59
Конструктор 580
Конструкция:
' … ' 32
" … " 31
25
/* */ 25
// 25
?> 18
` … ` 32
<? 21
<?= 21
<?php 18
array() 110, 114, 195
declare 206
echo 19, 62, 273
empty() 40
global 193
include 101, 540
include_once 101, 540
isset() 39, 140, 588
list() 123, 196
require 101, 540
require_once 102, 540
static 195
unset() 38, 158, 477, 585, 590
Конструкция SQL:
ALL 789, 790
AS 782, 820
AUTO_INCREMENT 769
BETWEEN 780
DEFAULT 764
DEFAULT CHARACTER SET 749
DISTINCT 788, 789, 817, 823, 824
DISTINCTROW 788
GROUP BY 789, 816, 822
IGNORE 769
IN 780
INTERVAL 767
LIMIT 772, 787
NOT BETWEEN 780
NOT IN 781
ORDER BY 783, 820, 824, 872
ORDER BY ... ASC 786
ORDER BY ... DESC 784
SEPARATOR 824
SET 773
VALUES 770, 774
WHERE 772, 773, 776, 781, 822
Контролируемый блок 660
Кривая Безье 918
Круговая диаграмма 924

Л

Локаль 266

М

Массив:
$_COOKIE 469
$_FILES 233
$_GET 214, 540
$_SERVER 177, 540
$_SESSION 475
$GLOBALS 194
ассоциативный 116
включительное пересечение 140
индексный 116
исключительное пересечение 139
ключи 165
количество элементов 145
многомерный 121
проверка существования элементов 140
равный другому массиву 136
разбиение 171
сечение 131
слияние 133
случайный элемент 147

(окончание рубрики см. на стр. 1002)

Массив *(окончание)*:
создание 110
сортировка 150
сравнение 136
сумма элементов 147
уникальные значения 170
фильтрация 172
эквивалентный другому массиву 136
Метка 35
Метод:
__autoload() 578, 586
__call() 578, 591
__clone() 578, 609
__construct() 577, 580
__destruct() 578, 584
__get() 578, 586
__isset() 578, 588
__set() 578, 586
__set_state() 578, 597
__sleep() 578, 612
__toString() 578, 593
__unset() 578, 589
__wakeup() 578
абстрактный 631
динамический 591
класса 568
перегрузка 627
статический 603

Н

Название скрипта 184
Наследование 568, 621

О

Объект 568
дамп 576
интерполяция 593
клонирование 608
создание 569
уничтожение 570
Округление чисел 370
Оператор:
! 78, 83
!= 72, 136
!== 72, 136
% 63
& 67
&& 78
* 63
. 61
.= 62
/ 63
@ 107
^ 67, 69
| 67, 68
|| 78, 80
~ 67, 70
+ 63, 133
++ 63, 66
< 72
<< 67, 70
<<< 35
<= 72
<> 136
= 27
== 72, 74, 136, 292
=== 72, 74, 136
=> 111
> 72
-> 572
>= 72
>> 67, 71
and 78
break 85, 92
catch 660
class 568
continue 89
CREATE USER 978
do ... while 96
extends 622
FLUSH TABLES 983
for 97
foreach 129
if 75
implements 636
instanceof 643
interface 635
new 581
or 78
return 102, 195
switch 85
throw 660
try 660
UNLOCK TABLES 983
while 90
x ? y
z 84
поразрядный 67
приоритет 108
Оператор SQL:
> 776
ALTER TABLE 760
AND 778
CREATE DATABASE 747, 860
CREATE TABLE 756
DELETE 771, 772, 844
DESCRIBE 759
DROP DATABASE 748
INSERT 769, 774, 844
INSERT, многострочный 770
INSERT, однострочный 763
NOT LIKE 782
OR 778
REPLACE 773, 774
SELECT 776, 789, 790, 822, 844
SET NAMES 832
SHOW DATABASES 747
SHOW TABLES 757
TRUNCATE TABLE 772
UNION 789, 790
UNION ALL 790
UPDATE 772, 844
Определитель:
заполнения 276
преобразования 275
типа 275
точности 277
ширины поля 277

П

Пакет 8
Параметр 971
--des-key-file 813
Параметр mysql:
--delimiter=name 980
-p 979
-u 975
Параметр mysqld:
--defaults-file 971
--sql-mode 791
--standalone 968
Параметр mysqldump:
–A 984
--all-databases 984
–B 984
--databases 984

Передача параметров:
по значению 189
по ссылке 189
Переключатель 231
Переменная 27
глобальная 193
динамическая 52
значение 27
инициализация 27
интерполяция 32
неинициализированная 35
окружения 977
проверка существования 39
статическая 194
уничтожение 38
Подавление кэширования 464
Поле пароля 224
Полиморфизм 568, 628
Пользователь:
root 978
имя 978
пароль 978
создание 978
Пользовательский агент 181, 526, 558
Порт 963
Последовательность * 775
Почтовое сообщение, отправка 885
Права доступа UNIX 438, 539
Прокси-сеервер 464
Протокол 8, 454
FTP 875
HTTP 506
IP 8
Пунктирная линия 926

Р

Расстояние Левенштейна 297
Рассылка писем 887
Регулярные выражения:
квантификаторы 315
метасимволы 314
модификаторы 316
позиционные проверки 316
специальные символы 315
функции 317
Реферер 524

С

Сглаженная кривая 918
Секретный ключ 813, 814
Сервер 11
Серверный язык 13
Сервис 964, 967
Серилизация 610
Сессия 473
Символическая ссылка 392
Скрипт PHP 18
Скрытое поле 226
Сокет 502
Спецификатор доступа:
private 571, 624
protected 626
public 571, 624
var 571
Список 229
Стандарт языка SQL 744
Стандартный порт 11
Столбец:
псевдоним 782
уникальные значения 788
Строка 31
Строка запроса 183
Структурированный язык запросов 743
СУБД 14, 743

Т

Таблица:
InnoDB 961
MyISAM 961
объединение 789
результирующая 776
создание 861
Текстовая область 225
Текстовое поле 223
Тип данных:
array 29, 54, 113
BIGINT 752, 828
BIT 752
BLOB 754
BOOL 752
boolean 28, 38, 54
BOOLEAN 752
CHAR 754
DATE 753, 766
DATETIME 753
DEC 752
DECIMAL 752
double 29, 30
DOUBLE 752
DOUBLE PRECISION 752
ENUM 754
float 29, 30, 54
FLOAT 752
INT 752
int64 28
integer 28, 54
INTEGER 752
LONGBLOB 754
LONGTEXT 754
MEDIUMBLOB 754
MEDIUMINT 751
MEDIUMTEXT 754
NULL 29, 37
NUMERIC 752
object 29, 54
REAL 752
resource 29, 54
SET 754
SMALLINT 751
string 29, 54
TEXT 754
TIME 753
TIMESTAMP 753
TINYBLOB 754
TINYINT 751
TINYTEXT 754
VARCHAR 754
YEAR 753
неявное приведение 46
строковый 765
явное приведение 48

У

Удаление изображений 329
Универсальный уникальный идентификатор 828
Уровень:
аппаратный 8
прикладной 11
приложений 9
сетевого интерфейса 8
транспортный 9
Утилита:
apachectl 951
ApachtMonitor 943

(окончание рубрики см. на стр. 1002)

Утилита *(окончание)*:
mysql 974
MySQL Command Line Client 966
MySQL Server Instance Configuration Wizard 961
mysqldump 972, 984
mysqlhotcopy 984
tar 985

Ф

Файл 392
CSV 422
db.opt 747
DES 813
hosts 947
ini 424
MIME-тип 899
абсолютный путь 393
архивный 450
атрибуты 432
бинарный режим 411
блокировка 417
временный 397
дозапись 415
загрузка на сервер 232
закрытие 414
запись 410
копирование 398
относительный путь 394
переименование 399
размер 898
режим трансляции 411
сетевой путь 395
создание 392
удаление 399
чтение 404
Флажок 227
Функция:
__autoload() 641
abs() 370
acos() 378, 381
acosh() 378, 382
addcslashes() 264, 266
addslashes() 264
array_change_key_case() 165, 166
array_chunk() 170, 171
array_combine() 118
array_count_values() 145, 146
array_diff() 137, 139
array_diff_assoc() 138
array_diff_key() 138
array_diff_uassoc() 138
array_diff_ukey() 138
array_fill() 114
array_filter() 170, 172
array_flip() 165, 168
array_intersect() 138, 140
array_intersect_assoc() 138
array_intersect_key() 138
array_intersect_uassoc() 138
array_intersect_ukey() 139
array_key_exists() 142, 143
array_keys() 165, 168
array_map() 170, 174
array_merge() 134
array_merge_recursive() 134
array_multisort() 150, 156
array_pad() 160, 161
array_pop() 160, 162
array_push() 160, 161
array_rand() 148
array_reduce() 160
array_reverse() 165
array_search() 142, 143
array_shift() 160, 163
array_slice() 131
array_splice() 131, 132
array_sum() 147
array_udiff() 138
array_udiff_assoc() 138
array_udiff_uassoc() 138
array_uintersect() 139
array_uintersect_assoc() 139
array_uintersect_uassoc() 139
array_unique() 170
array_unshift() 160, 162
array_values() 165, 167
array_walk() 170, 173
array_walk_recursive() 170
arsort() 150, 153
asin() 378, 381
asinh() 378, 381
asort() 150, 153
atan() 378, 382
atan2() 378
atanh() 378, 383
base_convert() 367
base64_decode() 283
base64_encode() 283, 284
basename() 301, 303
bcadd() 389, 390
bccomp() 390
bcdiv() 389
bcmod() 390
bcmul() 389
bcpow() 389
bcpowmod() 390
bcscale() 390
bcsqrt() 390
bcsub() 389
bin2hex() 298
bindec() 367, 369
call_exists() 579
call_user_func() 199, 579
call_user_func_array() 199, 579
call_user_method() 579
call_user_method_array() 578
ceil() 370, 371
chdir() 443, 444
checkdate() 349, 352
checkdnsrr() 530
chgrp() 439
chmod() 439, 440
chop() 249
chown() 439
chr() 237, 238
chroot() 443
chunk_split() 304, 306
clearstatcache() 433
closedir() 444
compact() 118, 119
constant() 58
convert_cyr_string() 283, 543
convert_uudecode() 283
convert_uuencode() 283
copy() 398
cos() 378
cosh() 378, 380
count() 145
count_chars() 237, 239
crc32() 495
create_function() 199
crypt() 495, 496
ctype_alnum() 241

ctype_alpha() 241
ctype_cntrl() 241
ctype_digit() 241
ctype_graph() 241
ctype_lower() 241
ctype_print() 241
ctype_punct() 241
ctype_space() 241
ctype_upper() 241
ctype_xdigit() 241
curl_exec() 508
curl_init() 508
curl_setopt() 508, 518
current() 126
date() 348, 349
date_default_timezone_get() 342, 343
date_default_timezone_set() 342, 343
date_parse() 349, 357
date_sun_info() 343, 347, 561
date_sunrise() 343, 346, 561
date_sunset() 343, 561
debug_backtrace() 647, 656
debug_print_backtrace() 648, 656
decbin() 367, 368
dechex() 367, 368
decoct() 367, 368
define() 54, 58
defined() 58, 59
deg2rad() 367, 370
die() 211
dirname() 301, 303
disk_free_space() 442
disk_total_space() 442
diskfreespace() 442
doubleval() 50
each() 126
empty() 211
end() 126
error_get_last() 648
error_log() 648, 657
error_reporting() 647, 650
eval() 52, 211, 597
exec() 540
exif_imagetype() 895, 897
exif_read_data() 895
exif_thumbnail() 895
exit() 211, 835
exp() 373, 374
explode() 304, 310
expm1() 373, 374
extension_loaded() 201, 204
extract() 118, 120
fclose() 393, 401, 414
feof() 401
fflush() 403, 415
fgetc() 401
fgetcsv() 422
fgets() 402, 405, 503, 506
fgetss() 402
file() 402, 409
file_exists() 432, 436
file_get_contents() 402, 408, 502, 503, 540
file_put_contents() 403, 412
fileatime() 432, 433
filectime() 432
filegroup() 439
fileinode() 432, 434
filemtime() 432, 433
fileowner() 439
fileperms() 439, 440
filesize() 407, 432
filetype() 432, 437
floatval() 50
flock() 403, 417
floor() 370, 372
fmod() 370, 372
fopen() 393, 396, 401, 503, 540
fpassthru() 402
fprintf() 273
fputcsv() 422
fputs() 403
fread() 402, 406, 540
fscanf() 402
fseek() 427, 429
fsockopen() 504
fstat() 433
ftell() 427
ftp_cdup() 877
ftp_chdir() 877
ftp_chmod() 877
ftp_close() 876, 877
ftp_connect() 875, 877
ftp_delete() 877
ftp_exec() 877
ftp_fget() 878
ftp_fput() 878
ftp_get() 878
ftp_login() 876, 878
ftp_mdtm() 878
ftp_mkdir() 878
ftp_nb_continue() 878
ftp_nb_fget() 878
ftp_nb_fput() 878
ftp_nb_get() 878
ftp_nb_put() 878
ftp_nlist() 879
ftp_pasv() 879
ftp_put() 879
ftp_pwd() 879
ftp_raw() 879
ftp_rawlist() 879
ftp_rename() 879
ftp_rmdir() 879
ftp_site() 879
ftp_size() 879
ftp_systype() 879
ftruncate() 403, 416
func_get_args() 192
func_num_args() 192
function_exists() 201
fwrite() 403, 410, 540
gd_info() 895
get_class() 579
get_class_methods() 579
get_class_vars() 579
get_declared_class() 579
get_declared_interfaces() 579, 640
get_defined_constants() 58, 60
get_defined_functions() 201—203
get_defined_vars() 41
get_extension_funcs() 201, 203, 204
get_headers() 463
get_html_translation_table() 255, 262
get_included_files() 106
get_loaded_extensions() 201
get_magic_quotes_gpc() 265, 555, 556
get_object_vars() 579
get_parent_class() 579
get_resource_type() 41

(продолжение рубрики см. на стр. 1006)

Функция *(продолжение)*:
getcwd() 443, 444
getdate() 349, 354
getenv() 211
gethostbyaddr() 530, 531
gethostbyname() 530
gethostbynamel() 530, 531
getimagesize() 895, 898
getrandmax() 365, 366
gettimeofday() 337, 339
gettype() 41, 42
glob() 444, 449
gmdate() 348
gmmktime() 337
gmstrftime() 349
gzclose() 450
gzcompress() 451
gzdeflate() 451
gzencode() 451
gzeof() 450
gzfile() 450
gzgetc() 450
gzgets() 450
gzgetss() 450
gzinflate() 451
gzopen() 450
gzpassthru() 450
gzputs() 451
gzread() 450
gzrewind() 451
gzseek() 451
gztell() 451
gzuncompress() 451
gzwrite() 451
header() 455, 456
headers_list() 455, 462
headers_sent() 455, 462
hebrev() 283
hebrevc() 283
hexdec() 367, 369
highlight_file() 545
highlight_string() 545
html_entity_decode() 255
htmlentities() 255, 261
htmlspecialchars() 255, 257, 543
htmlspecialchars_decode() 255, 261
http_build_query() 216
hypot() 378, 383
iconv() 285, 287
iconv_get_encoding() 289
iconv_mime_decode() 286
iconv_mime_decode_headers() 287
iconv_mime_encode() 286, 290
iconv_set_encoding() 286, 289
iconv_strlen() 287
iconv_strpos() 287
iconv_strrpos() 287
iconv_substr() 287
idate() 348
image_type_to_mime_type() 895, 899
image2wbmp() 904
imagealphablending() 925
imageantialias() 926
imagearc() 917
imagechar() 930
imagecharup() 930
imagecolorallocate() 910
imagecolorallocatealpha() 910
imagecolorat() 913
imagecolorclosest() 910
imagecolorclosestalpha() 910
imagecolorclosesthwb() 910
imagecolordeallocate() 910
imagecolorexact() 910
imagecolorexactalpha() 910
imagecolormatch() 911
imagecolorresolve() 910
imagecolorresolvealpha() 910
imagecolorset() 911
imagecolorsforindex() 910, 913
imagecolorstotal() 911
imagecolortransparent() 911
imagecopy() 905
imagecopymerge() 905
imagecopymergegray() 905
imagecopyresampled() 906, 908
imagecopyresized() 906
imagecreate() 901
imagecreatefromgd() 902
imagecreatefromgd2() 902
imagecreatefromgd2part() 902
imagecreatefromgif() 901
imagecreatefromjpeg() 901
imagecreatefrompng() 901
imagecreatefromstring() 902
imagecreatefromwbmp() 902
imagecreatefromxbm() 902
imagecreatefromxpm() 902
imagecreatetruecolor() 901
imagedashedline() 917
imagedestroy() 902
imageellipse() 917
imagefill() 918
imagefilledarc() 917
imagefilledellipse() 918, 921
imagefilledpolygon() 917
imagefilledrectangle() 917, 923
imagefilltoborder() 918
imagefilter() 906
imagefontheight() 928
imagefontwidth() 928
imageftbbox() 928
imagefttext() 929
imagegammacorrect() 911
imagegd() 904
imagegd2() 904
imagegif() 903
imageinterlace() 901
imageistruecolor() 909
imagejpeg() 903
imagelayereffect() 926
imageline() 917
imageloadfont() 929
imagepalettecopy() 911
imagepng() 904
imagepolygon() 917
imagepsbbox() 928
imagepscopyfont() 929
imagepsencodefont() 929
imagepsextendfont() 929
imagepsfreefont() 929
imagepsloadfont() 929
imagepsslantfont() 930
imagepstext() 929
imagerectangle() 917
imagerotate() 906
imagesavealpha() 902
imagesetbrush() 926
imagesetpixel() 917
imagesetstyle() 926
imagesetthickness() 926
imagesettile() 926
imagestring() 930
imagestringup() 930

imagesx() 895
imagesy() 895
imagetruecolortopalette() 909, 914, 915
imagettfbbox() 928
imagettftext() 929
imagetypes() 895
imagewbmp() 904
imagexbm() 904
implode() 304, 310
in_array() 142
ini_get() 564
ini_get_all() 564
ini_restore() 564
ini_set() 564
interface_exists() 579, 641
intval() 50, 51, 64
ip2long() 530
is_a() 579
is_array() 41
is_bool() 41
is_callable() 41, 580
is_dir() 432, 436
is_double() 42, 44, 384
is_executable() 440, 442
is_file() 432, 436
is_finite() 384, 387
is_float() 42, 384, 385
is_infinite() 384, 387
is_int() 42, 43, 44, 383, 384
is_integer() 42, 383
is_link() 433, 436
is_long() 42, 384
is_nan() 384, 386
is_null() 42
is_numeric() 42, 44, 242, 384, 385
is_object() 42, 580
is_readable() 439, 442
is_real() 42, 384
is_resource() 42, 45
is_scalar() 42
is_string() 42
is_subclass_of() 580
is_uploaded_file() 400
is_writable() 440, 442
is_writeable() 440
isset() 211
join() 304
jpeg2wbmp() 904
key() 126
krsort() 150, 154
ksort() 150, 153
lcg_value() 365, 366
lchgrp() 439
lchown() 439
levenshtein() 294, 297
link() 430
linkinfo() 430, 432
localeconv() 266, 269
localtime() 349, 359
log() 373, 376
log10() 373, 376
log1p() 373
long2ip() 530
lstat() 433
ltrim() 249
mail() 885
max() 363
mb_convert_encoding() 285
mcrypt_ecb() 501
md5() 495
md5_file() 495
memory_get_usage() 565
metaphone() 495
method_exists() 580
microtime() 336, 337
min() 363
mkdir() 443, 445
mktime() 336, 338
money_format() 274, 279
move_uploaded_file() 400
mt_getrandmax() 365
mt_rand() 365
mt_srand() 365
mysql_close() 830, 832
mysql_connect() 830, 831
mysql_data_seek() 867
mysql_erorr() 835
mysql_fetch_array() 836, 840, 849
mysql_fetch_assoc() 836, 839, 841
mysql_fetch_object() 836, 842
mysql_fetch_row() 836, 837, 841
mysql_insert_id() 865
mysql_num_fields() 844
mysql_num_rows() 844, 845
mysql_ping() 830
mysql_query() 835, 836, 837
mysql_real_escape_string() 865
mysql_result() 836, 837
mysql_select_db() 833
natcasesort() 150
natsort() 150, 154
next() 126
nl_langinfo() 266, 270
nl2br() 255, 256, 543
number_format() 274, 278
ob_clean() 482
ob_end_clean() 482
ob_end_flush() 482
ob_flush() 482
ob_get_clean() 482, 487
ob_get_contents() 482
ob_get_flush() 482
ob_get_lenght() 482
ob_get_length() 484
ob_get_level() 482, 487
ob_get_status() 482
ob_gzhandler() 482, 491
ob_hyperlink_handler() 488
ob_iconv_handler() 287, 491
ob_implicit_flush() 483
ob_list_handlers() 483, 489
ob_start() 482
octdec() 367, 369
opendir() 444, 445
ord() 237, 239
output_add_rewrite_var() 483, 492
output_reset_rewrite_vars() 483, 492
pack() 298
parse_ini_file() 424
parse_str() 305
parse_url() 216, 217
pathinfo() 301
php_sapi_name() 565
php_uname() 211, 213, 565
phpinfo() 211, 565, 953
phpversion() 211, 212, 565
pi() 378
png2wbmp() 904
pos() 126
pow() 373
preg_grep() 318

(продолжение рубрики см. на стр. 1008)

Функция *(продолжение)*:
preg_match() 317, 318
preg_match_all() 318, 323
preg_quote() 318, 334
preg_replace() 318, 326
preg_replace_callback() 318, 330
preg_split() 318, 332
prev() 126
print() 63, 273, 274
print_r() 110, 211, 576, 580, 841
printf() 273, 274
property_exits() 580
put_get_contents() 540
quoted_printable_decode() 264
quotemeta() 264
rad2deg() 367, 370
rand() 147, 365
range() 114, 115
rawurldecode() 216
rawurlencode() 216
readdir() 444, 445
readfile() 403
readgzfile() 451, 453
readlink() 430
realpath() 301, 302
register_shutdown_function() 206, 210
register_tick_function() 206, 209
rename() 398, 399
reset() 126
resizeimg() 906
restore_error_handler() 647
restore_exception_handler() 647, 655
rewind() 427
rewinddir() 444
rmdir() 443
round() 370, 371
rsort() 150, 152
rtrim() 249
scandir() 444, 448
serialize() 298, 300, 610
session_cache_expire() 466, 467
session_cache_limiter() 466
session_destroy() 475, 476
session_get_cookie_params() 475
session_id() 474, 477
session_is_registered() 474
session_name() 474
session_regenerate_id() 474
session_register() 474
session_save_path() 474
session_set_save_handler() 477
session_start() 474, 475, 477
session_unregister() 474
set_error_handler() 647, 652
set_exception_handler() 647, 688
setcookie() 468
setlocale() 266, 268
settype () 50
settype() 50
sha1() 495
sha1_file() 495
shell_exec() 540, 542
shuffle() 148, 149
similar_text() 294, 296
sin() 378, 379
sinh() 378, 379
sizeof() 145
sleep() 211
sort() 150, 151
soundex() 495
sprintf() 273
sqrt() 373, 375
srand() 365, 366
sscanf() 305, 311
stat() 433, 437
str_ireplace() 248
str_pad() 304, 305
str_repeat() 303, 305
str_replace() 248, 249
str_rot13() 495
str_shuffle() 237, 240
str_split() 306
str_word_count() 304, 308
strcasecmp() 293
strchr() 244
strcmp() 292, 294
strcoll() 292
strcspn() 293
strftime() 349, 352
strip_tags() 255, 263
stripcslashes() 264
stripos() 243
stripslashes() 264
stristr() 244, 247
strlen() 237, 238
strnatcasecmp() 293
strnatcmp() 293, 295
strncasecmp() 293
strncmp() 293
strpbrk() 244, 248
strpos() 243, 245
strptime() 349, 356
strrchr() 244, 247
strrev() 237, 240
strripos() 243
strrpos() 243, 246
strspn() 293
strstr() 243, 246, 507
strtok() 304, 307
strtolower () 254
strtotime() 337, 341
strtoupper() 254
strtr() 248, 249, 251
strval() 50
substr() 243, 244
substr_compare() 293
substr_count() 243, 244
substr_replace() 248, 250
symlink() 430, 431
system() 540
tan() 378, 380
tanh() 378, 381
tempnam() 393, 397
time() 336, 337
time_nanosleep() 211
timezone_abbreviations_list() 343, 344
timezone_identifiers_list() 342, 344
timezone_name_from_abbr() 343, 346
tmpfile() 393
touch() 393, 395, 432, 434
trigger_error() 648, 650
trim() 249, 252
uasort() 150
ucfirst() 254
ucwords() 254
uksort() 150
umask() 439
unlink() 398, 399
unpack() 298

unregister_tick_function() 206
unserialize() 298, 300, 610
unset() 211
urldecode() 216
urlencode() 216, 848
user_error() 648
usleep() 211
usort() 150
var_export() 595
vfprintf() 273
vprintf() 273
vsprintf() 273
wordwrap() 305, 311
zlib_get_coding_type() 451
анонимная 199
вложенная 197
динамическая 198
необязательный параметр 190
неявное выполнение 206
объявление 185
параметры 188
проверка существования 201
рекурсивная 196
Функция SQL:
ABS() 792
ACOS() 792
ADDDATE() 796
ADDTIME() 796
AES_DECRYPT() 813, 814
AES_ENCRYPT() 813, 814
ASCII() 805
ASIN() 792
ATAN() 792
ATAN2() 792
AVG() 816, 817
BIN() 805
BIT_AND() 816
BIT_LENGTH() 805
BIT_OR() 816
BIT_XOR() 816
CEIL() 792
CEILING() 792, 795
CHAR() 805
CHAR_LENGTH() 805
CHARSET() 805
COALESCE() 792
COLLATION() 805
COMPRESS() 805
CONCAT() 806, 811
CONCAT_WS() 806
CONV() 806
CONVERT() 806, 809
CONVERT_TZ() 797
COS() 792
COT() 792
COUNT() 782, 816, 820
CRC32() 792
CURDATE() 797
CURRENT_TIME() 797, 798
CURRENT_TIMESTAMP() 798
CURTIME() 797
DATE() 797
DATE_ADD() 796
DATE_FORMAT() 797, 801
DATE_SUB() 799
DATEDIFF() 797
DAY() 797
DAYNAME() 797
DAYOFMONTH() 797
DAYOFWEEK() 797
DAYOFYEAR() 797
DECODE() 813
DEFAULT() 827
DEGREES() 792
DES_DECRYPT() 813
DES_ENCRYPT() 813
ELT() 806
ENCODE() 813
ENCRYPT() 814
EXP() 792
EXPORT_SET() 806
EXTRACT() 797
FIELD() 806
FIND_IN_SET() 806
FLOOR() 792, 796, 803
FORMAT() 806
FROM_DAYS() 797
FROM_UNIXTIME() 797, 804
GET_FORMAT() 798
GET_LOCK() 827
GREATEST() 792
GROUP_CONCAT() 817, 823
HEX() 806
HOUR() 798
INET_ATON() 827, 828
INET_NTOA() 827, 829
INSERT() 807
INSTR() 807
INTERVAL() 792
IS_FREE_LOCK() 827
IS_USED_LOCK() 828
LAST_DAY() 798
LCASE() 807
LEAST() 792
LEFT() 807
LENGTH() 807
LN() 792
LOAD_FILE() 807
LOCALTIME 798
LOCALTIME() 798
LOCALTIMESTAMP 798
LOCALTIMESTAMP() 798
LOCATE() 807
LOG() 792
LOG10() 792
LOG2() 792
LOWER() 807, 812
LPAD() 807
LTRIM() 807
MAKE_SET() 807
MAKEDATE() 798
MAKETIME() 798
MAX() 817
MD5() 814, 815
MICROSECOND() 798
MID() 807
MIN() 817
MINUTE() 798
miscellaneous 827
MOD() 792
MONTH() 798
MONTHNAME() 798
NAME_CONST() 828
NOW() 767, 798
OCT() 808
OCTET_LENGTH() 807
OLD_PASSWORD() 814
ORD() 808
PASSWORD() 814
PERIOD_ADD() 798
PERIOD_DIFF() 798
PI() 793
POSITION() 807

(окончание рубрики см. на стр. 1010)

Функция SQL *(окончание)*:
POW() 793
POWER() 793
QUARTER() 799
QUOTE() 808
RADIANS() 793, 794
RAND() 793
RELEASE_LOCK() 828
REPEAT() 808
REPLACE() 808
REVERSE() 808
RIGHT() 808
ROUND() 793, 795
RPAD() 808
RTRIM() 808
SEC_TO_TIME() 799
SECOND() 799
SHA1() 814
SIGN() 793
SIN() 793, 794
SLEEP() 828
SOUNDEX() 808
SPACE() 808
SQRT() 793
STD() 817
STDDEV() 817
STDDEV_POP() 817
STDDEV_SAMP() 817
STR_TO_DATE() 799
SUBDATE() 799
SUBSTRING() 807, 810
SUBSTRING_INDEX() 808
SUBTIME() 799
SUM() 817, 826
TAN() 793
TIME() 797, 799
TIME_FORMAT() 799
TIME_TO_SEC() 800
TIMEDIFF() 799
TIMESTAMP() 799
TIMESTAMPADD() 799
TIMESTAMPDIFF() 799
TO_DAYS() 800, 803
TRIM() 809
TRUNCATE() 793, 796
UCASE() 809
UNCOMPRESS() 809
UNCOMPRESSED_LENCTH() 809
UNHEX() 809
UNIX_TIMESTAMP() 800
UPPER() 809, 812
UTC_DATE() 800
UTC_TIME() 800
UTC_TIMESTAMP() 800
UUID() 828
VAR_POP() 817
VAR_SAMP() 817
VARIANCE() 817
WEEK() 800
WEEKDAY() 800
WEEKOFYEAR() 800
YEAR() 800
YEARWEEK() 801
агрегатная 816
календарная 796
конструкции GROUP BY 816
математическая 791
строковая 805
суммирующая 816
шифрования 812

Ч

Число:
вещественное 30
научная запись 31
целое 29
Член:
класса 568
проверка существования 588
уничтожение 589

Ш

Шифрование:
необратимое 494, 814, 815
обратимое 501, 814

Э

Экранирование 33, 264, 555