Самоучитель

Максим Кузнецов, Игорь Симдянов

# PHP 5/6

*Третье издание*

Объектно-ориентированное программирование

Работа с базами данных

Защита приложений от взлома

Актуальные примеры

Новое в PHP 5/6

bhv

**Максим Кузнецов**
**Игорь Симдянов**

# Самоучитель PHP 5/6

*Третье издание*

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.06
ББК 32.973.26-018.2
К89

**Кузнецов, М. В.**

К89 Самоучитель PHP 5/6 / М. В. Кузнецов, И. В. Симдянов —
3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 672 с.: ил.

ISBN 978-5-9775-0409-6

Описаны самые последние версии языка разработки серверных сценариев PHP — 5.3 и 6.0. Рассмотрены основы языка, вопросы объектно-ориентированного программирования на PHP, обработки исключительных ситуаций, взаимодействия с MySQL, регулярные выражения, работа с электронной почтой. Книга содержит множество примеров, взятых из реальной практики разработки динамических Web-сайтов.

Третье издание книги, ранее выходившей под названием "Самоучитель PHP 5", существенно переработано, дополнено и будет интересно не только программистам, впервые знакомящимся с языком, но и читателям предыдущих изданий книги и профессионалам.

*Для программистов и Web-разработчиков*

УДК 681.3.06
ББК 32.973.26-018.2

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Евгений Рыбаков* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Анна Кузьмина* |
| Компьютерная верстка | *Натальи Смирновой* |
| Корректор | *Наталия Першакова* |
| Оформление обложки | *Елены Беляевой* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 03.12.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 54,18.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.002108.02.07
от 28.02.2007 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 978-5-9775-0409-6

© Кузнецов М. В., Симдянов И. В., 2008
© Оформление, издательство "БХВ-Петербург", 2008

# Оглавление

**Введение** ..... **1**

Нововведения PHP 6 ..... 2
Благодарности ..... 2

**Глава 1. Что представляет собой PHP?** ..... **3**

1.1. История PHP ..... 3
1.2. Место и роль PHP в Интернете ..... 5
1.2.1. Серверные технологии ..... 6
UNIX-подобная операционная система ..... 6
Web-сервер ..... 7
Серверный язык ..... 7
Файлы и базы данных ..... 8
Электронная почта ..... 9
1.2.2. Клиентские технологии ..... 9
Web-браузеры, HTML ..... 10
Каскадные таблицы стилей CSS и XML ..... 10
Flash-ролики ..... 11
FTP-клиенты ..... 11
Удаленный доступ к серверу. Протокол SSH ..... 12

**Глава 2. Быстрый старт** ..... **13**

2.1. Скрипты ..... 13
2.2. Начальные и конечные теги ..... 16
2.3. Использование точки с запятой ..... 18
2.4. Составные выражения. Фигурные скобки ..... 19
2.5. Комментарии ..... 21

**Глава 3. Переменные и типы данных ..... 23**

3.1. Объявление переменной. Оператор = ..... 23
3.2. Типы данных ..... 24
3.3. Целые числа ..... 25
3.4. Вещественные числа ..... 27
3.5. Строки ..... 28
3.6. Кавычки ..... 28
3.7. Оператор <<< ..... 32
3.8. Обращение к неинициализированной переменной. Замечания (Notice) ..... 32
3.9. Специальный тип *NULL* ..... 34
3.10. Логический тип ..... 35
3.11. Уничтожение переменной. Конструкция *unset()* ..... 36
3.12. Проверка существования переменной. Конструкции *isset()* и *empty()* ..... 36
3.13. Определение типа переменной ..... 38
3.14. Неявное приведение типов ..... 44
3.15. Явное приведение типов ..... 46
3.16. Динамические переменные ..... 51

**Глава 4. Константы ..... 53**

4.1. Объявление константы. Функция *define()* ..... 53
4.2. Функции для работы с константами ..... 57
4.3. Динамически константы. Функция *constant()* ..... 58
4.4. Проверка существования константы ..... 59
4.5. Предопределенные константы ..... 60

**Глава 5. Операторы и конструкции языка ..... 63**

5.1. Объединение строк. Оператор "точка" ..... 63
5.2. Конструкция *echo*. Оператор "запятая" ..... 64
5.3. Арифметические операторы ..... 65
5.4. Поразрядные операторы ..... 70
5.5. Операторы сравнения ..... 75
5.6. Условный оператор *if* ..... 79
5.7. Логические операторы ..... 81
5.8. Условный оператор *x ? y : z* ..... 89
5.9. Переключатель *switch* ..... 90
5.10. Цикл *while* ..... 95
5.11. Цикл *do ... while* ..... 101

5.12. Цикл *for* ... 102
5.13. Включение файлов ... 107
5.14. Подавление вывода ошибок. Оператор @ ... 113
5.15. Приоритет выполнения операторов ... 114

**Глава 6. Массивы ... 117**

6.1. Создание массива ... 117
6.2. Ассоциативные и индексные массивы ... 124
6.3. Многомерные массивы ... 129
6.4. Интерполяция элементов массива в строки ... 130
6.5. Конструкция *list()* ... 131
6.6. Обход массива ... 134
6.7. Цикл *foreach* ... 138
6.8. Проверка существования элементов массива ... 140
6.9. Количество элементов в массиве ... 144
6.10. Сумма элементов массива ... 146
6.11. Случайные элементы массива ... 147
6.12. Сортировка массивов ... 149
6.13. Суперглобальные массивы. Массив *$_SERVER* ... 159
6.13.1. Элемент *$_SERVER['DOCUMENT_ROOT']* ... 159
6.13.2. Элемент *$_SERVER['HTTP_REFERER']* ... 160
6.13.3. Элемент *$_SERVER['HTTP_USER_AGENT']* ... 161
6.13.4. Элемент *$_SERVER['REMOTE_ADDR']* ... 161
6.13.5. Элемент *$_SERVER['SCRIPT_FILENAME']* ... 162
6.13.6. Элемент *$_SERVER['SERVER_NAME']* ... 162
6.13.7. Элемент *$_SERVER['QUERY_STRING']* ... 163
6.13.8. Элемент *$_SERVER['PHP_SELF']* ... 164

**Глава 7. Функции ... 165**

7.1. Объявление и вызов функции ... 165
7.2. Параметры функции ... 168
7.3. Передача параметров по значению и ссылке ... 169
7.4. Необязательные параметры ... 170
7.5. Переменное количество параметров ... 172
7.6. Глобальные переменные ... 174
7.7. Статические переменные ... 175
7.8. Возврат массива функцией ... 176
7.9. Рекурсивные функции ... 177

7.10. Вложенные функции ... 179
7.11. Динамическое имя функции ... 179
7.12. Анонимные функции ... 180
7.13. Проверка существования функции ... 182
7.14. Неявное выполнение функций. Оператор *declare()* ... 188
7.15. Вспомогательные функции ... 193

**Глава 8. Взаимодействие PHP с HTML ... 197**

8.1. Передача параметров методом GET ... 197
8.2. HTML-форма и ее обработчик ... 202
8.3. Текстовое поле ... 207
8.4. Поле для приема пароля ... 208
8.5. Текстовая область ... 209
8.6. Скрытое поле ... 210
8.7. Флажок ... 211
8.8. Список ... 213
8.9. Переключатель ... 215
8.10. Загрузка файла на сервер ... 217

**Глава 9. Строковые функции ... 221**

9.1. Функции для работы с символами ... 221
9.2. Поиск в строке ... 225
9.3. Замена в тексте ... 231
9.4. Преобразование регистра ... 237
9.5. Работа с HTML-кодом ... 238
9.6. Экранирование ... 247
9.7. Форматный вывод ... 250
9.8. Преобразование кодировок ... 256
9.9. Сравнение строк ... 259
9.10. Хранение данных ... 265
9.11. Работа с путями к файлам и каталогами ... 269
9.12. Объединение и разбиение строк ... 271

**Глава 10. Регулярные выражения ... 283**

10.1. Как изучать регулярные выражения? ... 283
10.2. Синтаксис регулярных выражений ... 284

10.3. Функции для работы с регулярными выражениями ..... 288
10.4. Функции *preg_match()* ..... 289
10.5. Функция *preg_match_all()* ..... 294
10.6. Функция *preg_replace()* ..... 297
10.7. Функция *preg_replace_callback()* ..... 302
10.8. Функция *preg_split()* ..... 304
10.9. Функция *preg_quote()* ..... 306

**Глава 11. Дата и время ..... 309**

11.1. Формирование даты и времени ..... 309
11.2. Географическая привязка ..... 316
11.3. Форматирование даты и времени ..... 322

**Глава 12. Математические функции ..... 337**

12.1. Предопределенные константы ..... 337
12.2. Поиск максимума и минимума ..... 338
12.3. Генерация случайных чисел ..... 340
12.4. Преобразование значений между различными системами счисления ..... 342
12.5. Округление чисел ..... 346
12.6. Логарифмические и степенные функции ..... 349
12.7. Тригонометрические функции ..... 353
12.8. Информационные функции ..... 355

**Глава 13. Файлы и каталоги ..... 363**

13.1. Создание файлов ..... 363
13.2. Манипулирование файлами ..... 370
13.3. Чтение и запись файлов ..... 373
13.3.1. Чтение файлов ..... 376
13.3.2. Запись файлов ..... 383
13.3.3. Обязательно ли закрывать файлы? ..... 387
13.3.4. Дозапись файлов ..... 389
13.3.5. Блокировка файлов ..... 390
13.3.6. Прямое манипулирование файловым указателем ..... 395
13.4. Права доступа ..... 399
13.5. Каталоги ..... 403

**Глава 14. HTTP-заголовки ..... 411**

14.1. Функции для управления HTTP-заголовками ..... 412
14.2. Кодировка страницы ..... 414
14.3. HTTP-коды состояния ..... 415
14.4. Список HTTP-заголовков ..... 416
14.5. Подавление кэширования ..... 419

**Глава 15. Cookie ..... 425**

**Глава 16. Сессии ..... 431**

**Глава 17. Электронная почта ..... 437**

17.1. Отправка почтового сообщения ..... 437
17.2. Рассылка писем ..... 439

**Глава 18. Объектно-ориентированные возможности PHP ..... 441**

18.1. Введение в объектно-ориентированное программирование ..... 441
18.2. Создание класса ..... 443
18.3. Создание объекта ..... 443
18.4. Инкапсуляция. Спецификаторы доступа ..... 445
18.5. Методы класса. Член *$this* ..... 447
18.6. Специальные методы класса ..... 451
18.7. Функции для работы с методами и классами ..... 452
18.8. Конструктор. Метод *__construct()* ..... 454
18.9. Параметры конструктора ..... 457
18.10. Деструктор. Метод *__destruct()* ..... 459
18.11. Автозагрузка классов. Функция *__autoload()* ..... 460
18.12. Аксессоры. Методы *__set()* и *__get()* ..... 461
18.13. Проверка существования члена класса. Метод *__isset()* ..... 463
18.14. Уничтожение члена класса. Метод *__unset()* ..... 464
18.15. Динамические методы. Метод *__call()* ..... 466
18.16. Интерполяция объекта. Метод *__toString()* ..... 468
18.17. Наследование ..... 470

18.18. Спецификаторы доступа и наследование ..... 473
18.19. Перегрузка методов ..... 476
18.20. Полиморфизм ..... 478
18.21. Абстрактные классы ..... 480
18.22. Абстрактные методы ..... 481
18.23. Создание интерфейса ..... 483
18.24. Реализация нескольких интерфейсов ..... 485
18.25. Наследование интерфейсов ..... 486
18.26. Статические члены класса ..... 487
18.27. Статические методы класса ..... 490
18.28. Константы класса ..... 491
18.29. Предопределенные константы ..... 493
18.30. *Final*-методы класса ..... 494
18.31. *Final*-классы ..... 496
18.32. Клонирование объекта ..... 497
18.33. Управление процессом клонирования. Метод *__clone()* ..... 498
18.34. Управление сериализацией. Методы *__sleep()* и *__wakeup()* ..... 500
18.35. Синтаксис исключений ..... 509

**Глава 19. Работа с СУБД MySQL ..... 513**

19.1. Введение в СУБД и SQL ..... 514
19.2. Первичные ключи ..... 517
19.3. Создание и удаление базы данных ..... 519
19.4. Выбор базы данных ..... 521
19.5. Типы данных ..... 523
19.6. Создание и удаление таблиц ..... 529
19.7. Вставка числовых значений в таблицу ..... 536
19.8. Вставка строковых значений в таблицу ..... 538
19.9. Вставка календарных значений ..... 540
19.10. Вставка уникальных значений ..... 543
19.11. Механизм *AUTO_INCREMENT* ..... 544
19.12. Многострочный оператор *INSERT* ..... 544
19.13. Удаление данных ..... 545
19.14. Обновление записей ..... 547
19.15. Выборка данных ..... 549
19.16. Условная выборка ..... 551
19.17. Псевдонимы столбцов ..... 558
19.18. Сортировка записей ..... 558
19.19. Вывод записей в случайном порядке ..... 561
19.20. Ограничение выборки ..... 562

19.21. Вывод уникальных значений ... 563
19.22. Объединение таблиц ... 565

**Глава 20. Взаимодействие MySQL и PHP ... 569**

20.1. Функция *mysql_connect()* ... 569
20.2. Функция *mysql_close()* ... 571
20.3. Функция *mysql_select_db()* ... 572
20.4. Функция *mysql_query()* ... 573
20.5. Функция *mysql_result()* ... 575
20.6. Функция *mysql_fetch_row()* ... 576
20.7. Функция *mysql_fetch_assoc()* ... 577
20.8. Функция *mysql_fetch_array()* ... 580
20.9. Функция *mysql_fetch_object()* ... 582
20.10. Функция *mysql_num_rows()* ... 583

**Заключение ... 587**

Online-поддержка ... 588
Портал по программированию *SoftTime.ru* ... 588
Портал *Softtime.org* ... 590
Сайт *Softtime.biz* ... 590

**ПРИЛОЖЕНИЯ ... 593**

**Приложение 1. Установка и настройка PHP, Web-сервера Apache и MySQL-сервера ... 595**

П1.1. Где взять дистрибутивы? ... 595
П1.1.1. Дистрибутив PHP ... 596
П1.1.2. Дистрибутив Apache ... 597
П1.1.3. Дистрибутив MySQL ... 598
П1.2. Установка Web-сервера Apache под Windows ... 599
П1.3. Установка Web-сервера Apache под Linux ... 601
П1.4. Настройка виртуальных хостов ... 602
П1.5. Настройка кодировки по умолчанию ... 606
П1.6. Управление запуском и остановкой Web-сервера Apache ... 607

П1.7. Управление Apache из командной строки ..... 608
П1.8. Установка PHP под Windows ..... 609
П1.8.1. Установка PHP в качестве модуля ..... 609
П1.8.2. Установка PHP как CGI-приложения ..... 610
П1.9. Установка PHP под Linux ..... 612
П1.10. Общая настройка конфигурационного файла php.ini ..... 613
П1.11. Настройка и проверка работоспособности расширений PHP ..... 616

**Приложение 2. Установка MySQL ..... 618**

П2.1. Установка MySQL под Windows ..... 618
П2.1.1. Процесс установки ..... 618
П2.1.2. Постинсталляционная настройка ..... 624
П2.1.3. Проверка работоспособности MySQL ..... 631
П2.2. Установка MySQL под Linux ..... 634
П2.3. Конфигурационный файл ..... 637

**Предметный указатель ..... 641**

# Введение

PHP является молодым и динамично развивающимся языком программирования, который используется главным образом для создания Web-приложений. Быстрая динамика развития, 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 при разработке 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.

Что-то из заявленного в PHP 6 в настоящий момент реализовано, что-то еще ожидает своего часа. Тем не менее, язык меняется и книги, написанные ранее, устаревают. Поэтому нами было принято решение выпустить третье издание самоучителя PHP, которое описывает современное состояние дел.

Книга будет интересна как начинающим разработчикам, которым необходимо последовательное изложение языка, так и опытным программистам, желающим познакомиться с нововведениями языка, появившимися в последнее время. Текущее издание будет также интересно читателям предыдущих изданий самоучителя, т. к. нами было принято решение о полной переработке всех глав: по сути, они написаны заново. Связано это с тем, что структура и организация языка значительно изменились за последние годы, поменялась и аудитория PHP-разработчиков.

# Нововведения PHP 6

Как уже отмечалось, не все нововведения пока реализованы: на данный момент нет сечений массивов и строк, отсутствует (в стабильных версиях) поддержка Unicode на уроне ядра, в расширении для работы с графикой не поддерживается работа с анимированными GIF-файлами. Список этот можно 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).

# Благодарности

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

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

## Глава 1

# Что представляет собой PHP?

Язык программирования PHP является универсальным, т. е. может использоваться для создания практически любых программ. Однако наибольшее распространение он получил в области Web-разработки. Львиная доля приложений, созданных с применением PHP, обслуживает Web-сайты.

Язык PHP является интерпретируемым языком программирования. Это означает, что программа выполняется строчка за строчкой, а не компилируется в исполняемый модуль. С одной стороны это приводит к тому, что программы выполняются более медленно по сравнению с компилируемыми языками, с другой стороны исходный код приложения может быть отредактирован в любой удобный момент.

В этой главе обсуждается эволюция языка, мы посмотрим, что представляет собой PHP на сегодняшний день, а также взаимодействие PHP с другими технологиями и языками программирования.

## 1.1. История PHP

Датой создания языка PHP считается осень 1994 года, а автором языка — программист Расмус Лердорф (Rasmus Lerdorf). Сначала Лердорф собрался написать простой движок для своей персональной странички и завершил эту работу к началу 1995 года. Движок был написан на языке Perl и умел делать очень немного, т. к. создавался только для подсчета количества посетителей домашней странички Расмуса, на которой было размещено его резюме. Этот движок был назван Personal Home Page Tools (PHPT), а позже название трансформировалось в аббревиатуру PHP, что следует понимать как рекурсивный акроним Hypertext Preprocessor (такое решение о переименовании было принято в 1997 году).

Стоит отметить, что в 1994 году специальных инструментов для создания Web-приложений не существовало, да и сам Web только начинал свое развитие. Поэтому движок, разработанный Расмусом, вызвал живейший интерес разработчиков, и к нему хлынул поток писем с просьбами предоставить свой инструментарий. Такой

успех PHP привел к тому, что Лердорф приступил к разработке различных расширений языка. В том же 1994 году Расмус разработал пакет, предназначенный для обработки HTML-форм, который назывался FI (Form Interpretator). А к середине 1995 года, объединив движок для своей странички с интерпретатором форм, Лердорф выпустил вторую версию языка, которая называлась PHP/FI. К этому времени при разработке языка Расмус перешел на компилируемый язык C (Perl является интерпретируемым и значительно уступает по производительности C, одному из самых быстрых языков программирования). Тогда же в PHP была добавлена поддержка основных баз данных, что еще более усилило популярность вновь созданного языка.

***Замечание***

И по сегодняшний день расширения являются одним из основных средств увеличения функциональности PHP. Существуют сотни расширений (для работы с базами данных, графикой, PDF-файлами и т. п.), и любой разработчик может самостоятельно создать собственное расширение для своих нужд. В стандартной поставке PHP содержит лишь наиболее популярные и проверенные расширения.

К концу 1997 года два программиста — Зив Сураски (Zeev Suraski) и Энди Гутманс (Andi Gutmans) — переписали первоначальный лексический анализатор, и к лету 1998 года в полной мере увидела свет третья версия языка — PHP 3. Развитие PHP стремительно продолжалось, в язык сотнями добавлялись новые функции, и в 1999 году количество разработчиков, использующих PHP, превысило 1 миллион, что сделало PHP одним из самых популярных языков для разработки Web-приложений. К этому времени к разработке языка подключилось большое количество программистов со всего мира. Стало понятно, что вследствие все более растущей популярности языка необходима его адаптация для разработки крупномасштабных приложений. Дело в том, что поскольку PHP изначально проектировался для разработки несложных приложений, лексический анализ и компиляция кода в нем происходили одновременно. Этим достигалась быстрота выполнения простых сценариев за счет сокращения времени от запуска до начала выполнения. Однако при выполнении более сложных сценариев все происходило с точностью до наоборот по причине многократного повторения лексического анализа кода. Стало очевидным, что необходимо модифицировать основное ядро, что и было быстро и успешно осуществлено фирмой Zend Technologies Ltd. Был создан более устойчивый лексический анализатор, на базе которого уже можно было строить полномасштабные приложения, и в 2000 году появилась четвертая версия языка — PHP 4.0. В этой версии уже существовала возможность создания объектно-ориентированных приложений, что было с радостью воспринято многими пользователями. Однако объектно-ориентированная парадигма находилась в PHP 4.0 в зачаточном состоянии, поскольку в языке отсутствовала реализация многих концепций, свойственных объектно-ориентированной технологии. В 2004 году выходит новая версия языка — PHP 5.0 на базе машины Zend Engine 2. Основные изменения в новой версии как

раз коснулись реализации объектно-ориентированного подхода. Практически сразу после выхода версии PHP 5.0 разработчики приняли план по созданию PHP 6.0. Изменения было решено вводить не революционно, а постепенно добавляя их в новые релизы PHP: PHP 5.1, PHP 5.2 и PHP 5.3. Одновременно изменения добавляются в версию PHP 6.0, которая пока не стабильна и предназначена лишь для тестирования. Таким образом, когда все изменения будут внедрены в движок языка и отлажены, будет выпущена конечная версия PHP 6.0.

Итак, что представляет собой PHP на сегодняшний день? PHP сейчас это:

- поддержка платформ Win32 (9*x*/NT/2000/XP), UNIX, OS/2, QNX, MacOS, BeOS, OCX;
- совместимость с серверами: Apache (Win32, UNIX), phttpd, fhttpd, thttpd, ISAPI (Zeus, IIS), NSAPI, модулем Roxen/Caudium, AOLServer;
- поддержка технологий COM, XML, Java, CORBA, WDDX, Macromedia Flash;
- развитая функциональность для работы с сетевыми соединениями;
- поддержка свыше 20 баз данных и развитая функциональность для работы с ними;
- возможность создания полноценных объектно-ориентированных приложений;
- сравнительно простой синтаксис и удобство в практическом использовании;
- бесплатность;
- открытость кода, благодаря которой вы можете создавать собственные расширения языка.

## 1.2. Место и роль PHP в Интернете

На сегодняшний день PHP используют 20 млн сайтов. При помощи PHP 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. 1.1 представлена схема, демонстрирующая место и время действия компонентов типичной среды Web-разработки.

Как видно из схемы, помимо PHP в работе Web-приложений работает 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. 1.1.

**Рис. 1.1.** Клиент-серверное взаимодействие

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 по протоколу HTTP на сервер.

## 1.2.1. Серверные технологии

### UNIX-подобная операционная система

Все программы, 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, по крайней мере, в

Российской Федерации в качестве операционной системы используют 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 кросс-платформенную 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. Существуют 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. Помимо того, что он 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, управление которыми возможно 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 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 также как и Web-сервер Apache, PHP имеет ядро, где сосредоточены базовые функции и множество подключаемых 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 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 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 эксплуатировались значительно более длительное время). 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 Phyton.

## Файлы и базы данных

Файлы играют значительную роль в любом языке программирования или технологии, 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 профессионалами 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 с коммерческих 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, различные базы данных имеют разные 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).

## 1.2.2. 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 интерпретируется 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, Opera и Chrome. 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 своего собственного словаря. В настоящий момент уже разработано огромное число словарей XML, позволяющих описывать любую информацию — от химических реакций (CML, Chemical Markup Language) до финансовых операций (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 мини-сайт, 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.

## 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. В среде 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 того, что интерпретатор PHP и Web-сервер установлены у вас на локальной машине и связаны таким образом, что скрипты, помещенные в файл с расширение 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>
```

В результате работы скрипта из листинга 2.1 в окно браузера будет выведена фраза "Hello world!" (рис. 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.

**Рис. 2.1.** Результат работы скрипта из листинга 2.1

### *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 тегов, представленных в табл. 2.1.

> ***Замечание***
>
> До версии PHP 6.0 поддерживались начальные и конечные теги, характерные для ASP-скриптов, `<%` и `%>`, включить которые можно было в конфигурационном файле php.ini, назначив директиве `asp_tags` значение `On`. 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">`<br>`...`<br>`</script>` | Расширенный вариант PHP-тегов, так же как и вариант `<?php ... ?>`, доступен 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, предназначенный 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 чем одну PHP-вставку. В листинге 2.4 приводится пример, который содержит две вставки: одна задает название 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 надобность отделять его от других выражений отсутствует. 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 точки с запятой после каждого выражения, т. к. добавление новых операторов может привести к появлению трудноулавливаемых 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, подобных той, которая приведена в листинге 2.6. Понятно написанный код — залог успеха, т. к. такой код намного проще и быстрее 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 зеленого слова "Истина", либо красного слова "Ложь". 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, обсуждается 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 следующих 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 не волей 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 слэшей (`//`) и заканчивающийся переводом строки |
| `# ...` | Комментарий в стиле скриптовых языков UNIX, начинающийся с символа диеза (`#`) и заканчивающийся переводом строки |
| `/*`<br>`...`<br>`*/` | 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 симво-
лами, являющимися комментариями HTML. Однако он может быть просмотрен в исход-
ном 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 после 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 (его 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 присваивания =, 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 чисел**

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

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

**Листинг 3.3. Инициализация трех переменных одним значением**

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

# 3.2. Типы данных

Язык PHP является слаботипизированным, 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 данных, которые поддерживаются PHP.

***Замечание***

Тип `int64` планируется ввести в PHP 6.0.

***Таблица 3.1.*** *Типы данных PHP*

| Тип данных | 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 |
| `double` (или `float`) | Вещественное число, минимально возможное значение которого составляет $\pm 1.7 \times 10^{-308}$, а максимально возможное $\pm 1.7 \times 10^{308}$ |
| `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 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 *главе 18* |
| `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 чисел с плавающей точкой используется математический сопроцессор с достаточно сложными алгоритмами их обработки, то оперировать целыми числами процессор может напрямую.

**ЗАМЕЧАНИЕ**

В отличие от других языков программирования переполнения (т. е. выхода 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 исчисления, 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.

**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
  // Научная запись
  $var = -1.23456e-2; // -0.0123456
?>
```

*Научная запись* помимо мантиссы (целой и дробной части) содержит порядок, который начинается со строчной или прописной буквы E и целого положительного (или отрицательного) числа. Так, запись 1.2e–3 эквивалентна произведению $1.2\times10^{-3}$ (или 0.0012). Вещественное число 1.1e+2 (или 1.1e2) эквивалентно $1.1\times10^{2}$ (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 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 применять 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 подставляется их значения |
| `' ... '` | 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 поддерживаются два варианта кавычек: одиночные и двойные, для того чтобы при необходимости применять одиночные ка-

вычки для обрамления двойных (листинг 3.6), а двойных для обрамления одиночных (листинг 3.7).

**Листинг 3.6. Двойные кавычки**

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

**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, 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 слэшем, 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 слэша 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 значения*

| Значение | Описание |
|---|---|
| `\n` | Перевод строки |
| `\r` | Возврат каретки |
| `\t` | Символ табуляции |
| `\\` | Обратный слэш |
| `\"` | 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 ($), будет искать пробел и посчитает переменными последовательности `$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 также, 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 заглавных 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 достаточно просто, а обнаружить ее бывает нелегко.

## 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 возникно-

вение 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` (листинг 3.15) 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 конфигурационного файла php.ini — `error_reporting`.

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, рекомендуемого разработчикам для использования. В файле php.ini-dist значение директивы `error_reporting` выставлено в `E_ALL & ~E_NOTICE`, которое означает, 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.

**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 не логично, 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 запятыми 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 запятыми 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 или нет, 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, 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 некоторые функции (особенно функции для работы с массивами и объектами) крайне чувст-

вительны 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 `$var` какая-либо из существующих функций. Более подробно функции обсуждаются в *главе 7* |
| `is_double($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` |

***Таблица 3.4*** *(окончание)*

| Функция | Описание |
|---|---|
| `is_integer($var)` | Синоним для функции `is_int()` |
| `is_long($var)` | Синоним для функции `is_long()` |
| `is_null($var)` | Возвращает `TRUE`, если `$var` равен `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, независимо 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 слаботипизированном языке программирования, как PHP. Как будет показано в следующем разделе, числа в строках зачас-

тую могут быть автоматически приведены к одному из числовых типов, поэтому при проверках зачастую 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 19* и *20*.

**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 сильнотипизированные и слаботипизированные. В сильнотипизированных языках программирования при объявлении переменной, как правило, следует указывать ее тип, а использование переменной неправильного типа приводит к ошибке. В слаботипизированных язы-

ках 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 `""`. Преобразование логического типа к числу приводит к превращению `TRUE` в 1, а `FALSE` в 0. Поэтому `TRUE` всегда выводится как единица, а `FALSE`, как пустая строка (листинг 3.32).

**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 типов, поддерживаемых PHP. Для такого преобразования предусмотрено несколько меха-

низмов. 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 округление, 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 |

***Таблица 3.5*** *(окончание)*

| Преобразование | Описание |
|---|---|
| `$var = (array) $var;` | Приведение к массиву |
| `$var = (object) $var;` | Приведение к объекту |

Обычно явное преобразование типов не требуется, однако оно 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 снованию `$base` |
| `strval($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 завершение главы рассмотрим еще один способ создания переменных. Он заключается 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 `$mode` в скрипте следующим образом: `$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 символами верхнего регистра, хотя допускается создание констант с именами, содержащими символы 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 следует 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 (см. 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 выше. Если же первый из вызовов функции `define()` 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 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 одним 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 55555[, $case_insensitive] )` | Создает константу с именем `$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($name)` | Проверяет, существует константа с именем `$name`, и возвращает `TRUE`, если константа существует, и `FALSE` в противном случае |
| `constant($name)` | Возвращает значение константы с именем `$name` |

***Таблица 4.1*** *(окончание)*

| Функция | Описание |
|---|---|
| `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. Динамически константы. 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()`, эквиваленты.

***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;
  // Получение значения константы
  // через функцию constant()
  echo constant("VALUE");
?>
```

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. Динамические константы**

```
<?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;
?>
```

# 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 *главе 18*.

***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, определение той или иной константы зависит от того, подключено расширение или нет. 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 точки**

```
<?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 объединением 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 $number;
?>
```

## 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 запятыми. Эти параметры 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!1
  // 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 С-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.

Операторы инкремента (++) и декремента (--) подразделяют на префиксные и постфиксные. При префиксной операции допустим инкремент: увеличение значения операнда на 1 происходит до того, как возвращается значение. Соответственно, при постфиксной — после. Таким образом, мы можем записывать операторы инкремента и декремента в двух формах — префиксной и постфиксной (листинг 5.12). Для удобства далее в примерах будем говорить 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 (листинг 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. Учитывайте это различие, поскольку из-за невнимательности при работе с операциями инкремента и декремента проистекает достаточно много ошибок, найти которые в большом коде не всегда тривиально.

Операндом операций инкремента и декремента должны быть только переменные. То есть запись `++1` является неверной, запись `++($a + $b)` также неверна.

Операторы `++` и `--` можно применять 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 операторов, предназначенных 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 & (простой пример)**

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

Этот, казалось бы, парадоксальный результат можно легко понять, если перевести числа 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 `&` (сложный пример)**

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

Для наглядности представим складываемые числа в двоичном представлении:

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

Оператор `|` соответствует побитовому ИЛИ (объединение). 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 значению.

**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:

- для `6 << 1`:

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

- для `6 << 2`:

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

- для `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 позицию, 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 значению (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:

- для `6144 >> 10`:

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

- для `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 приводятся 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` равно `$y` |
| `$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 числам 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 5.22 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
?>
```

Заметим, что 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 |

***Таблица 5.10*** *(окончание)*

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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(`*`condition`*`)` *`operator1`* `else` *`operator2`*

В качестве аргумента *`condition`* оператор `if` принимает логическую переменную или выражение, возвращающее логическую переменную. Если условие истинно, то выполняется оператор *`operatot1`*. В противном случае выполняется оператор *`operator2`* (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 оператора `else`. Если блок `else` не требуется, его можно пустить (листинг 5.25). Кроме этого, *`operator1`* и *`operator2`* 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 `else` в каждом `if` может быть только один. 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 бинарные 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 |
| `! $x` | Возвращает `TRUE`, если `$x` равен `FALSE`, или `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 приложению — пара из них рано или поздно останется не поправленной, приводя к труднолокализуемым ошибкам.

**Листинг 5.31. Неудачное дублирование кода**

```
<?php
  $flag1 = TRUE; // Истина
  $flag2 = TRUE; // Истина
  if($flag1)
  {
    if($flag2)
    {
      echo "<p>Оба флага истинны</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`, которые имеют более низкий приоритет. То есть если в арифметических операциях сначала выпол-

няется 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 — независимо 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 выполняет-

ся — 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 отменить, 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 результате чего 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.

Иногда бывает необходимо безальтернативно проверить условие 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 не-

удачно. 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 *operator1* и *operator2* 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 `$x == 1`, является сокращенным и в качестве *operator1* включает полный условный оператор, где проверяется условие `$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 также, как C++ и Java, предоставляет возможность заменять блоки `if ... else` условной операцией, которая имеет следующий синтаксис:

*выражение_1* ? *выражение_2* : *выражение_3*

Первым вычисляется значение *выражение_1*. Если оно истинно, то вычисляется значение *выражение_2*, которое и становится результатом. Если *выражение_1* ложно, то в качестве результата берется *выражение_3*. 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 — ее знак меняется, 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 представлен ниже:

```
switch(expression)  // переключающее выражение
{
  case value1:      // выражение 1
    statements;     // блок операторов
    break;
  case value2:      // выражение 2
    statements;
    break;
  default:
    statements;
}
```

Управляющая структура `switch` передает управление тому из помеченных операторов `case`, для которого значение константного выражения совпадает со значением переключающего выражения *`expression`*. Сначала анализируется переключающее выражение *`expression`* 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 одной 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 листинге 5.45 будет пропущен оператор `break`, то на экран будут выведены сразу два сообщения. В листинге 5.46 выводятся названия нечетных десятичных цифр от 1 до 9 не меньше введенной.

**Листинг 5.46. Вывод цифр**

```
<?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, большее 9 или меньшее 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 в листинге 5.49, благодаря тому, что логическое выражение можно сравнивать со

значением TRUE или FALSE. В листинге 5.50 приводится скрипт, аналогичный по функциональности скрипту из листинга 5.49.

**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 заменить, 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 эквиваленты, 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 С-подобных языках программирования совместно с оператором switch используется break. Отклонения от этого правила потребуют от большинства программистов дополнительных усилий на разбор кода, а следовательно, приведут 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. Синтаксис оператора выглядит следующим образом:

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

При входе в цикл вычисляется выражение-условие *`condition`*, 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 *`condition`* 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 (листинга 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 из листинга 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 листинг 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 `$number` принимает значение 0, цикл прекращает свою работу. В листинге 5.59 используется префиксная форма декремента, в которой из числа сначала вычитается единица, и лишь затем возвращается значение, поэтому цифра 0 не выводится. В листинге 5.60 по достижении `$number` значения 1 в операторе `while` сначала возвращается значение 1, и лишь затем из `$number` вычитается еди-

ница, 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(condition);
```

Такой цикл всегда будет выполнен хотя бы один раз. После выполнения тела цикла вычисляется выражение-условие *`condition`*, 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***
>
> В цикле `do ... while` также допускается использование операторов `break` и `continue`, порядок применения 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(begin; condition; body)
{
  операторы;
}
```

Здесь оператор *`begin`* (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 (*`condition`*) такой же, как и у циклов с пред- и постусловиями — цикл выполняется до тех пор, пока выражение *`condition`* истинно (`TRUE`), и прекращает свою работу, когда оно ложно (`FALSE`). При отсутствии выражения-условия предполагается, что его значение всегда истинно. Выражения *`body`* вычисляются в конце каждой итерации после выполнения тела цикла.

> ***Замечание***
>
> В цикле `for` также допускается использование операторов `break` и `continue`, порядок применения 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 4.30 выглядит следующим образом:

```
5
4
3
2
1
```

Значение переменной `$i` инициируется значением 5, на каждой итерации цикла это значение уменьшается на единицу при помощи оператора декремента (`$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, поскольку 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 С-подобного языка программирования), но не приветствуется, т. к. здорово уменьшает читабельность кода.

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

> ***Замечание***
>
> Пример 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 *begin*, *condition* и *body* не обязательно должны присутствовать. Так, в листинге 5.68 приводится пример цикла `for`, в котором отсутствуют выражения *begin* и *body*.

**Листинг 5.68. Отсутствие выражений *begin* и *body***

```
<?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, которое 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 осуществляется 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 приводятся 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 вызовов конструкции не встречалось. В случае отсутствия файла index.php выводится 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 |
| `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 исход-

ном 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.

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

Выполнение включаемого скрипта second.php можно прекратить досрочно, объявив во включаемом файле оператор `return` (листинг 5.74).

**Листинг 5.74. Файл second.php (вариант 2)**

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

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.

**Рис. 5.2.** Досрочное прекращение выполняемого файла

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. 5.12) 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 бы могли затруднить чтение кода.

Как было упомянуто в начале раздела, помимо функции `include()` существует функция `require()`, выполняющая аналогичные действия. Различия в этих функциях заключаются 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, независимо 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 непо-

средственно 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 может подставить в качестве пути к файлу собственный 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.

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;
?>
```

**Рис. 5.3.** Вывод замечания Notice из-за неинициализированной переменной `$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 |
| \| | 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 `$arr`, `$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 С-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 значения 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 случае 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:

```
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 один 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 `$values` не совпадает) |
| `compact($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:

```
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 *$prefix* добавляется только к переменным с недопустимыми именами (числовыми) |
| EXTR_IF_EXISTS | Переопределяются только уже существующие переменные, в противном случае никакие действия не предпринимаются |
| EXTR_PREFIX_IF_EXISTS | Создать переменные, к именам которых будет добавлен префикс *$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 также 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, в которые заключается 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 отменить, 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 не инициализированными.

**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 не инициализированной. 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 числовых 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($array)` | Возвращает текущее значение массива `$array`, на которое указывает курсор |
| `pos()` | Синоним для функции `current()` |
| `each($array)` | Возвращает текущее значение массива `$array`, на которое указывает курсор, и переводит его на следующую позицию |
| `end($array)` | Переносит курсор в конец массива `$array` |
| `key($array)` | Возвращает ключ текущего элемента массива `$array` |
| `next($array)` | Переводит курсор на следующую позицию массива `$array` |

***Таблица 6.4*** *(окончание)*

| Функция | Описание |
|---|---|
| `prev($array)` | Переводит курсор на предыдущую позицию массива `$array` |
| `reset($array)` | Устанавливает курсор на начало массива `$array` |
| `list($varname, ...)` | Преобразует массив в переменные. Конструкция 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 функции `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 ниже:

```
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 оператора `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. Проверка существования элементов массива

Проверить, существует тот или иной элемент массива, можно при помощи конструкции `isset()` (листинг 6.37).

**Листинг 6.37. 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 6.37 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 6.38).

**Листинг 6.38. 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 6.38 выводит следующие строки:

```
Это массив
Это не массив
```

Конструкция `isset()` выполняет проверку существования элемента с заданным ключом. Зачастую требуется выяснить, входит ли в состав того или иного массива элемент с заданным ключом или значением. Для этого удобнее использовать специализированные функции (табл. 6.5).

***Таблица 6.5.*** *Проверка существования элементов*

| Функция | Описание |
|---|---|
| `in_array($needle, $array [, $strict])` | Проверяет, существует ли значение `$needle` в массиве `$array`, и возвращает `TRUE`, если значение найдено, и `FALSE` в противном случае. Если необязательный параметр `$strict` принимает значение `TRUE`, то при сравнении значения элемента массива с `$search_value` будет использоваться оператор эквивалентности `===`, в противном случае будет использован оператор равенства `==` |

***Таблица 6.5*** *(окончание)*

| Функция | Описание |
|---|---|
| array_key_exists ($*key*, $*search*) | Возвращает TRUE, если в массиве $*search* присутствует ключ $*key*, в противном случае возвращается FALSE |
| array_search($*needle*, $*haystack* [,$*strict*]) | Ищет значение $*needle* в массиве $*haystack* 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($*needle*, $*array* [, $*strict*])

Функция ищет в массиве $*array* значение $*needle* и возвращает TRUE, если значение найдено, и FALSE в противном случае (листинг 6.39).

**Листинг 6.39. 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 6.40).

**Листинг 6.40. 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 найдено 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, $array)
```

Функция возвращает `TRUE`, если ключ `$key` найден в массиве `$array` (листинг 6.41).

**Листинг 6.41. 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 ($needle, $array [, $strict])
```

Функция ищет значение `$needle` в массиве `$array` 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 6.42.

**Листинг 6.42. 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;
?>
```

# 6.9. Количество элементов в массиве

Функции данной группы осуществляют подсчет количества элементов в массиве (табл. 6.6).

***Таблица 6.6.*** *Количество элементов в массиве*

| Функция | Описание |
|---|---|
| `count($array [, $mode])` | Возвращает количество элементов массива `$array`. Если необязательный параметр `$mode` принимает значение константы `COUNT_RECURSIVE`, функция рекурсивно обходит многомерный массив, в противном случае подсчитывается количество элементов только на текущем уровне |
| `sizeof()` | Синоним для функции `count()` |
| `array_count_values($input)` | Подсчитывает количество уникальных значений среди элементов массива. В результате возвращается ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значений — количество их вхождений в массив `$input` |

Для подсчета количества элементов в массиве предназначена функция `count()`, которая имеет следующий синтаксис:

```
count($array [, $mode])
```

Возвращает количество элементов массива `$array`. В листинге 6.43 демонстрируется пример с использованием этой функции.

**Листинг 6.43. 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 размер текущего измерения (листинг 6.44).

**Листинг 6.44. 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 во вложенных массивах, необязательному параметру `$mode` необходимо передать в качестве значения константу `COUNT_RECURSIVE` (листинг 6.45).

**Листинг 6.45. 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($array)`

Функция возвращает ассоциативный массив, в качестве ключей которого выступают значения массива, а в качестве значений — количество их вхождений в массив `$array` (листинг 6.46).

***Замечание***

Массив *$array* может иметь в качестве элементов лишь числа или строки, в противном случае генерируется предупреждение.

**Листинг 6.46. 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 6.46 будет выведен следующий дамп массива $new:

```
Array
(
    [1] => 2
    [hello] => 2
    [world] => 1
)
```

# 6.10. 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 использования функции array_sum() демонстрируется в листинге 6.47.

**Листинг 6.47. Использование функции array_sum()**

```
<?php
  $arr = array(1,2,3,4,5);
  $sum = array_sum($arr);
  echo $sum; // выводит 15
?>
```

# 6.11. 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 6.48).

> ***Замечание***
>
> Функция `rand()` является математической функцией и более подробно рассматривается в *главе 12*.

**Листинг 6.48. 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];
?>
```

Еще две функции описаны в табл. 6.7.

***Таблица 6.7.*** *Случайные элементы массива*

| Функция | Описание |
|---|---|
| `array_rand($input [, $num_req])` | Выбирает одно или несколько случайных значений из массива. Если параметр *$num_req* не указывается, возвращается ключ для случайного элемента массива *$input*. Если в параметре *$num_req* задается число большее 1, возвращается массив со случайным набором ключей массива *$input* |

***Таблица 6.7*** *(окончание)*

| Функция | Описание |
|---|---|
| `shuffle($array)` | Перемешивает элементы массива `$array` в случайном порядке |

Подход, описанный в листинге 6.48, 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. Функция имеет следующий синтаксис:

```
array_rand($input [, $num_req])
```

Функция выбирает случайное значение из массива `$input`. Если параметр `$num_req` не указывается, возвращается ключ для случайного элемента массива `$input`, если в параметре `$num_req` задается число большее 1, возвращается массив со случайным набором ключей массива `$input` (листинг 6.49).

**Листинг 6.49. 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($array)
```

Листинг 6.50 демонстрирует работу функции.

**Листинг 6.50. 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
)
```

# 6.12. Сортировка массивов

PHP предоставляет разработчику широкие возможности для сортировки массивов (табл. 6.8).

***Таблица 6.8.*** *Функции для сортировки массивов*

| **Функция** | **Описание** |
|---|---|
| `sort($array [, $sort_flags])` | Сортирует массив `$array` в прямом порядке. Параметр `$sort_flags` задает параметры сортировки:<br>• `SORT_REGULAR` — обычная сортировка;<br>• `SORT_NUMERIC` — сравнивать элементы как числа;<br>• `SORT_STRING` — сравнивать элементы как строки;<br>• `SORT_LOCALE_STRING` — сравнивать элементы как строки, основываясь на текущей локали |
| `rsort($array [, $sort_flags])` | Сортирует массив `$array` в обратном порядке |
| `asort($array [, $sort_flags])` | Сортирует массив `$array` в прямом порядке с сохранением отношения "ключ-значение" |

***Таблица 6.8*** *(окончание)*

| Функция | Описание |
|---|---|
| `arsort($array [, $sort_flags] )` | Сортирует массив `$array` в обратном порядке с сохранением отношения "ключ-значение" |
| `natsort($array)` | Осуществляет "естественную" сортировку массива `$array`, учитывая регистр |
| `natcasesort($array)` | Осуществляет "естественную" сортировку массива `$array`, без учета регистра |
| `ksort($array [, $sort_flags])` | Сортирует массив `$array` в прямом порядке по ключам с сохранением отношения "ключ-значение" |
| `krsort($array [, $sort_flags])` | Сортирует массив `$array` в обратном порядке по ключам с сохранением отношения "ключ-значение" |
| `usort($array, $cmp_function)` | Сортирует массив `$array` с применением пользовательской функции сравнения элементов `$cmp_function` |
| `uasort ($array, $cmp_function)` | Сортирует массив `$array` с применением пользовательской функции сравнения элементов `$cmp_function` и сохранением отношения "ключ-значение" |
| `uksort($array, $cmp_function)` | Сортирует массив `$array` по ключам с применением 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 6.51 приводится пример использования функции `sort()`.

**Листинг 6.51. 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 альфа-бета 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()` выполняет сортировку массива по убыванию и имеет следующий синтаксис:

```
rsort($arr [, $sort_flags])
```

Во всем остальном ведет себя аналогично функции `sort()`. В листинге 6.52 приводится пример использования функции `rsort()`.

**Листинг 6.52. 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 зна-

чениями (листинг 6.53), чего нет в функции `sort()` (там эти связи попросту разрываются).

**Листинг 6.53. 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])
```

Функция `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 6.54 приводится пример использования функции `ksort()`.

**Листинг 6.54. 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 6.55).

**Листинг 6.55. 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. 6.2).

**Рис. 6.2.** Аргументы функции `array_multisort()` образуют таблицу

В листинге 6.56 приводится пример сортировки двух массивов: `$arr1` и `$arr2`.

**Листинг 6.56. 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. 6.9).

### *Замечание*

Группы констант `SORT_ASC` и `SORT_DESC`, а также `SORT_REGULAR`, `SORT_NUMERIC` и `SORT_STRING` являются взаимоисключающими.

***Таблица 6.9.*** *Константы, определяющие поведение функции* `array_multisort()`

| Константа | Описание |
|---|---|
| `SORT_ASC` | Сортировать в возрастающем порядке |
| `SORT_DESC` | Сортировать в убывающем порядке |

***Таблица 6.9*** *(окончание)*

| Константа | Описание |
|---|---|
| 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 6.57 приводится модифицированный вариант скрипта, сортирующий таблицу в обратном порядке.

**Листинг 6.57. 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 6.57 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
)
```

# 6.13. 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.

## 6.13.1. Элемент *$_SERVER['DOCUMENT_ROOT']*

Элемент `$_SERVER['DOCUMENT_ROOT']` содержит путь к корневому каталогу сервера. Если скрипт выполняется в виртуальном хосте, в данном элементе, как правило, указывается путь к корневому каталогу виртуального хоста. То есть если в конфигурационном файле httpd.conf виртуальный хост имеет директиву `DocumentRoot`, которой присвоено значение "`D:/main`" (листинг 6.58), то элемент `$_SERVER['DOCUMENT_ROOT']` будет содержать значение "`D:\main`".

**Листинг 6.58. 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>
```

## 6.13.2. 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 6.59) и page.php (листинг 6.60).

**Листинг 6.59. 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 иметь аналогичное содержание, но ссылка будет указывать на страницу index.php (см. листинг 6.59).

**Листинг 6.60. 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. 6.3).

**Рис. 6.3.** Фиксация адреса предыдущей страницы

## 6.13.3. 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 Navigator 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 систему Linux с ядром, оптимизированным под Pentium 4, находясь в графической оболочке X-Window. Этот механизм удобно использовать для сбора статистической информации, которая позволяет дизайнерам оптимизировать страницы под наиболее распространенные браузеры.

## 6.13.4. Элемент *$_SERVER['REMOTE_ADDR']*

В элемент `$_SERVER['REMOTE_ADDR']` помещается IP-адрес клиента. При тестировании на локальной машине этот адрес будет равен 127.0.0.1. Однако при тестировании в сети переменная вернет IP-адрес клиента или последнего прокси-сервера, через который клиент попал на сервер. Если клиент использует прокси-сервер, то узнать его IP-адрес можно при помощи переменной окружения `$_SERBER['HTTP_X_FORWARDED_FOR']`.

***Замечание***

Прокси-серверы являются специальными промежуточными серверами, предоставляющими особый вид услуг: сжатие трафика, кодирование данных, адаптация 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`.

## 6.13.5. 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"`.

## 6.13.6. 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 6.61 приводится пример использования данных элементов.

**Листинг 6.61. 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 6.61 представлен на рис. 6.4.

**Рис. 6.4.** Результат работы скрипта из листинга 6.61

## 6.13.7. 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 6.62, помещая в строке запроса произвольный текст после знака `?`, получим страницу с введенным текстом.

**Рис. 6.5.** Результат работы скрипта из листинга 6.62

**Листинг 6.62. Использование элемента $_SERVER['QUERY_STRING']**

```
<?php
  echo $_SERVER['QUERY_STRING'];
?>
```

Результат работы скрипта из листинга 6.62 представлен на рис. 6.5.

## 6.13.8. 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"`.

## 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 свои 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
}
```

Рассмотрим простой пример (листинг 7.1).

**Листинг 7.1. Пример простой функции**

```
<?php
  function get_sum()
  {
    $sum = 10 + 5;
    return $sum;
  }
  echo get_sum(10, 5); // 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 конструкции `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 конструкцию `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 языках программирования функция не может вызываться до ее объявления. В PHP отсутствуют подобные ограничения (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 7.10).

**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. То есть 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 7.7 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. Если аргумент передается по ссылке, то 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);
?>
```

**Рис. 7.2.** Вместо двух параметров функции передан лишь один

Впрочем, попытка выполнения скрипта из листинга 7.11 не заканчивается прекращением выполнения скрипта — 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).

## 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 если даже 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)
?>
```

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. Организация 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 в 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` в этом случае 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 функции 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 сложным по восприятию конструкциям языка, и отладка их достаточно сложна, особенно когда приходится иметь дело не с простейшей рекурсивной функцией, представленной в листинге 7.22, 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, возникает 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 удалены файлы на самом глубоком уровне вложенности. К та-

ким задачам относится так же 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 оператора 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 |

**Таблица 7.2** *(окончание)*

| Функция | Описание |
|---|---|
| `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 20*.

**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. 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, а часть может подключаться извне.

> ***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 вызовом при помощи специализированных функций вроде `call_user_func_array()` и `call_user_func()`. PHP предоставляет разработчикам возможность неявного, условного вызова функций при помощи конструкции `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 |

**Таблица 7.4** *(окончание)*

| Функция | Описание |
|---|---|
| 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 через 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**

```
<?php
  // Объявляем функцию
  function line()
  {
    echo "Line<br>";
  }

  // Регистрируем функцию для выполнения в условии declare
  register_tick_function ("line");

  // Вызываем функцию line() через 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`**

```
<?php
  // Объявляем функцию
  function line()
  {
    echo "Line<br>";
  }

  // Регистрируем функцию для выполнения в условии declare
  register_tick_function ("line");

  // Вызываем функцию line() через 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)* |

***Таблица 7.5*** *(окончание)*

| Функция | Описание |
|---|---|
| `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. 6.13* |

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, начинающиеся с цифры или содержащие 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 выше 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($str)` | Обратная `urlencode()` функция, осуществляющая декодирование символов, начинающихся с `%` |
| `rawurlencode($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 |

***Таблица 8.1*** *(окончание)*

| Функция | Описание |
|---|---|
| `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 `?`;
- `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).

Для того чтобы предотвратить искажение, представленное на рис. 8.2, элементы управления помещают в ячейки HTML-таблицы (листинг 8.7).

**Рис. 8.2.** Искажение дизайна HTML-формы

**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>
```

**Рис. 8.3.** Выравнивание элементов управления HTML-формы

Результат интерпретации HTML-кода из листинга 8.7 представлен на рис. 8.3.

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`.<br>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) |
| `target` | Указывает окно для вывода результата, полученного от обработчика HTML-формы. Атрибут может принимать следующие значения:<br>• `_blank` — результат открывается в новом окне;<br>• `_self` — результат открывается в текущем окне; данный режим используется по умолчанию, если атрибут `target` не указан явно;<br>• `_parent` — результат открывается в родительском фрейме; при отсутствии фреймов режим аналогичен `_self`;<br>• `_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 они направляются 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 загружалась 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 следующего вида:

```
<input type='text'>
```

Следует отметить, что 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 при помощи 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 как ключ 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, то 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 указание 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 `enctype` формы определяет вид кодировки, которую браузер применяет к параметрам формы. Для того чтобы отправка файлов на сервер действовала, атрибуту `enctype` 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".

Следующий скрипт позволяет вывести характеристики загруженного файла. Для этого необходимо модифицировать скрипт так, как представлено в листинге 8.21.

**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 показано 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 разделах данной главы будут рассмотрены функции, позволяющие обрабатывать строки.

## 9.1. Функции для работы с символами

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 также 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
?>
```

# 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. 9.2 приводится список строковых функций, осуществляющих поиск в строке.

***Таблица 9.2.*** *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 |

***Таблица 9.2*** *(окончание)*

| Функция | Описание |
|---|---|
| `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 |
| `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.

Рассмотрим пример (листинг 9.8).

**Листинг 9.8. 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 9.8 приводится пример использования функции `substr()` — из строки "04.05.2009" извлекаются дата, месяц и год.

Функция `substr_count()` позволяет ответить на вопрос, сколько раз подстрока встречается в строке. В листинге 9.9 при помощи этой функции анализируется, сколько раз подстрока "hello" входит в состав строки.

**Листинг 9.9. 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 9.10 функция обнаружит вхождение только одного символа `"l"`, т. к. поиск начинается лишь с пробела и продолжается только до запятой.

**Листинг 9.10. 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 9.11 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 9.11. 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 9.12).

**Листинг 9.12. 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 9.13).

**Листинг 9.13. 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 9.13 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.

То есть 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 9.14 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 9.14. 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 9.15. 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 9.15 первый вызов функции `strstr()` возвращает часть адреса, начиная с подстроки `"www"` и до конца строки. Второй вызов возвращает часть строки, предшествующую подстроке `"www"`.

Следует отметить, что функция `strstr()` чувствительна к регистру, т. е. скрипт в листинге 9.16 не выведет ничего, т. к. подстрока `"WWW"` не будет найдена в строке `"http://www.softtime.ru"`.

**Листинг 9.16. 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 9.17).

**Листинг 9.17. Функция `stristr()` не 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 9.18 при помощи функции `strrchr()` из пути извлекается имя файла.

**Листинг 9.18. 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 9.19 приводится пример использования функции `strpbrk()`.

**Листинг 9.19. Использование функции `strpbrk()`**

```
<?php
  echo strpbrk("Hello world", 'wl'); // llo world
?>
```

# 9.3. Замена в тексте

Функции замены осуществляют преобразование строк, связанное с заменой одних подстрок другими, а также удалением подстрок. Полный список функций этой группы представлен в табл. 9.3.

***Таблица 9.3.*** *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` |
| `strtr($str, $replace_pairs)` | Возвращает копию строки `$str`, в которой замены осуществлены в соответствии с массивом `$replace_pairs` (ключи заменяются значениями массива) |

***Таблица 9.3*** *(окончание)*

| Функция | Описание |
|---|---|
| `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 9.20).

**Листинг 9.20. 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 из листинга 9.20 будет замена всех символов `[b]` на `<b>`, а `[/b]` на `</b>`.

Специальным видом замены является удаление подстрок из строки. Например, следующий скрипт удаляет из текста все теги `<b>` и `</b>` (листинг 9.21).

**Листинг 9.21. 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 9.22 аналогичен по результату скрипту из листинга 9.21.

**Листинг 9.22. 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 9.23).

**Листинг 9.23. 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 9.23 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 9.24 демонстрируется использование функции `substr_replace()`.

**Листинг 9.24. 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 9.25 приводится пример преобразования символов верхнего регистра в нижний регистр.

**Листинг 9.25. 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 9.26).

**Листинг 9.26. 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 9.25 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 аргумента *$str* 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 9.27).

**Листинг 9.27. 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 9.28.

### *Замечание*

Список пробельных символов в параметре *$charlist* задается в шестнадцатеричном формате. Последовательность .. позволяет задать интервал. Так, в листинге 9.28 задается интервал ASCII-кодов от 0 (\x00) до 32 (\x1F), после чего отдельно задается символ с кодом 160 (\xA0).

**Листинг 9.28. 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. 9.4.

***Таблица 9.4.*** *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 9.29 и 9.30 приводятся примеры использования функций `strtolower()` и `strtoupper()`.

**Листинг 9.29. Использование функции `strtolower()`**

```
<?php
  echo strtolower("HELLO WORLD<br>"); // hello world
?>
```

**Листинг 9.30. Использование функции `strtoupper()`**

```
<?php
  echo strtoupper("Hello World"); // HELLO WORLD
?>
```

Функция `ucfirst()` производит преобразование первого символа строки `$str` в верхний регистр (листинг 9.31). Если первый символ строки `$str` уже находится в верхнем регистре, функция возвращает строку без изменений.

**Листинг 9.31. 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 9.32).

**Листинг 9.32. 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-разработки, среди строковых функций имеется несколько специально предназначенных для Web-разработки. В табл. 9.5 приводится список строковых функций, предназначенных для работы с HTML-кодом.

***Таблица 9.5.*** *Функции для работы с HTML-кодом*

| Функция | Описание |
|---|---|
| `nl2br($str)` | Добавляет перед переводами строки HTML-тег перевода строки `<br>` |
| `htmlspecialchars($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. |

***Таблица 9.5*** *(окончание)*

| **Функция** | **Описание** |
|---|---|
| | 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 [, $quote_style])` | Выполняет обратную функции `htmlspecialchars()` задачу — преобразует представление HTML-тегов в исходные HTML-теги |
| `htmlentities($string [, $quote_style [, $charset [, $double_encode]]])` | Функция аналогична `htmlspecialchars()`, однако преобразует не минимальное количество символов для представления HTML-кода, а все возможные символы, для которых предусмотрено представление |
| `html_entity_decode($string [, $quote_style [,$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 9.33).

**Листинг 9.33. Использование функции `nl2br()`**

```
<?php
  $str = '
```

```
  $str = "Строка";
  for($i = 0; $i < strlen($str); $i++)
  {
     echo $str[$i];
  }';
  echo $str; // Выводим как есть
  echo "<br><br><br>";
  echo nl2br($str);  // Обрабатываем переменную $str перед выводом
?>
```

Результат работы скрипта из листинга 9.33 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 неудобным, если текст далее предназначен для работы с JavaScript-скриптом. В этом случае вместо функции `nl2br()` удобнее воспользоваться функцией `str_replace()` (листинг 9.34).

### 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 ряд программ (например, Блокнот) не воспринимают его как полноценный перевод строки и выводят в виде квадратика.

**Листинг 9.34. Замена перевода строк HTML-эквивалентом <br>**

```
<?php
  $str = "Текст, содержащий\nперевод строки";
  echo str_replace("\n", "<br>", $str);
?>
```

Скрипт, представленный в листинге 9.34, заменяет лишь символ перевода строки `\n`, оставляя не затронутым символ возврата каретки `\r`. Для того чтобы преобразовать как UNIX, так и Windows-переводы строк, необходимо воспользоваться скриптом, представленным в листинге 9.35.

**Листинг 9.35. 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. 9.6.

### Замечание

Те кодировки, которые отсутствуют в табл. 9.6, не поддерживаются. Вместо них используется ISO-8859-1.

***Таблица 9.6.*** *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 |

***Таблица 9.6*** *(окончание)*

| Кодировка | Псевдоним | Описание |
|---|---|---|
| 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 9.36 приведен код HTML-формы для ввода сообщений (index.html).

**Листинг 9.36. 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 9.36, имеет текстовую область `msg`, кнопку **Добавить**, после нажатия которой данные отправляются в обработчик handler.php, код которого приведен в листинге 9.37.

**Листинг 9.37. 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

**Рис. 9.3.** Результат обработки введенного текста при помощи функции `htmlspecialchars()`

Так как при обработке сообщения использовалась функция `htmlspecialchars()`, ничего страшного не произойдет, обработчик просто выведет текст скрипта, который был набран (рис. 9.3).

Если же введенный пользователем текст не обрабатывать функцией `htmlspecialchars()`, то вместо текста скрипта будет выведен результат его выполнения (рис. 9.4).

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;
```

**Рис. 9.4.** Вот что получается без использования функции `htmlspecialchars()`

Для того чтобы избежать такого поведения, четвертому параметру функции `htmlspecialchars()` следует передать `FALSE` (листинг 9.38).

**Листинг 9.38. 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 9.39 приводится пример использования функции `htmlspecialchars_decode()`.

**Листинг 9.39. 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 9.40 одна и та же фраза подвергается преобразованию функций `htmlspecialchars()` и `htmlentities()`.

**Листинг 9.40. 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, то 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.

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 пробельными символами, что может быть не всегда удобно.

В листинге 9.41 приводится пример использования функции `strip_tags()`. Для того чтобы был виден результат, перед выводом в окно браузера он пропускается через функцию `htmlspecialchars()`.

**Листинг 9.41. 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 9.41 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. 9.7.

***Таблица 9.7.*** *Функции экранирования*

| **Функция** | **Описание** |
|---|---|
| `addslashes($str)` | Возвращает копию строки `$str`, в которой перед каждым спецсимволом добавлен обратный слэш (`\`). Экранируются одиночная кавычка (`'`), двойная кавычка (`"`), обратный слэш (`\`) и NUL (символ `\0`) |
| `stripslashes($str)` | Функция `stripslashes()` является обратной для `addslashes()`, предназначена для удаления обратных слэшей |
| `addcslashes($str, $charlist)` | Возвращает копию сроки `$str`, в которой перед каждым символом из перечисленных в `$charlist` добавлен обратный слэш (`\`) |
| `stripcslashes($str)` | Функция `stripcslashes()` удаляет символ обратного слэша и является противоположной функции `addcslashes()` |

***Таблица 9.7*** *(окончание)*

| Функция | Описание |
|---|---|
| `quoted_printable_decode($str)` | Декодирует строку `$str`, закодированную методом quoted printable |
| `quotemeta($str)` | Возвращает копию строки `$str`, в которой перед каждым символом `. \\ + * ? [ ^ ] ( $ )` помещается обратный слэш (`\`). Функция удобна для экранирования данных в регулярных выражениях |

Функция `addslashes()` удобна для экранирования данных, например, при вставке в базу данных (листинг 9.42).

**Листинг 9.42. 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, задав 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 cookie), автоматически экранируются. Если режим отключен, автоматическое экранирование 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 9.43 приводится типичный способ обработки данных, полученных от HTML-формы методом POST.

**Листинг 9.43. Использование функции `get_magic_quotes_gpc()`**

```
<?php
  if(!get_magic_quotes_gpc())
  {
    $_POST['name'] = addslashes($_POST['name']);
  }
?>
```

Как видно из листинга 9.43, если режим магических кавычек отключен, элемент `name` суперглобального массива `$_POST` подвергается обработке функции `addslashes()`, если режим отключен — обработка не требуется и даже вредна, т. к. приведет к появлению лишних слэшей в элементе `$_POST['name']`. Из-за того что ряд серверов может включать режим обратных кавычек, ряд отключать, было принято решение, начиная с PHP 6.0.0, вообще отказаться от такой возможности. Функция `get_magic_quotes_gpc()` должна была 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 9.44).

**Листинг 9.44. 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 9.44, 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 строки `$str`, в которой перед каждым символом из перечисленных в параметре `$charlist` добавлен обратный слэш (`\`).

> ***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` (`NULL`), `\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 9.45.

**Листинг 9.45. Использование функции `addcslashes()`**

```
<?php
  $escaped = addcslashes($not_escaped, "\0..\37!@\177..\377");
?>
```

# 9.7. Форматный вывод

Функции данной группы осуществляют вывод информации в окно браузера и ее форматирование. Список функций приводится в табл. 9.8.

> ***Замечание***
>
> `echo` не является функцией и относится к языковым конструкциям наряду с операторами `if`, `for` и т. п.

***Таблица 9.8.*** *Функции форматного вывода*

| Функция | Описание |
|---|---|
| `echo($arg1 [, ...])` | Выводит аргументы `$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` |

**Таблица 9.8** *(окончание)*

| Функция | Описание |
|---|---|
| `vsprintf($format, $args)` | Возвращает строку, состоящую из элементов массива `$args`, отформатированных в соответствии со строкой `$format` |
| `number_format($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 9.46 демонстрируется ее использование.

**Листинг 9.46. 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 9.47.

**Листинг 9.47. 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()` помещает результат 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 9.47, определитель преобразования `%d` подставляет в строку число, которое передается в качестве второго аргумента функции.

**Листинг 9.48. 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. 9.9 представлены определители типа, которые допускаются в строке формата функции `printf()`.

***Таблица 9.9.*** *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` |

***Таблица 9.9*** *(окончание)*

| Определитель | Описание |
|---|---|
| `%f` | Спецификатор десятичного числа с плавающей точкой, например, `156.001` |
| `%o` | Спецификатор используется для подстановки в строку формата восьмеричное число 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 9.49 демонстрируется форматный вывод числа 1024 с использованием разнообразных определителей типа.

**Листинг 9.49. 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 9.50).

**Листинг 9.50. Преобразование цвета из десятичного в шестнадцатеричный формат**

```
<?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 9.51 под вывод числа 45 отводится 5 символов, т. к. само число занимает лишь 2 символа, три ведущих символа будут содержать символ заполнения.

**Листинг 9.51. 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 являются 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 9.52).

**Листинг 9.52. 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 отводится 8 символов, два из которых нужны 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 9.53).

**Листинг 9.53. 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 9.54).

**Листинг 9.54. 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 9.55).

**Листинг 9.55. 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
?>
```

# 9.8. Преобразование кодировок

Локальные настройки, рассмотренные в *разд. 9.8*, позволяют лишь установить системные настройки. Для преобразования кодировки строк предназначен отдельный набор функций. В табл. 9.10 приводятся стандартные функции, позволяющие осуществлять преобразование кодировок.

***Таблица 9.10.*** *Функции преобразования кодировок*

| Функция | Описание |
|---|---|
| `convert_cyr_string ($str, $from, $to)` | Преобразует строку `$str` из кодировки `$from` в кодировку `$to` |

***Таблица 9.10*** *(окончание)*

| **Функция** | **Описание** |
|---|---|
| | Параметры `$from` и `$to` могут принимать следующие значения:<br>• `'k'` — koi8-r;<br>• `'w'` — windows-1251;<br>• `'i'` — iso8859-5;<br>• `'a'` — x-cp866;<br>• `'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 `$hebrew_text` из логической кодировки в визуальную с преобразованием перевода строки в HTML-тег перевода строки `<br>`. Необязательный параметр `$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.

Пример использования функции `convert_cyr_string()` приведен в листинге 9.56, где перекодируется слово "определяющий" из кодировки windows-1251 в koi8-r и обратно.

**Листинг 9.56. 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 9.57 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 9.57. Использование функции `base64_encode()`**

```
<?php
  echo base64_encode("Hello world!");     // SGVsbG8gd29ybGQh
```

```
  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.

> ***Замечание***
>
> 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-encoding [, $from-encoding])`

Функция возвращает строку `$str`, преобразованную из кодировки `$from-encoding` в кодировку `$to-encoding`. Рассмотрим блок кода, преобразующий текст в формате UTF-16 в формат windows-1251.

> ***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 9.58. Использование функции `mb_convert_encoding()`**

```
<?php
  $data = mb_convert_encoding($data, "Windows-1251", "UTF-16");
?>
```

# 9.9. 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 осуществить приведение строк к верхнему регистру при помощи функции `strtoupper()`

или к нижнему с использованием функции `strtolower()`, как это показано в листинге 9.59.

> ***Замечание***
>
> Синтаксис функций `strtoupper()` и `strtolower()` более подробно рассматривается в *разд. 9.4*.

**Листинг 9.59. 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. 9.11.

***Таблица 9.11.*** *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 `$str2` |
| `strcasecmp($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 |

***Таблица 9.11*** *(продолжение)*

| **Функция** | **Описание** |
|---|---|
| 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 9.11*** *(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 9.60 демонстрируется использование функции `strcmp()`.

**Листинг 9.60. Сравнение строк функцией `strcmp()`**

```
<?php
  $str1 = "ttt";
  $str2 = "tttttttttt";
  echo "<br>Результат сравнения ($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 9.60 выглядит следующим образом:

```
Результат сравнения (ttt , tttttttttt): -1
```

```
Результат сравнения (tttttttttt , 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 9.61 приводится пример сортировки массива имен файлов при помощи функции `usort()`, которая осуществляет сортировку с использованием пользовательских функций. В качестве таких функций выступают `strcmp()` и `strnatcmp()`.

**Листинг 9.61. 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 9.61 представлен на рис. 9.5.

**Рис. 9.5.** Обычная и естественная сортировки

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 9.62).

**Листинг 9.62. 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 9.62 представлен на рис. 9.6.

**Рис. 9.6.** Сравнение строк при помощи функции `similar_text()`

# 9.10. Хранение данных

Строки очень часто используются для хранения разнообразных данных. Поэтому PHP имеет целый арсенал функций для упаковки данных в строки и извлечения их. В табл. 9.12 приводится список функций данной группы.

***Таблица 9.12.*** *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 определяются 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 9.63 перекодирована в шестнадцатеричный вид фраза "Hello, php!"

**Листинг 9.63. 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 9.64).

**Листинг 9.64. 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 9.64 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 возвращается как беззнаковое целое со старшими разрядами в конце.

Следующая пара функций `serialize()` и `unserialize()` позволяет 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 их сохранение 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 9.65 демонстрируется использование рассмотренных функций `serialize()` и `unserialize()`.

**Листинг 9.65. 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 9.65 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.

# 9.11. Работа с путями к файлам и каталогами

Функции данной группы позволяют облегчить работу с путями к файлам и каталогам (табл. 9.13).

***Таблица 9.13.*** *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 сохраняется 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:<br>• `PATHINFO_DIRNAME` — путь к файлу (каталог);<br>• `PATHINFO_BASENAME` — имя файла;<br>• `PATHINFO_EXTENSION` — расширение файла;<br>• `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 его элементах сохраняются каталог, в котором расположен файл, имя файла, его расширение и имя без расширения (листинг 9.66).

**Листинг 9.66. Работа с функцией `pathinfo()`**

```
<?php
  $path = pathinfo("C:/www/htdocs/index.html");

  echo "<pre>";
  print_r($path);
  echo "</pre>";
?>
```

Результатом выполнения скрипта из листинга 9.66 будет следующий дамп массива `$path`:

```
Array
(
    [dirname] => C:/www/htdocs
    [basename] => index.html
    [extension] => html
    [filename] => index
)
```

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 9.67).

> ***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 9.67. 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 9.68).

**Листинг 9.68. 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 9.69).

**Листинг 9.69. Использование функции `dirname()`**

```
<?php
  $path = "c:/www/html/index.html";
  echo dirname($path); // c:/www/html
?>
```

# 9.12. Объединение и разбиение строк

Функции данной группы осуществляют объединение подстрок в единую строку, а также разбиение строки на отдельные подстроки (табл. 9.14).

***Таблица 9.14.*** *Функции объединения и разбиения строк*

| Функция | Описание |
|---|---|
| `str_repeat($str, $number)` | Создает новую строку, состоящую из `$number` повторов строки `$str` |
| `str_pad($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 `$pad_type` определяет, с какой стороны добавляются новые символы, и может принимать следующие значения:<br>• `STR_PAD_RIGHT` — новые символы добавляются в конец строки (по умолчанию);<br>• `STR_PAD_LEFT` — новые символы добавляются в начало строки;<br>• `STR_PAD_BOTH` — новые символы добавляются как в конец, так и в начало |
| `chunk_split($str [, $chunklen [, $end]])` | Возвращает копию строки `$str`, в которой через каждые `$chunklen` (по умолчанию 76) символов вставляется подстрока `$end` (по умолчанию `\r\n`) |

***Таблица 9.14*** *(продолжение)*

| Функция | Описание |
|---|---|
| `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:<br>• 1 — возвращается массив, содержащий все слова, входящие в строку `$str`;<br>• 2 — возвращается массив, индексами которого являются позиции в строке `$str`, а значениями — соответствующие слова.<br>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 `$delimiter` |
| `join()` | Синоним для функции `implode()` |
| `wordwrap($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) |

**Таблица 9.14** *(окончание)*

| Функция | Описание |
|---|---|
| `sscanf($str, $format [, ...])` | Разбивает строку `$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 9.70).

**Листинг 9.70. Использование `str_repeat()`**

```
<?php
  echo str_repeat("Hello!",3); // Hello!Hello!Hello!
?>
```

Функция `str_pad()` увеличивает размер строки 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 9.71 демонстрируется использование функции `str_pad()`.

**Листинг 9.71. 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 9.72 приводится пример использования функции `chunk_split()` — строка разбивается на подстроки длиной 10 символов, каждая строка отделена от другой строки при помощи тега `<br>`.

**Листинг 9.72. 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 9.72 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 9.73.

**Листинг 9.73. 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 9.73 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 очередного 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 9.74).

**Листинг 9.74. 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 9.74).

**Листинг 9.75. 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 определить, как разбивать строку на отдельные слова, так 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 9.76.

**Листинг 9.76. 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 9.76 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 `$str`, находящемуся между разделителями, определяемыми аргументом `$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 9.77.

**Листинг 9.77. 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 9.77 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 9.78.

**Листинг 9.78. 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, который приводится в листинге 9.79.

**Листинг 9.79. 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 9.79.

При указании необязательных параметров функции `sscanf()` их нужно передавать по ссылке. Пример с использованием необязательных параметров приведен в листинге 9.80.

**Листинг 9.80. 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 9.81).

**Листинг 9.81. 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 величина, то в этом случае слово будет разорвано (листинг 9.82).

**Листинг 9.82. Использование параметра `cut`**

```
<?php
  $text = "Очень длинное слоооооооооооооооо.";
  $newtext = wordwrap($text, 8, "<br>", 1);
  echo "$newtext";
?>
```

Результат работы скрипта:

```
Очень
длинное
слоооооо
ооооооо
ооо.
```

## 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 функциям, 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, они 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 с самых разных языков 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, новые задачи 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 |

***Таблица 10.1*** *(окончание)*

| **Специальный символ** | **Описание** |
|---|---|
| `^` | 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 |

***Таблица 10.2*** *(окончание)*

| Квантификатор | Описание |
|---|---|
| `{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 слэшем 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 |
| `\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` |

***Таблица 10.3*** *(окончание)*

| Специальный символ | Описание |
|---|---|
| `\s` | Соответствует любому пробельному символу |
| `\S` | Соответствует любому непробельному 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 слэш (`/`), 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 неэкранируемые 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 `$offest` позволяет указать альтернативные позиции для начала поиска (по умолчанию поиск осуществляется с первой позиции строки `$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 $*offest* 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. Функции *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 наличие 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 слэшем: `"\."`. (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.

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.

**Рис. 10.1.** HTML-форма проверки адреса электронной почты

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]*$|"
```

Число 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, используя регулярное выражение `"|<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[0]`) 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 следующий 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, доступным 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 до-
билась победы со счетом 2:1. Оба спартаковских гола были забиты ударами с пе-
нальти 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 10.6 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`-ным 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, поскольку 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 изображения `<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, без затрагивания последовательности "г.", можно воспользоваться регулярным выражением `"|(?<!г)\.|is"` (листинг 10.11).

**Листинг 10.11. Замена точки на троеточие, без затрагивания "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 пе-
риоде 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 негативные ретроспективные 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 подстроки. 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 (листинг 10.14).

### *Замечание*

Синтаксис функции `wordwrap()`, которая позволяет разбивать строку на части, более подробно обсуждался в *главе 9*.

**Листинг 10.14. Разбивка длинной строки**

```
<?php
  $text = "AAAAAAAAAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAAAAAAAAAAAAAaaaaaaaaaa";

  $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. 10.14.

**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 слэш 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 со временем и датой несколько изменился, а количество функций этой группы значительно возросло.

## 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 |

***Таблица 11.1*** *(окончание)*

| Функция | Описание |
|---|---|
| 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 $*id_dst* принимает значение 1, считается, что установлено летнее время, если 0 — зимнее, если –1 или параметр не указан вовсе, PHP пытается автоматически определить его |
| gmmktime([$*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* [, $*time-stamp*]) | Разбирает строку с датой и временем и возвращает их в формате UNIXSTAMP |

Для получения текущего времени и даты в формате UNIXSTAMP традиционно используется 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 использования функции `microtime()` как с использованием, так и без использования необязательного параметра *`get_as_float`*.

**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 `$time` может принимать различные значения, начиная с "now", обозначающего текущее время, заканчивая датой в формате "10 Сентября 2008 года" и интервалами вида "+1 week" ("+1 неделя"), которые позволяют прибавлять к текущей дате указанный интервал. Необязательный параметр `$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 дней 4 час. 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.

| | |
|---|---|
| now | 10.05.2008 03:07:53 |
| 10 September 2000 | 10.09.2000 00:00:00 |
| +1 day | 11.05.2008 03:07:53 |
| +1 week | 17.05.2008 03:07:53 |
| +1 week 2 days 4 hours 2 seconds | 19.05.2008 07:07:55 |
| next Thursday | 15.05.2008 00:00:00 |
| last Monday | 05.05.2008 00:00:00 |

**Рис. 11.1.** Результат преобразования дат и временных интервалом 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 часа летом.

***Замечание***

Земной шар представляет собой шар (точнее геоид), поэтому если с одной его стороны ночь — с другой день.

Функции географической привязки перечислены в табл. 11.2.

***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 |
| `timezone_abbreviations_list()` | Возвращает массив с названиями часовых зон, смещениями 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. |

***Таблица 11.2*** *(окончание)*

| **Функция** | **Описание** |
|---|---|
| | Функция обычно самостоятельно вычисляет такое смещение по широте, однако не учитывает летнее время (именно в этом случае и используется 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 |

Функции `date_default_timezone_set()` и `date_default_timezone_get()` позволяют, соответственно, установить и возвратить текущую часовую зону. В листинге 11.9 приводится пример использования функций.

**Листинг 11.9. 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 11.10).

**Листинг 11.10. 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()`, возвращающей массив, зоны в которой разделены по географическим группам. Группа географических зон Восточной Европы находится в элементе с ключом `eest` (листинг 11.11).

**Листинг 11.11. 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 11.12 приводится пример использования функции.

**Листинг 11.12. 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 11.13 приводится пример поиска Московского часового пояса по смещению относительно Гринвича. Следует обратить внимание, что в качестве первого параметра передается пустая строка.

**Листинг 11.13. 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 используется параметр `$gmt_offset`). В листинге 11.14 приводится пример использования функции `date_sunrise()`.

### *Замечание*

Функция `date_sunset()` принимает параметры, аналогичные функции `date_sunrise()`, однако возвращает время заката солнца.

**Листинг 11.14. 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 11.15 приводится пример использования функции `date_sun_info()`.

**Листинг 11.15. 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`. |

***Таблица 11.3*** *(окончание)*

| **Функция** | **Описание** |
|---|---|
| | Может отформатировать произвольную дату, если она передана в качестве второго необязательного параметра `$timestamp`. В отличие от функции `date()` возвращает лишь целые значения |
| `checkdate($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 |
| 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 |

***Таблица 11.4*** *(окончание)*

| Символ | Описание | Пример возвращаемого значения |
|---|---|---|
| `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 |
| `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 11.16 приводится пример использования функции `date()`.

**Листинг 11.16. 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 11.17 выводит приветствие в зависимости от времени суток.

**Листинг 11.17. 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 локальные дату и время (с учетом часовой зоны), разработчикам доступна функция `gmdate()`, которая возвращает время по Гринвичу (листинг 11.18).

**Листинг 11.18. 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 11.19 приводится пример использования функции `checkdate()`.

**Листинг 11.19. 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 `$timestamp` позволяет задать время (в секундах с 00:00 1 января 1970 года). При его отсутствии функция возвращает текущее время. Параметр `$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 в текущей локали |
| `%B` | Полное название месяца 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 |

***Таблица 11.5*** *(окончание)*

| Символ | Описание |
|---|---|
| `%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.) |
| `%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 11.20 приводится пример использования функции `strftime()`.

**Листинг 11.20. 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 11.21 перед вызовом функции устанавливается русская локаль, поэтому в дате выводится сначала день, потом месяц, затем год вместо последовательности "месяц, день, год", характерной для американской локали.

**Листинг 11.21. 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 текущие время и дату в виде ассоциативного массива и имеет следующий синтаксис:

`getdate([$timestamp])`

Через необязательный параметр `$timestamp` функции можно передать время (в секундах с 00:00 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 |

***Таблица 11.6*** *(окончание)*

| Ключ | Описание | Диапазон |
|---|---|---|
| "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 | Платформо-зависимый, в большинстве случаев от –2 147 483 648 до 2 147 483 647 |

В листинге 11.22 приводится пример использования функции `getdate()`.

**Листинг 11.22. 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 11.23 приводится пример совместного использования функций `strftime()` и `strptime()`.

### *Замечание*

Функция `strptime()` не поддерживается в операционной системе Windows.

**Листинг 11.23. 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 11.23 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"` | Годы, 4 числа | Например, 1999 или 2008 |
| `"month"` | Месяцы | От 1 до 12 |
| `"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 11.24 приводится пример использования функции `date_parse()`.

**Листинг 11.24. 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 11.24 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", и 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 текущие 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 11.25 приводится пример использования функции `localtime()`.

**Листинг 11.25. 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 11.25 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 | π/2 |
| `M_PI_4` | 0.78539816339744830962 | π/4 |

***Таблица 12.1*** *(окончание)*

| Константа | Значение | Математическое значение |
|---|---|---|
| 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 | $\sqrt{2}$ |
| 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 бóльшим (или меньшим), массив признается бóльшим (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, $max])` | Генерирует случайное число в интервале от `$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. |

**Таблица 12.3** *(окончание)*

| Функция | Описание |
|---|---|
| | 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 значений 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 |

**Таблица 12.4** *(окончание)*

| Функция | Описание |
|---|---|
| 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 шестнадцатеричное 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 `$number` минус единица |
| `sqrt($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` плюс единица |

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 возвести отрицательное число в дробную степень, 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()`.

**Листинг 12.25. Вычисление степени $e^2$**

```
<?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 степени $e^2$**

```
<?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` начинает выдавать ошибочные значения, в то время как функция `expm1()` выдает точные (рис. 12.1).

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
?>
```

http://photon/ - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Назад Поиск Избранное

Адрес: http://photon/ Переход Ссылки

| 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 |

Готово Местная интрасеть

**Рис. 12.1.** Сравнение функций `expm1()` и `exp()` в окрестностях нуля

Как видно из листинга 12.28, при попытке вычисления корня квадратного из отрицательного числа возвращается константа `NAN`, сигнализирующая о том, что результат 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
?>
```

# 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 |

***Таблица 12.7*** *(окончание)*

| Функция | Описание |
|---|---|
| `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 12.32 приводится пример использования функции.

**Листинг 12.32. 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 12.33).

**Листинг 12.33. 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 12.34 приводится пример использования функции `sin()`.

**Листинг 12.34. 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 (листинг 12.35).

***Замечание***

Функция `deg2rad()` описывается в *разд. 12.4*.

**Листинг 12.35. Совместное использование функций `sin()` и `deg2rad()`**

```
<?php
  echo sin(deg2rad(90)); // 1
?>
```

# 12.8. Информационные функции

Информационные функции позволяют ответить на вопрос, является ли содержимое переменной 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($value)` | Синоним для функции `is_int()` |
| `is_long($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` |

***Таблица 12.8*** *(окончание)*

| Функция | Описание |
|---|---|
| `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 12.36).

**Листинг 12.36. 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 12.36 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 12.37).

**Листинг 12.37. 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 12.37 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 12.38).

**Листинг 12.38. 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 12.38 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 12.39 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 12.39. 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 12.39 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 результат `NAN` (листинг 12.40).

> ***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 12.40. 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, являются 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 12.41 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 12.41. 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 12.41 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 12.42).

**Листинг 12.42. 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 дает в результате ноль (листинг 12.43).

**Листинг 12.43. Использование бесконечных чисел в выражениях**

```
<?php
  echo 1/pow(0, -1);          // 0
  echo 123.24e306/123.24e308; // 0
  echo 123.24e308/123.24e306; // INF
?>
```

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, информации 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 (\) слэш. В UNIX-подобных операционных системах используется только прямой слэш.

*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.

## 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 содержимым файла.

***Таблица 13.1.*** *Функции создания файлов*

| Функция | Описание |
|---|---|
| `fopen($filename, $mode [, $use_include_path [, $context]])` | Открывает файл с именем `$filename` в режиме `$mode`. Некоторые режимы позволяют создать файл, некоторые требуют, чтобы файл уже существовал. Необязательный параметр `$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 флоппи-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 разных операционных системах используются различные разделители между каталогами и файлами. В UNIX-подобных операционных системах в качестве разделителя используется прямой слэш (/), в Windows — обратный (\). Обратный слэш традиционно применяется 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 использовать как прямой слэш (/), так и обратный слэш (\). Однако при использовании обратного слэша 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 слэша 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 слэшами (/), именно они и будут использоваться при дальнейшем изложении.

**Рис. 13.1.** Гипотетическая структура файловой системы

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. 13.5*.

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 приводится 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 предваряется ни буквой раздела, ни символом 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 (`..`), обозначающих родительский каталог (листинг 13.3).

**ЗАМЕЧАНИЕ**

Помимо родительского каталога, обозначающегося двумя точками (`..`), большинство файловых систем поддерживает текущий каталог, который обозначается одной точкой.

**Листинг 13.3. Использование родительской 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. Одна-

ко 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, он 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 используется 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 загруженный файл 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, осуществляющие 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 `$mode`. Некоторые режимы позволяют создать файл, некоторые требуют, чтобы файл уже существовал. Необязательный параметр `$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 `$handler` указывает на конец файла, и `FALSE` в противном случае |
| `fgetc($handle)` | Считывает и возвращает в качестве результата из открытого файла `$handle` один символ |
| `fgets($handle [, $length])` | Считывает и возвращает в качестве результата из открытого файла `$handle` строку. Чтение заканчивается по достижении строки длины `$handle` – 1 (по умолчанию `$handle` принимает значение 1000), по достижении символа перевода строки или символа конца файла |

***Таблица 13.5*** *(продолжение)*

| Функция | Описание |
|---|---|
| `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 `$length` символов. В отличие от функции `fgets()` переводы строк не прекращают чтение |
| `fscanf($handle, $format [, ...])` | Аналогична функции форматного разбора строки `sscanf()`, рассмотренной в *разд. 9.13*, однако чтение и разбор строки по форматному правилу `$format` осуществляются 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 |
| `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 |
| `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. |

***Таблица 13.5*** *(окончание)*

| **Функция** | **Описание** |
|---|---|
| | 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 достижении этого количества запись останавливается). В случае успеха функция возвращает количество записанных байтов, в случае неудачи — `FALSE` |
| `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 `$context` позволяет задать HTTP-заголовки, отправляемые сервером при обращении к сетевому файлу |
| `fflush($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 |
| `flock($handle, $operation [, $wouldblock])` | Устанавливает для открытого файла с дескриптором `$handle` режим блокировки `$operation`. Необязательный параметр `$wouldblock` позволяет выяснить успешность операции |
| `ftruncate($handle, $size)` | Уменьшает размер файла с дескриптором `$handle` до `$size` байтов. Возвращает `TRUE` в случае успеха и `FALSE` в противном случае |

При открытии файла при помощи функции `fopen()` он может существовать или отсутствовать, а 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 производятся с текущего положения файлового указателя от начала к концу. Чтение и запись файлов в обратном порядке не допускаются.

**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 достигаем 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 способ, 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 достигаем 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 оперируют текстовыми файлами, которые удобнее читать построчно. Для этого часто используют функцию `fgets()`, которая имеет следующий синтаксис:

```
fgets($handle [, $length])
```

Функция считывает из открытого файла `$handler` строку. Чтение заканчивается по достижении строки длины `$handle` – 1 (по умолчанию `$handle` принимает значение 1000), по достижении 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 достигаем 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 достигаем 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 возвращает объем файла в байтах. 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 паке-

та, и при слишком большой его величине передача файла захлебывается. Наиболее оптимальным размером при загрузке файла через сеть считается размер 4096 байтов (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 достигаем 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 работы скрипта 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 массива**

```
<?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 |

***Таблица 13.7*** *(окончание)*

| Значение | Описание |
|---|---|
| FILE_IGNORE_NEW_LINES | Не помещать в конец каждого элемента результирующего массива пустую строку |
| 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 3.18.

**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 достижении 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 различных операционных системах используются разные обозначения конца строки. В UNIX-подобных операционных системах это последовательность `\n`, в Windows — `\r\n`, в MacOS — `\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 обрабатывают 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 при помощи 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 13.22.

**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 встречалась 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 посредством 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 вовремя сброшена на диск, если своевременно закрыть файл 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 его размер 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. 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. Этот 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 никакая запись |
| `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 13.32 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 13.32. 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 $filename");
  // Сбрасываем содержимое буфера на диск
  fflush($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 13.32 можно переписать, исключив вызов функции `flock()`, снимающей блокировку, и функции `fflush()`, сбрасывающей данные из буфера на жесткий диск (листинг 13.33).

**Листинг 13.33. 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 13.34 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 удержива-

ется разделяемой блокировкой — PHP заботится о предоставлении доступа для обоих видов блокировок.

**Листинг 13.34. 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 в исходе операции можно по третьему параметру функции, который принимает значение `TRUE`, если блокировка получена, и `FALSE` в противном случае. В листинге 13.35 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 13.35. 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);
  }
?>
```

## 13.3.6. 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, т. е. все рассмотренные выше функции оперировали файловым указателем неявно. Это не всегда удобно, поэтому PHP предоставляет разработчикам функции для явного управления положением файлового указателя (табл. 13.10).

***Таблица 13.10.*** *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 `$offset` байтов относительно позиции `$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 13.36), что файловый указатель находится в начале файла (функция возвращает 0).

**Листинг 13.36. 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 13.36 выведет два положения 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 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 снова. Однако проще всего возвратить файловый указатель в начало файла при помощи функции `rewind()`. Функция принимает в качестве единственного параметра дескриптор открытого файла. В листинге 13.37 приводится пример скрипта, который выводит из текстового файла три последних строки. Неизвестно, сколько байтов содержится в одной строке, поэтому необходимо их предварительно посчитать (в результате чего файловый указатель оказывается в конце), а потом передвинуть файловый указатель в начало файла, чтобы повторно отобрать только последние три строки.

**Листинг 13.37. 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 `$offset` байтов относительно позиции `$whence`. Параметр `$whence` может принимать значения из табл. 13.11. В случае успеха возвращает 0, в случае неудачи –1.

***Таблица 13.11.*** *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 13.38 приводится пример использовании функции `fseek()` — на только что записанную в файл строку накладывается дополнительная строка.

**Листинг 13.38. 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!");

  // Перемещаем 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 13.39 демонстрируется скрипт, позволяющий создать файл из 10 000 символов за счет перемещения файлового указателя за конец файла.

**Листинг 13.39. 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 9999 позиций
  // от начала файла
  fseek($fd, 9999, SEEK_END);

  // Записываем пробел
  fwrite($fd, " ");

  // Закрываем файл
  fclose($fd);
?>
```

# 13.4. 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 — 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. 13.12 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. 13.12 в этой операционной системе не имеет смысла.

***Таблица 13.12.*** *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 |

***Таблица 13.12*** *(окончание)*

| Функция | Описание |
|---|---|
| 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 $*filename* доступен для чтения, в противном случае возвращает FALSE |
| is_writable($*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 13.40 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 13.40. Использование функции `chmod()`**

```
<?php
  chmod("index.php", 0644);
?>
```

Как видно из листинга 13.40, 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 13.41).

**Листинг 13.41. 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. 13.13 приводится соответствие масок различным типам файлов.

***Таблица 3.13.*** *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 оператором поразрядного пересечения `&` (листинг 13.42).

### *Замечание*

Поразрядные операторы подробно рассматриваются в *разд. 5.4*.

**Листинг 13.42. 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 13.43).

**Листинг 13.43. 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>";
?>
```

# 13.5. 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. 13.14).

***Таблица 13.14.*** *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. |

***Таблица 13.14*** *(окончание)*

| Функция | Описание |
|---|---|
| | 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` |
| `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 с 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 13.44).

**Листинг 13.44. Использование функции `getcwd()`**

```
<?php
  // Получаем абсолютный путь к текущему каталогу
  echo getcwd();
?>
```

При необходимости можно назначить новый текущий каталог при помощи функции `chdir()` (листинг 13.45). 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 13.45. Использование функции `chdir()`**

```
<?php
  // Устанавливаем новый каталог по умолчанию
  chdir("/var/www/html/test/");
?>
```

В листинге 13.46 при помощи функции `mkdir()` в текущем каталоге создается новый подкаталог 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. 13.4*.

**Листинг 13.46. 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 13.47).

**Листинг 13.47. 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 каталог — функция не удалит данный 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 13.48.

**Листинг 13.48. 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 13.49 приводится скрипт, осуществляющий копирование вложенного каталога.

**Листинг 13.49. Рекурсивное копирование каталога**

```
<?php
  // Копируем содержимое каталога home в home2
  lowering("home","home2");
  ////////////////////////////////////////////////////////
  // Функция рекурсивного 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 13.50 демонстрируется скрипт, позволяющий вывести содержимое текущего каталога.

**Листинг 13.50. 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. 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 13.51 демонстрируется простой пример, возвращающий массив с именами файлов и каталогов в заданном каталоге.

**Листинг 13.51. 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 13.51 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. 13.15, позволяя задать режим интерпретации шаблона.

***Таблица 13.15.*** *Возможные значения параметра `$flags` функции `glob()`*

| Значение | Описание |
|---|---|
| GLOB_MARK | Добавляет прямой слэш (/) в UNIX-подобных операционных системах и обратный слэш (\) в Windows к тем элементам результирующего массива, которые являются каталогами |

***Таблица 13.15*** *(окончание)*

| Значение | Описание |
|---|---|
| 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 слэша. 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 листинг 13.52 демонстрируется пример использования функции glob().

**Листинг 13.52. 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>";
  }
?>
```

## Глава 14

# HTTP-заголовки

*Протокол* — это набор синтаксических и семантических правил, использующихся при обмене данными между двумя компьютерами, который 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 cookie и т. п. Язык разметки HTML и серверный язык PHP создавались таким образом, чтобы разработчик мог работать без знания HTTP — это позволяет 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).

**Рис. 14.1.** HTTP-документ состоит из HTTP-заголовков и тела документа

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.

Браузер, получая HTTP-заголовки, автоматически выполняет предписания, даже не показывая посетителю их содержимое. Таким образом, HTTP-заголовки служат своеобразной мета-информацией, 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, осуществляемая 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).

# 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. HTTP-заголовки согласно протоколу 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 функций `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", сообщающее, 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 неполный 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 использование 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 медиа-типа, 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 изме-

няющейся 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, кэширование, 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 игнорируют 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 достижении 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`. Данным параметром задается 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, равное 30 минутам
  $cache_expire =  session_cache_expire(30*60);
  echo $cache_expire
?>
```

## Глава 15

# Cookie

HTTP-протокол, лежащий в основе Интернета, не сохраняет информацию 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 были введены механизмы 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 из клиентов.

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 подобрано адекватного русского перевода, поэтому в данной 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*;

- `$expire` — время в секундах, прошедшее с 0 часов 00 минут 1 января 1970 года. По истечении этого времени cookie удаляется с машины клиента;
- `$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 видны cookies**

```
<?php
  setcookie("name", "value", time() + 600, "/", "www.softtime.ru");
?>
```

Уничтожить cookie можно, установив нулевое время жизни, как это сделано в листинге 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`, сообщающая скрипту, что проверка выполнена и необходимо выяснить, 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`.

### *Замечание*

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.

### *Замечание*

Директива `session.save_path` в конфигурационном файле php.ini позволяет задать путь к каталогу, в котором сохраняются файлы сессий; это может быть удоб-

ным для отладки Web-приложений на локальном сервере. Если данная директива не задана, сессии хранятся в оперативной памяти сервера.

Для работы с сессиями предназначен специальный набор функций, представленных в табл. 16.1.

***Таблица 16.1.*** *Функции для работы с сессиями*

| **Функция** | **Описание** |
|---|---|
| `session_start()` | Инициирует сессию. Любая страница, использующая сессии, 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 |

***Таблица 16.1*** *(окончание)*

| Функция | Описание |
|---|---|
| `session_name($name])` | Позволяет создать пространство с именем `$name`, данные из разных пространств имен обрабатываются независимо. Таким образом можно разделить приложения, использующие одинаковые ключи для сессионных данных |
| `session_get_cookie_params()` | Данные сессии хранятся на сервере, однако уникальный идентификатор сессии SID передается клиенту через cookie. Функция `session_get_cookie_params()` возвращает массив с параметрами cookie, используемого 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, в который будет помещаться сериализованный массив `$_SESSION`. 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` сохраняются 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 из листинга 16.3 может быть строка:

```
a27e8c4724f07cae913c50e55cea1b38
```

## Глава 17

# Электронная почта

Почта является основным средством общения для миллионов пользователей Интернета. Большинство пользователей Интернета привыкли именно к этому виду общения, поэтому для успешной разработки сайтов необходимо владеть приемами отправки и получения почты через PHP.

## 17.1. 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 осуществляются 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 17.1.

**Листинг 17.1. 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 17.2).

**Листинг 17.2. 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 17.2 оно будет представлено как письмо от пользователя `server` с электронным адресом `someone@somewhere.ru`.

# 17.2. 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 17.3 представлен вариант кода, осуществляющего рассылку с использованием первого приема.

**Листинг 17.3. 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 17.4.

***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 17.4. 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 17.4, в цикле формируются лишь почтовые заголовки. Функция `mail()`, отправляющая письма, вызывается только один раз.

# Глава 18

# Объектно-ориентированные возможности PHP

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"[1].

## 18.1. 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 свои собственные типы данных (классы) и объявлять "переменные" таких типов данных (объекты). Создавая собственные 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

[1] Кузнецов М., Симдянов И. Объектно-ориентированное программирование на PHP. — СПб.: БХВ-Петербург, 2007.

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 абстрактного типа данных. 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.

**Рис. 18.1.** Переменные объявляются при помощи типа, объекты — при помощи класса

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-ориентиро-

ванный 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. 18.1).

## 18.2. 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 18.1 приводится общий синтаксис объявления класса.

**Листинг 18.1. 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.

## 18.3. Создание объекта

Объект объявляется при помощи ключевого слова `new`, за которым следует имя класса. В листинге 18.2 создается пустой класс `emp` и объявляется объект `$obj` данного класса.

### Замечание

После имени класса могут следовать необязательные круглые скобки.

**Листинг 18.2. 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 `$obj` является обычной переменной PHP, правда, со специфичными свойствами. Как и любую другую переменную, объект можно передавать в качестве параметра функции, использовать как элемент массива или присваивать ему новое значение. В листинге 18.3 приводится пример, в котором объекту `$obj` по мере выполнения программы присваивается некоторое числовое значение, и `$obj` становится переменной числового типа.

**Листинг 18.3. 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 18.4). Использование конструкции `unset()` может быть полезным, если объект занимает большой объем оперативной памяти, и ее следует освободить, чтобы не допустить переполнения.

**Листинг 18.4. Объект — это обычная переменная**

```
<?php
  // Объявление класса
```

```
  class emp {}
  // Создание объекта класса
  $obj = new emp();
  // Уничтожение объекта
  unset($obj);
?>
```

В отличие от других языков программирования, объект в PHP, объявленный внутри блока, ограниченного фигурными скобками, существует и вне его, не подвергаясь уничтожению при выходе из блока.

## 18.4. 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 18.5 представлен класс "сотрудник" (`employee`), который содержит четыре члена:

- `$surname` — фамилия сотрудника;
- `$name` — имя сотрудника;

- `$patronymic` — отчество сотрудника;
- `$age` — возраст сотрудника.

**Листинг 18.5. 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 18.6 объявляется объект `$emp` класса `employee`, при этом члены класса получают необходимые значения и выводятся в окно браузера.

**Листинг 18.6. 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 18.6, при обращении к члену класса используется оператор `->`, после которого следует ввести имя этого члена. Заметьте, что 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. 18.2 приводится схематическое изображение процесса доступа к членам объекта, снабженным разными спецификаторами доступа.

**Рис. 18.2.** Открытые и закрытые члены объекта

# 18.5. 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 18.7 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 18.7 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 18.7. Объявление метода класса**

```
<?php
  class cls_mth
```

```
  {
    public function show_message()
    {
      echo "Hello, world!";
    }
  }

  $obj = new cls_mth();

  $obj->show_message();
?>
```

В листинге 18.7 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 18.5. Класс содержит закрытую переменную `$age`, определяющую возраст сотрудника. Создадим четыре метода (листинг 18.8):

- `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 18.8. 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 18.8, 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 18.8 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 18.9 приводится пример использования класса `employee` из листинга 18.8.

**Листинг 18.9. 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)
?>
```

# 18.6. Специальные методы класса

Помимо методов, создаваемых разработчиком класса, объектно-ориентированная модель PHP предоставляет разработчику специальные методы. Эти методы позволяют разработчику задать неявные свойства поведения объектов и выполняются, как правило, автоматически.

В табл. 18.1 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 18.1.*** *Специальные методы классов*

| Метод | Описание |
|---|---|
| `__construct()` | Конструктор класса; метод, который автоматически выполняется в момент создания объекта до вызова всех остальных методов класса |
| `__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 свойству |
| `__get()` | Аксессор; метод, предназначенный для чтения свойства |
| `__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()` |

# 18.7. 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. 18.2); часть из них более подробно обсуждаются в последующих разделах данной главы.

***Таблица 18.2.*** *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()` |
| `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. |

***Таблица 18.2*** *(окончание)*

| Функция | Описание |
|---|---|
| | 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 |
| `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 |

## 18.8. 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 18.10 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 5 допускается использование старого метода объявления конструктора, однако такой подход признан устаревшим и может быть исключен из следующих версий языка.

**Листинг 18.10. 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 18.11 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 — другой 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 18.11).

**Листинг 18.11. 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 18.11 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.

# 18.9. 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 18.12 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 оси ординат соответственно.

**Листинг 18.12. 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 18.13 демонстрируется использование конструктора, инициализирующего закрытые члены класса.

**Листинг 18.13. 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 18.14 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 18.14. 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.

## 18.10. 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 18.15 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 18.15. 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 18.16. приводится пример создания объекта класса `cls`.

**Листинг 18.16. 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 18.16, 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.

## 18.11. 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 18.17).

**ЗАМЕЧАНИЕ**

Функция `__autoload()` не является методом класса — это независимая функция, которую PHP-разработчик может перегружать.

**Листинг 18.17. Автозагрузка классов**

```
<?php
  // Функция автозагрузки классов
  function __autoload($classname)
  {
    require_once("class/class.$classname.php");
  }
  ...
?>
```

## 18.12. 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 18.18 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 18.18. 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 18.18, класс `cls` перехватывает все обращения к членам объекта и создает соответствующий элемент в закрытом массиве `$arr`. В листинге 18.19 демонстрируется, как обращение к члену `$name` приводит к созданию соответствующего элемента массива.

**Листинг 18.19. 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 18.19 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.

## 18.13. 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 18.20 представлена реализация метода `__isset()` для класса `cls` из листинга 18.18.

> ***Замечание***
>
> Метод `__isset()` доступен в PHP, начиная с версии 5.1.0.

**Листинг 18.20. Перегрузка метода `__isset()`**

```
<?php
  class cls
  {
    ...
    private function __isset($index)
    {
```

```
      return isset($this->$index);
    }
    ...
  }
?>
```

# 18.14. Уничтожение члена класса. Метод __*unset()*

Для уничтожения членов класса служит специальный метод `__unset()`. Вернемся к классу `cls`, рассмотренному в листинге 18.18. 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 18.21).

> ***Замечание***
>
> Метод `__unset()` доступен в PHP, начиная с версии 5.1.0.

**Листинг 18.21. 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 18.22).

**Листинг 18.22. 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 18.22 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.

## 18.15. 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.

В отличие от других С-подобных языков программирования в PHP отсутствуют функции с переменным количеством параметров. Тем не менее, при помощи специального метода `__call()` можно эмулировать наличие таких методов в классе. В листинге 18.23 представлен класс `minmax`, который предоставляет пользователю два метода — `min()` и `max()`, принимающих произвольное количество числовых параметров и определяющих минимальное и максимальное значения соответственно.

**Листинг 18.23. 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 18.23 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 передан ни один аргумент, возвращается значение `false`.

В листинге 18.24 приводится пример использования класса `minmax` для получения максимального и минимального значений последовательности.

**Листинг 18.24. 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 18.25).

**Листинг 18.25. 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 18.25 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()".

## 18.16. Интерполяция объекта. Метод *__toString()*

Специальный метод `__toString()` позволяет интерполировать (подставлять) объект в строку. Для подстановки значений переменных необходимо заключить строку в двойные кавычки (листинг 18.26).

> ***Замечание***
>
> Следует отметить, что значение переменной не интерполируется, если вместо двойных кавычек используются одинарные.

**Листинг 18.26. 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 с помощью метода `__toString()`, который преобразует объект в строку.

Модифицируем класс `employee` (см. листинг 18.8) таким образом, чтобы его подстановка в строку приводила к выводу фамилии сотрудника и его инициалов (листинг 18.27).

***Замечание***

Следует обратить внимание, что метод `__toString()` выводит результат при помощи конструкции `return`, а не `echo`.

**Листинг 18.27. 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 18.28 приводится пример интерполяции объекта класса `employee`, при этом вместо объекта `$obj` будет подставлена фраза `Борисов Игорь Иванович`.

**Листинг 18.28. 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".

# 18.17. 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, приходиться расплачиваться. В случае, рассмотренном в предыдущем разделе, платой служит более сложная организации кода и более длительные сроки его подготовки.

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 18.29 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 18.29. 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 18.30 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 18.30. 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 18.30, 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.

## 18.18. 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 18.31 приводится пример попытки обратиться к закрытому члену `$var` базового класса `base` из конструктора производного класса `derived`.

**Листинг 18.31. 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 закрытому 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 18.32 на примерах базового класса `base` и производного класса `derived` демонстрируется использование защищенного члена `$var`.

**Листинг 18.32. 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 18.32 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 18.33).

**Листинг 18.33. 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.

## 18.19. 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 18.34 приводится пример перегрузки метода `print_var()` в производном классе `derived`, который наследуется от базового класса `base`.

**Листинг 18.34. 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 18.34 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 18.35 представлена перегрузка метода `print_var()`, при которой метод производного класса сначала вызывает метод базового класса.

**Листинг 18.35. 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 18.35 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.

# 18.20. 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 "транспортное средство" 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. 18.3).

Скорее всего метод `move()` придется переопределить для классов "автомобиль", "воздушный транспорт", "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.

**Рис. 18.3.** Схема иерархии объектов "транспорт"

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. 18.4.

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. По-

этому 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).

**Рис. 18.4.** Иерархия классов постраничной навигации

Эти методы не имеют реализации в классе `pager` (пустые) и перегружаются производными классами, которые формируют параметры в зависимости от источника.

# 18.21. Абстрактные классы

Возвращаясь к иерархии классов постраничной навигации, рассмотренной в предыдущем разделе, 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 18.36).

**Листинг 18.36. Объявление абстрактного класса**

```
<?php
  abstract class pager
  {
    ...
  }
?>
```

Теперь попытка создать экземпляр класса `pager` (листинг 18.37) будет заканчиваться сообщением об ошибке "Fatal error: Cannot instantiate abstract class pager" ("Невозможно объявить объект абстрактного класса").

**Листинг 18.37. 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, возвратимся к транспортной сети (см. рис. 18.3). Объекты "транспорт", "автомобиль", "воздушный транспорт", "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. (рис. 18.5).

**Рис. 18.5.** Схема транспортной сети

# 18.22. Абстрактные методы

Если во время работы над сайтом появляется новый источник данных, к которому нужно будет применить постраничную навигацию *(см. разд. 18.20)*, 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 18.38 методы `get_total()`, `get_pnumber()`, `get_page_link()` и `get_parameters()` класса `pager` объявляются абстрактными.

**Листинг 18.38. 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.

# 18.23. Создание интерфейса

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. При этом нельзя объявить объекты базовых классов, поскольку они не могут 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 состоящая из абстрактных методов. Это позволяет внести ясность в процесс разработки: классы задают поведение объектов, а интерфейсы — поведение группы классов. Класс, реализующий интерфейс, должен перегрузить 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.

Для создания интерфейса используется ключевое слово `interface`. В листинге 18.39 демонстрируется пример интерфейса `pager` для реализации набора классов постраничной навигации, рассматривавшихся в *разд. 18.21*. Напомним, что 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 отрытые методы; при попытке добавить в интерфейс закрытый или защищенный метод возникает сообщение об ошибке.

**Листинг 18.39. 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 `extends` используется ключевое слово `implements`. В листинге 18.40 объявляются два класса — `pager_sample` и `pager_example`, которые реализуют интерфейс `pager`.

**Листинг 18.40. 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").

## 18.24. 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 18.41).

> ***Замечание***
>
> Именно возможность реализации нескольких интерфейсов позволяет скомпенсировать отсутствие множественного наследования в объектно-ориентированной модели языка.

**Листинг 18.41. 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 18.41, класс `cls` реализует сразу три интерфейса: `pager`, `print_list` и `document`.

## 18.25. Наследование интерфейсов

Интерфейсы, так же как и классы, могут наследовать друг другу при помощи ключевого слова `extends`. В листинге 18.42 производный интерфейс `extended_pager` наследует методы базового интерфейса `pager`.

**Листинг 18.42. 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();
  }
?>
```

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 18.43).

**Листинг 18.43. 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 в классе `pager_class` хотя бы одного метода из интерфейсов `pager` и `extended_pager` приведет к ошибке.

# 18.26. 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 18.44).

**Листинг 18.44. Объявление статического члена класса**

```
<?php
  class cls
```

```
  {
     public static $staticvar = 100;
  }

  echo cls::$staticvar; // 100
?>
```

К статическим членам класса нельзя обращаться через префикс `$this->`, и они не отображаются в списках членов объектов при выводе дампа. В листинге 18.45 приводится модифицированный вариант класса `cls`, в котором метод `set_staticvar()` осуществляет попытку изменить значение статического члена класса `$staticvar` при помощи префикса `$this->`.

**Листинг 18.45. 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 18.45 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. 18.6).

**Рис. 18.6.** 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 18.46 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 18.46. 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 18.47.

**Листинг 18.47. 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
?>
```

# 18.27. 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 18.48 приводится пример класса `cls`, содержащего единственный статический метод `static_method()`.

**Листинг 18.48. 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.

# 18.28. Константы класса

Наряду с членами классы могут содержать константы, которые определяются при помощи ключевого слова `const`. В листинге 18.49 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 18.49. 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 указываться в любом регистре, однако традиционно используется верхний. Лучше придерживаться данной традиции, т. к. это позволяет значительно увеличить читабельность кода (большинство программистов будут 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 18.50).

**ЗАМЕЧАНИЕ**

При проверке классовых констант следует в обязательном порядке использовать оператор разрешения области видимости `::` и имя класса.

**Листинг 18.50. 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>";
?>
```

# 18.29. Предопределенные константы

Помимо констант, которые разработчик может вводить в класс, существуют предопределенные константы, которые определяет PHP-интерпретатор (табл. 18.3).

***Таблица 18.3.*** *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.

Для использования в классах предусмотрены также специализированные константы: `__CLASS__` и `__METHOD__` (листинг 18.51).

**Листинг 18.51. 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 18.51 будут следующие строки:

```
Вызов метода cls::get_point
класса cls
в файле D:\main\oop\06\class.cls.php
в строке 9
```

## 18.30. *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 18.52 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 18.52. 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 18.53 в качестве `final`-метода объявляется конструктор класса `base`, в результате чего становится невозможным наследование производного класса `derived` с переопределенным конструктором.

**Листинг 18.53. 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 18.53, не является надежным способом запрета наследования от класса `base`. Достаточно не переопределять конструктор производного класса `derived`, чтобы наследование стало возможным (листинг 18.54).

**Листинг 18.54. 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.

## 18.31. *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 18.55).

> ***Замечание***
>
> Класс не может одновременно являться и абстрактным (`abstract`), и `final`-классом.

**Листинг 18.55. 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)").

# 18.32. 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 18.56 представлена операция присвоения одного объекта класса `cls` другому, при этом изменение члена класса нового объекта `$new_obj` отражается на старом объекте `$obj`.

**Листинг 18.56. 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. 18.7.

**Рис. 18.7.** 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 18.57).

**Листинг 18.57. 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. 18.8.

**Рис. 18.8.** Получение независимой копии объекта `$obj` посредством клонирования

# 18.33. 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 18.58).

**Листинг 18.58. 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 18.58 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 18.59 приводится пример улучшенного класса `counter`, который подсчитывает количество созданных объектов и подробно рассматривался в *разд. 18.26*. 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 18.59. 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.

## 18.34. 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 18.60 приводится возможная реализация класса `user`.

**Листинг 18.60. 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 18.61 демонстрируется скрипт, который подвергает сериализации объект `$obj` класса `user`.

**Листинг 18.61. 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>";

  // Сериализуем объект
  $object = serialize($obj);

  // Выводим сериализованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 18.61 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;}
```

**Рис. 18.9.** Схема сериализации объекта

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. 18.9).

В листинге 18.62 представлен модифицированный вариант класса `user`, который обнуляет значение члена `$password` при сериализации объекта.

**Листинг 18.62. 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 18.63).

**Листинг 18.63. 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>";

  // Сериализуем объект
  $object = serialize($obj);

  // Выводим дамп объекта
  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  // Выводим сериализованный объект
  echo $object;
?>
```

Результатом работы скрипта из листинга 18.63 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 18.64).

**Листинг 18.64. 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 18.64 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 `$time` авторизации пользователя. Разумно при этом обновить член `$time` при вызове функции `unserialize()`. Для решения этой задачи предназначен специальный метод `__wakeup()`, который вызывается сразу после восстановления объекта (рис. 18.10).

**Рис. 18.10.** Схема восстановления объекта

В листинге 18.65 представлен модифицированный класс `user`, который обеспечивает обновление времени авторизации пользователя при восстановлении объекта.

**Листинг 18.65. 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 18.66 представлен скрипт, который демонстрирует восстановление объекта из сериализованного состояния с обновлением члена `$time`.

**Листинг 18.66. 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 18.66 будет следующий дамп объекта класса `user`:

```
user Object
(
    [name] => nick
    [password] => password
    [referrer] => /oop/07/index.php
    [time] => 1177677626
)
```

В листинге 18.66 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. Один из таких вариантов обсуждается в следующем разделе.

# 18.35. Синтаксис исключений

Исключения не являются непременным атрибутом объектно-ориентированного подхода де-юре, однако сопровождают каждый объектно-ориентированный язык де-факто. Дело в том, что объектно-ориентированный подход серьезно изменяет структуру кода: использующий его программист отчасти сам становится автором языка программирования в рамках предметной области, для которой разрабатываются классы. В свою очередь структурные изменения кода требуют новых способов обработки ошибок (нештатных ситуаций).

Создание класса — это всегда работа на абстрактном уровне: создается не конкретная область памяти, а только поведение объектов. Класс выступает инструментом, который, как и язык программирования, может применяться в совершенно разных областях и приложениях. Обработка нештатных ситуаций, ошибок как в коде, так и ввода данных, может быть различной для разных приложений: где-то достаточно вывести сообщение при помощи функции `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, а его предопределенность 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 должен сгенерировать исключение, а пользователь класса может его обработать по своему усмотрению.

Разработчики могут проектировать свои 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 к объектно-ориентированному подходу вовсе не означает, что следует пренебрегать процедурными средствами обработки ошибок. В PHP имеется развитая система отслеживания и контроля ошибок, и пренебрегать ею не стоит.

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 18.67).

**Листинг 18.67. 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 18.68 представлен скрипт, в котором по случайному закону либо генерируется, либо не генерируется исключение.

**Листинг 18.68. 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 слова `new` непосредственно при вызове оператора `throw`. Однако объект можно подготовить заранее (листинг 18.69).

**Листинг 18.69. Предварительное создание объекта исключения**

```
<?php
  try
  {
    if(rand(0,1))
    {
      $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.

## Глава 19

# 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.

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"[1], "MySQL 5" (серия "В подлиннике")[2] и "MySQL на примерах"[3].

# 19.1. 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 — 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, язык структурированных запросов).

Таким образом, для получения информации из базы данных необходимо направить ей запрос, созданный с использованием SQL, результатом выполнения которого будет результирующая таблица (рис. 19.1).

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 предоставляются 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);

[1] Кузнецов М., Симдянов И. Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2006.
[2] Кузнецов М., Симдянов И. MySQL 5. — СПб.: БХВ-Петербург, 2005.
[3] Кузнецов М., Симдянов И. MySQL на примерах. — СПб.: БХВ-Петербург, 2007.

- 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.

| | Таблица catalogs | |
|---|---|---|
| | id_catalog | name |
| | 1 | Процессоры |
| | 2 | Материнские платы |
| Строка | 3 | Видеоадаптеры |
| | 4 | Жесткие диски |
| | 5 | Оперативная память |

Столбец

**Рис. 19.1.** Таблица реляционной базы данных

На рис. 19.1. приведен пример таблицы `catalogs` базы данных электронного магазина изделий компьютерных комплектующих, которые подразделяются на разделы. Каждая строка этой таблицы представляет собой один вид товарной позиции, 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 (рис. 19.2).

**Рис. 19.2.** Связанные друг с другом таблицы

На рис. 19.2 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).

> ***Примечание***
>
> В математике таблица, все строки которой отличаются друг от друга, называется *отношением* (от англ. *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. Очень важно, что SQL-запрос возвращает таблицу: 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.

## 19.2. 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. 19.2 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. 19.2).

### Совет

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.

**Рис. 19.3.** Связь между таблицами `products` и `catalogs`

Рассмотрим, например, уже упоминавшуюся базу данных электронного магазина (рис. 19.3). Каждая из таблиц имеет свой первичный ключ, значение которого уни-

кально 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).

## 19.3. 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 19.1 приведен пример создания базы данных `wet`.

**Листинг 19.1. 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 19.1 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 19.2).

**Листинг 19.2. Использование оператора SHOW DATABASES**

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
| wet                |
+--------------------+
```

Как видно из листинга 19.2, 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.

Попробуйте 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 19.3).

**Листинг 19.3. 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 19.4).

> ***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 19.4. 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 19.5).

**Листинг 19.5. 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
```

# 19.4. 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 `USE`. В листинге 19.6 при помощи команды `USE` в качестве текущей выбирается база данных `test`.

**Листинг 19.6. 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 19.7).

> ***Замечание***
>
> Оператор `SHOW TABLES` выводит список таблиц в текущей базе данных.

**Листинг 19.7. 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 19.8).

**Листинг 19.8. 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 19.9 из таблицы `User` базы данных `mysql` извлекаются поля `User` и `Host`.

**Листинг 19.9. 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 19.10).

> ***Замечание***
>
> Более подробно каждая из функций листинга 19.10 обсуждается в следующих разделах текущей главы.

**Листинг 19.10. 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'");
?>
```

# 19.5. Типы данных

Как сообщалось ранее, таблицы состоят из столбцов и строк. Тип данных поля определяется столбцом. Список наиболее часто встречающихся типов приведен в табл. 19.1—19.3.

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 запятой, 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 19.1.*** *Числовые типы*

| **Тип** | **Объем занимаемой памяти** | **Диапазон допустимых значений** |
|---|---|---|
| TINYINT[(*M*)] | 1 байт | От –128 до 127 (от $-2^7$ до $2^7-1$)<br>От 0 до 255 (от 0 до $2^8-1$) |
| SMALLINT[(*M*)] | 2 байта | От –32 768 до 32 767 (от $-2^{15}$ до $2^{15}-1$)<br>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 $2^{23}-1$)<br>От 0 до 16 777 215 (от 0 до $2^{24}-1$) |
| INT[(*M*)],<br>INTEGER[(*M*)] | 4 байта | От –2 147 683 648 до 2147 683 647 (от $-2^{31}$ до $2^{31}-1$)<br>От 0 до 4 294 967 295 (от 0 до $2^{32}-1$) |
| BIGINT[(*M*)] | 8 байтов | От $-2^{63}$ до $2^{63}-1$<br>От 0 до $2^{64}$ |
| BIT[(*M*)] | (*M*+7)/8 байтов | От 1 до 64 бита, в зависимости от значения M |
| BOOL,<br>BOOLEAN | 1 байт | 0 или 1 |
| DECIMAL[(*M*[,*D*])],<br>DEC[(*M*[,*D*)]],<br>NUMERUC[(*M*[,*D*)]] | *M* + 2 байта | Повышенная точность; зависит от параметров *M* и *D* |
| FLOAT[(*M*,*D*)] | 4 байта | Минимальное значение $\pm 1{,}175494351 \cdot 10^{-39}$<br>Максимальное значение $\pm 3{,}402823466 \cdot 10^{38}$ |
| DOUBLE[(*M*,*D*)],<br>REAL[(*M*,*D*)],<br>DOUBLE PRECISION [(*M*,*D*)] | 8 байтов | Минимальное значение $\pm 2{,}2250738585072014 \cdot 10^{-308}$<br>Максимальное значение $\pm 1{,}797693134862315 \cdot 10^{308}$ |

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. 19.1 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-й 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. 19.2). 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 19.2.*** *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)<br>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:

```
'2008-04-06 02:04:08'
'2008-4-06 02:04:08'
'2008-4-6 02:04:08'
'2008-4-6 2:04:08'
'2008-4-6 2:4:08'
'2008-4-6 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:

```
'08-12-31 11:30:45'
'08.12.31 11+30+45'
'08/12/31 11*30*45'
'08@12@31 11^30^45'
```

Дата и время суток могут также быть представлены в форматах 'YYYYMMDDhhmmss' и 'YYMMDDhhmmss'. Например, строки '20080523091528' и '080523091528' аналогичны '2008-05-23 09:15:28', однако строка '081122129015' уже не может рассматриваться как дата и время суток, т. к. ожидаемое для минут зна-

чение равно 90 и выходит за допустимый интервал. Вместо строк допустимы и целочисленные значения, например, 20080523091528 и 080523091528 рассматриваются как '2008-05-23 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 отсутствующих величин будут подставлены нулевые значения.

Строковые типы данных, максимальный размер и требования к памяти приведены в табл. 19.3.

***Таблица 19.3.*** *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. 19.3, 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* байтов. В то же время тип `CHAR` обрабатывается эффективнее переменных типов, т. к. всегда заранее известно, где заканчивается очередной блок данных (табл. 19.4).

***Таблица 19.4.*** *Сравнение типов* `CHAR` *и* `VARCHAR`

| Значение | CHAR(4) | | VARCHAR(4) | |
|---|---|---|---|---|
| | Представ-ление | Занимаемый объем памяти, байт | Представ-ление | 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).

# 19.6. 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 19.11. 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 19.11. 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 19.12).

**Листинг 19.12. 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 19.13.

**Листинг 19.13. 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 19.14).

**Листинг 19.14. 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 19.15).

**Листинг 19.15. 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 оно 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 19.16.

**Листинг 19.16. 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 19.17.

**Листинг 19.17. 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. 19.5.

***Таблица 19.5.*** *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 |

***Таблица 19.5*** *(окончание)*

| Синтаксис | Описание команды |
|---|---|
| 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 19.18), разместив его после столбца `name`.

**Листинг 19.18. 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 19.19).

**Листинг 19.19. 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 19.20).

**Листинг 19.20. 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 19.21).

**Листинг 19.21. 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 19.22.

**Листинг 19.22. Удаление таблицы `forums`**

```
mysql> DROP TABLE forums;
```

# 19.7. Вставка числовых значений в таблицу

Для вставки записи в таблицу используется 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 19.23).

**Листинг 19.23. 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 19.24).

**Листинг 19.24. 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 19.25), синтаксис которого подробно разбирается далее в этой главе.

**Листинг 19.25. Просмотр содержимого таблицы**

```
SELECT * FROM tbl;
+----+--------+
| id | id_cat |
+----+--------+
| 10 |     20 |
| 20 |     10 |
| 30 |   NULL |
|  5 |   NULL |
+----+--------+
```

Рассмотрим различные формы оператора INSERT из листинга 19.24 более подробно. Первая форма оператора INSERT из листинга 19.24 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 следует задать порядок следования столбцов в дополнительных круглых скобках после имени таблицы. В листинге 19.24 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 19.24 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 (см. листинг 19.23). Эффекта последнего оператора можно добиться, если использовать

вместо значений ключевое слово DEFAULT. В листинге 19.26 оба оператора эквивалентны.

**Листинг 19.26. Использование ключевого слова DEFAULT**

```
INSERT INTO tbl () VALUES ();
INSERT INTO tbl (id, id_cat) VALUES (DEFAULT, DEFAULT);
```

Приведенная в листинге 19.26 форма оператора INSERT является стандартной и поддерживается всеми СУБД, которые реализуют стандарт SQL. Однако СУБД MySQL поддерживает альтернативный синтаксис оператора INSERT (листинг 19.27).

**Листинг 19.27. Альтернативный синтаксис оператора INSERT**

```
INSERT INTO tbl SET id = 40, id_cat = 50;
INSERT INTO tbl SET id = 50;
```

Как видно из листинга 19.27, значения присваиваются столбцам при помощи ключевого слова SET; не указанные в операторе столбцы принимают значение по умолчанию.

## 19.8. Вставка строковых значений в таблицу

Создадим таблицу catalogs, которая будет содержать два столбца: числовой столбец id_catalog и текстовый столбец name (листинг 19.28).

**Листинг 19.28. 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 19.29.

**Листинг 19.29. Добавление новой записи в таблицу catalogs**

```
INSERT INTO catalogs VALUES (1, 'Процессоры');
SELECT * FROM catalogs;
+------------+------------+
```

```
| id_catalog | name       |
+------------+------------+
|          1 | Процессоры |
+------------+------------+
```

Как видно из листинга 19.29, 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 19.30). Когда в текстовое поле необходимо вставить строку, содержащую двойные кавычки, можно использовать для обрамления строки одиночные кавычки и наоборот.

**Листинг 19.30. 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 19.31).

***Замечание***

При помещении в базу данных текста, набранного пользователем, всегда следует экранировать кавычки, для того чтобы предотвратить возникновение ошибки и атаку SQL-инъекцией.

**Листинг 19.31. Экранирование кавычек**

```
INSERT INTO catalogs VALUES (3, "Память \"DDR\"");
SELECT * FROM catalogs;
+------------+--------------+
| id_catalog | name         |
+------------+--------------+
```

```
|           1 | Процессоры   |
|           2 | Память       |
|           3 | Память "DDR" |
+------------+--------------+
```

# 19.9. 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 19.32).

**Листинг 19.32. Создание таблицы `tbl`**

```
CREATE TABLE tbl (
  id INT(11) NOT NULL,
  putdate DATETIME NOT NULL,
  lastdate DATE NOT NULL
);
```

В листинге 19.33 приводится пример вставки записи в таблицу, содержащую столбцы календарного типа.

**Листинг 19.33. Вставка календарных значений**

```
INSERT INTO tbl VALUES (1, '2008-01-03 0:00:00', '2008-01-03');
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2008-01-03 00:00:00 | 2008-01-03 |
+----+---------------------+------------+
```

Если столбцу передается "лишняя" информация, то она отбрасывается. В листинге 19.34 в столбец типа `DATE` передаются часы, минуты и секунды, однако в столбец попадает только информация о дате.

**Листинг 19.34. Отбрасывание "лишней" информации**

```
INSERT INTO tbl VALUES (2, '2008-01-03 0:00:00', '2008-01-03 0:00:00');
SELECT * FROM tbl;
```

```
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2008-01-03 00:00:00 | 2008-01-03 |
|  2 | 2008-01-03 00:00:00 | 2008-01-03 |
+----+---------------------+------------+
```

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 19.35 при помощи функции `NOW()` в таблицу `tbl` вставляется запись с текущей временной меткой.

**Листинг 19.35. Использование функции `NOW()`**

```
INSERT INTO tbl VALUES (3, NOW(), NOW());
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
|  1 | 2008-01-03 00:00:00 | 2008-01-03 |
|  2 | 2008-01-03 00:00:00 | 2008-01-03 |
|  3 | 2008-01-03 15:03:33 | 2008-01-03 |
+----+---------------------+------------+
```

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 19.36 в поле `putdate` помещается дата, эквивалентная началу 2008 года за вычетом 3 недель, а в поле `lastdate` помещается дата, равная текущей плюс 3 месяца.

**Листинг 19.36. Использование интервалов**

```
INSERT INTO tbl
VALUES (4, '2008-01-01 0:00:00' - INTERVAL 3 WEEK,
           NOW() + INTERVAL 3 MONTH);
SELECT * FROM tbl;
+----+---------------------+------------+
| id | putdate             | lastdate   |
+----+---------------------+------------+
```

```
|  1 | 2008-01-03 00:00:00 | 2008-01-03 |
|  2 | 2008-01-03 00:00:00 | 2008-01-03 |
|  3 | 2008-01-03 15:03:33 | 2008-01-03 |
|  4 | 2007-12-11 00:00:00 | 2008-04-03 |
+----+---------------------+------------+
```

Как видно из листинга 19.36, интервалы могут быть различного типа. Полный список допустимых интервалов приводится в табл. 19.6.

***Таблица 19.6.*** *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 микросекунды | 'mm:ss.xxxxxx' |
| MINUTE_SECOND | Минуты и секунды | 'mm:ss' |
| HOUR_MICROSECOND | Часы, минуты, секунды и микросекунды | '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' |

***Таблица 19.6*** *(окончание)*

| Тип | Описание | Формат ввода |
|---|---|---|
| DAY_HOUR | Дни и часы | 'DD hh' |
| YEAR_MONTH | Года и месяцы | 'YY-MM' |

# 19.10. 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 19.37).

**Листинг 19.37. 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 19.38. Использование ключевого слова IGNORE**

```
INSERT IGNORE INTO tbl VALUES (1, 'Видеоадаптеры');
SELECT * FROM tbl;
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Видеоадаптеры |
+----+---------------+
```

Как видно из листинга 19.38, генерации ошибки не происходит, тем не менее, новая запись также не добавляется.

# 19.11. 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 можно передать данному столбцу в качестве значения `NULL` или 0 — поле автоматически получит значение, равное максимальному значению в столбце, плюс единица. В листинге 19.39 создается таблица `tbl`, состоящая из первичного ключа `id` и текстового поля `name`. Первичный ключ `id` снабжен атрибутом `AUTO_INCREMENT`.

**Листинг 19.39. 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     |
+----+-------------------+
```

# 19.12. 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 19.40 добавляется 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 19.40. 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 19.41).

**Листинг 19.41. Изменение состава столбцов в многострочном операторе `INSERT`**

```
INSERT INTO catalogs (name) VALUES ('Процессоры'),
('Материнские платы'),
('Видеоадаптеры'),
('Жесткие диски'),
('Оперативная память');
```

# 19.13. 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 19.42 из таблицы `catalogs` удаляются записи, значение первичного ключа `id` которых больше двух.

**Листинг 19.42. Удаление записей из таблицы `catalogs`**

```
DELETE FROM catalogs WHERE id > 2;
SELECT * FROM catalogs;
+----+-------------------+---------------------+
| id | name              | putdate             |
+----+-------------------+---------------------+
|  1 | Процессоры        | 2008-01-19 21:40:30 |
|  2 | Материнские платы | 2008-02-19 10:30:30 |
+----+-------------------+---------------------+
```

Если в операторе `DELETE` отсутствует условие `WHERE`, из таблицы удаляются все записи (листинг 19.43).

**Листинг 19.43. 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 19.44 удаляется не более 3 записей таблицы `catalogs`.

**Листинг 19.44. 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 19.45.

**ЗАМЕЧАНИЕ**

Оптимизатор запросов СУБД MySQL автоматически использует оператор `TRUNCATE TABLE`, если оператор `DELETE` не содержит `WHERE`-условия или конструкции `LIMIT`.

**Листинг 19.45. Удаление всех записей таблицы `products`**

```
TRUNCATE TABLE products;
```

# 19.14. 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 19.46).

**Листинг 19.46. 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 19.47, где в таблицу `catalogs` добавляется пять новых записей.

***Замечание***

Многотабличный синтаксис для операторов `REPLACE` и `INSERT` не предусмотрен.

**Листинг 19.47. 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              |
+------------+------------------------+
```

# 19.15. Выборка данных

Рассмотрим запросы на выборку данных на примере таблицы `catalogs` (листинг 19.48). Таблица имеет два поля: первичный ключ `id_catalog` и название элемента каталога `name`.

**Листинг 19.48. 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 19.49.

**Листинг 19.49. 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 19.50).

**Листинг 19.50. 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 столбов в результирующей таблице или выбрать только часть столбцов (листинг 19.51).

**Листинг 19.51. 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 |
+--------------------+------------+
```

# 19.16. 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 19.52 приводится пример запроса, извлекающего из таблицы `catalogs` записи, чей первичный ключ `id_catalog` больше (оператор >) 2.

**Листинг 19.52. 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. Если логическое выражение возвращает TRUE для текущей записи, запись попадает в результирующую таблицу.

Помимо оператора > имеется еще несколько логических операторов, представленных в табл. 19.7.

***Таблица 19.7.*** *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 считается 0, а за истину любое число, отличное от нуля. В этом легко убедиться, если вывести логическое выражение в результирующую таблицу (листинг 19.53).

**Листинг 19.53. 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 19.54 используется составное условие: первичный ключ должен быть больше двух 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 19.54. 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. 19.8 и 19.9 представлены правила, по которым операторы AND и OR объединяют свои операнды.

***Таблица 19.8.*** *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. 19.8, оператор AND возвращает TRUE только в том случае, если оба его операнда принимают истинное значение (TRUE).

***Таблица 19.9.*** *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. 19.9, 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 19.55 демонстрируется использование оператора `NOT`.

**Листинг 19.55. 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. 19.10.

***Таблица 19.10.*** *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 19.56).

**Листинг 19.56. 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 19.56, в результирующую таблицу возвращаются записи в диапазоне от 3 до 4.

Существует конструкция, противоположенная конструкции `BETWEEN` — `NOT BETWEEN`, которая возвращает записи, не попадающие в интервал между *`min`* и *`max`* (листинг 19.57).

**Листинг 19.57. 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 19.58. Для этого предназначена конструкция `IN`.

**Листинг 19.58. 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 19.59).

**Листинг 19.59. 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 19.60 из таблицы `catalogs` извлекается запись, соответствующая элементу каталога "Процессоры".

**Листинг 19.60. 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 при помощи 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. 19.11.

***Таблица 19.11.*** *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. 19.11, можно задать различные шаблоны соответствия. В листинге 19.61 приводится пример выборки записей, которые содержат названия элементов каталога, заканчивающиеся на символ `'ы'`.

**Листинг 19.61. 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 19.62, 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 19.62. Использование оператора `NOT LIKE`**

```
SELECT * FROM catalogs WHERE name NOT LIKE '%ы';
+------------+--------------------+
| id_catalog | name               |
+------------+--------------------+
|          4 | Жесткие диски      |
|          5 | Оперативная память |
+------------+--------------------+
```

# 19.17. 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 19.63 результату функции DATE_FORMAT() присваивается новый псевдоним printdate.

**Листинг 19.63. Использование оператора AS**

```
SELECT id_catalog, name, DATE_FORMAT(putdate, '%d.%m.%Y') AS printdate
FROM catalogs;
+------------+--------------------+------------+
| id_catalog | name               | printdate  |
+------------+--------------------+------------+
|          3 | Видеоадаптеры      | 10.01.2008 |
|          4 | Жесткие диски      | 05.01.2008 |
|          2 | Материнские платы  | 28.12.2007 |
|          5 | Оперативная память | 20.12.2007 |
|          1 | Процессоры         | 10.01.2008 |
+------------+--------------------+------------+
```

# 19.18. 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 19.64, первый запрос сортирует результат выборки по полю id_catalog, а второй — по полю name.

**Листинг 19.64. 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 19.65).

**Листинг 19.65. 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 19.66 приводится дамп, позволяющий развернуть таблицу `tbl`.

**Листинг 19.66. Таблица tbl**

```
CREATE TABLE tbl (
  id_catalog int(11) NOT NULL,
  putdate datetime NOT NULL
);
INSERT INTO tbl VALUES (5, '2008-01-04 05:01:58');
INSERT INTO tbl VALUES (3, '2008-01-03 12:10:45');
INSERT INTO tbl VALUES (4, '2008-01-10 16:10:25');
INSERT INTO tbl VALUES (1, '2007-12-20 08:34:09');
INSERT INTO tbl VALUES (2, '2008-01-06 20:57:42');
INSERT INTO tbl VALUES (2, '2007-12-24 18:42:41');
INSERT INTO tbl VALUES (5, '2007-12-25 09:35:01');
INSERT INTO tbl VALUES (1, '2007-12-23 15:14:26');
INSERT INTO tbl VALUES (4, '2007-12-26 21:32:00');
INSERT INTO tbl VALUES (3, '2007-12-25 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 19.67.

**Листинг 19.67. Сортировка таблицы по двум столбцам**

```
SELECT * FROM tbl ORDER BY id_catalog, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          1 | 2007-12-23 15:14:26 |
|          1 | 2007-12-20 08:34:09 |
|          2 | 2008-01-06 20:57:42 |
|          2 | 2007-12-24 18:42:41 |
|          3 | 2008-01-03 12:10:45 |
|          3 | 2007-12-25 12:11:10 |
|          4 | 2008-01-10 16:10:25 |
|          4 | 2007-12-26 21:32:00 |
|          5 | 2008-01-04 05:01:58 |
|          5 | 2007-12-25 09:35:01 |
+------------+---------------------+
```

Как видно из листинга 19.67, записи в таблице `tbl` сначала сортируются по столбцу `id_catalog`, а совпадающие в рамках одного значения `id_catalog` записи сортиру-

ются 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 19.68).

**Листинг 19.68. Обратная сортировка по двум столбцам**

```
SELECT * FROM tbl ORDER BY id_catalog DESC, putdate DESC;
+------------+---------------------+
| id_catalog | putdate             |
+------------+---------------------+
|          5 | 2008-01-04 05:01:58 |
|          5 | 2007-12-25 09:35:01 |
|          4 | 2008-01-10 16:10:25 |
|          4 | 2007-12-26 21:32:00 |
|          3 | 2008-01-03 12:10:45 |
|          3 | 2007-12-25 12:11:10 |
|          2 | 2008-01-06 20:57:42 |
|          2 | 2007-12-24 18:42:41 |
|          1 | 2007-12-23 15:14:26 |
|          1 | 2007-12-20 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.

## 19.19. Вывод записей в случайном порядке

Для вывода записей в случайном порядке используется конструкция `ORDER BY RAND()`. В листинге 19.69 демонстрируется вывод содержимого таблицы `catalogs` в случайном порядке.

**Листинг 19.69. 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 19.70).

**Листинг 19.70. Вывод одной случайной записи**

```
SELECT id_catalog, name FROM catalogs ORDER BY RAND() LIMIT 1;
+------------+-------------------+
| id_catalog | name              |
+------------+-------------------+
|           2 | Материнские платы |
+------------+-------------------+
```

# 19.20. 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 19.71 извлекаются первые две записи таблицы `catalogs`, при этом одновременно осуществляется их обратная сортировка по полю `id_catalog`.

**Листинг 19.71. 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 19.72).

**Листинг 19.72. 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`.

# 19.21. Вывод уникальных значений

Очень часто встает задача выбора из таблицы уникальных значений. Воспользуемся таблицей `tbl` из листинга 19.68. Пусть требуется вывести все значения поля `id_catalog`; для этого можно воспользоваться запросом, представленным в листинге 19.73.

**Листинг 19.73. 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 19.73, 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 19.74).

> ***Замечание***
>
> Ключевое слово `DISTINCT` имеет синоним — `DISTINCTROW`.

**Листинг 19.74. Выборка уникальных значений**

```
SELECT DISTINCT id_catalog FROM tbl ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|           1 |
|           2 |
|           3 |
|           4 |
|           5 |
+------------+
```

Как показано в листинге 19.74, результат запроса не содержит ни одного повторяющегося значения.

Для ключевого слова `DISTINCT` имеется противоположенное 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 19.75).

***Замечание***

Конструкция `GROUP BY` располагается в `SELECT`-запросе перед конструкциями `ORDER BY` и `LIMIT`.

**Листинг 19.75. Использование конструкции `GROUP BY`**

```
SELECT id_catalog FROM tbl
GROUP BY id_catalog ORDER BY id_catalog;
+------------+
| id_catalog |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
```

# 19.22. 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 с использованием оператора `UNION`. Пусть имеются два `SELECT`-запроса, представленные в листинге 19.76.

**Листинг 19.76. 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 19.77.

**Листинг 19.77. 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 19.78).

**Листинг 19.78. 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 |
+------------+
```

## Глава 20

# 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 программирование PHP получила очень широкое распространение среди Web-разработчиков.

## 20.1. Функция *mysql_connect()*

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 20.1 приведен пример использования функции `mysql_connect()`.

**Листинг 20.1. 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 20.1 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 20.2. Иначе русский текст будет попадать в базу данных в виде знаков вопроса.

> ***Замечание***
>
> Функция `mysql_query()` рассматривается более подробно в *разд. 20.4*.

**Листинг 20.2. Установка кодировки**

```
<?php
  mysql_query ("SET NAMES 'cp1251'");
?>
```

# 20.2. Функция *mysql_close()*

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 20.3 приведен пример использования функции.

**Листинг 20.3. 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");
  }
?>
```

## 20.3. Функция *mysql_select_db()*

Использование функции `mysql_select_db()` эквивалентно вызову команды `USE` в SQL-запросе, т. е. функция `mysql_select_db()` выбирает базу данных для дальнейшей работы, и все последующие SQL-запросы применяются к выбранной базе данных.

`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 20.4 приводится пример использования данной функции.

**Листинг 20.4. 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 20.5), и подключать данный файл при помощи конструкции `include_once()` или `require_once()`, если требуется обращение к базе данных MySQL.

**Листинг 20.5. 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'");
?>
```

# 20.4. Функция *mysql_query()*

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 20.6 представлен скрипт, создающий новую таблицу `catalogs`.

**Листинг 20.6. 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 20.6 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 20.7.

**Листинг 20.7. 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');
```

## 20.5. Функция *mysql_result()*

Первая функция `mysql_result()` возвращает результат запроса, выполненного функцией `mysql_query()`. С ее помощью можно получить доступ к отдельному полю записи. Синтаксис функции представлен ниже:

```
$mysql_result($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 20.8 при помощи данной функции выводится название первого каталога в таблице `catalogs` учебной базы данных `shop`.

**Листинг 20.8. 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 20.8 представлен на рис. 20.1.

**Рис. 20.1.** Результат работы скрипта из листинга 20.8

# 20.6. Функция *mysql_fetch_row()*

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:

```
array mysql_fetch_row($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. При работе с массивом, который возвращает функция `mysql_fetch_row()`, удобно использовать конструкцию `list()`, преобразующую элементы массива в переменные. В листинг 20.9 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

**Листинг 20.9. 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 20.9 представлен на рис. 20.2.

**Рис. 20.2.** Результат работы скрипта из листинга 20.9

# 20.7. Функция *mysql_fetch_assoc()*

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 20.10 выводится HTML-таблица, в которой размещаются данные из таблицы `catalogs`.

**Листинг 20.10. 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 20.10 выглядит точно так же, как результат работы скрипта из листинга 20.9 (см. рис. 20.2). 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 20.11 при помощи встроенной функции MySQL `VERSION()` извлекается версия сервера MySQL. При этом в качестве ключа массива выступает строка `'VERSION()'`.

**Листинг 20.11. 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 20.11 можно переписать так, как показано в листинге 20.12.

**Листинг 20.12. 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'];
?>
```

# 20.8. Функция *mysql_fetch_array()*

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 20.13 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 20.13. 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 20.14).

**Листинг 20.14. 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>";
?>
```

# 20.9. Функция *mysql_fetch_object()*

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 следующий 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 20.15 приведен пример, в котором с помощью этой функции из таблицы `catalogs` выводятся первичный ключ таблицы `id_catalog` и имя элемента каталога `name`.

**Листинг 20.15. 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 20.15 представлен на рис. 20.3.

**Рис. 20.3.** Результат работы скрипта из листинга 20.15

# 20.10. Функция *mysql_num_rows()*

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:

```
int 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 20.16).

**Листинг 20.16. 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 20.17).

**Листинг 20.17. 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);
?>
```

В результате работы скрипта из листинга 20.17 будет возникать ошибка (рис. 20.4).

**Рис. 20.4.** Попытка извлечь несуществующую запись

Для предотвращения возникновения такой ошибки удобно воспользоваться функцией `mysql_num_rows()` (листинг 20.18).

**Листинг 20.18. 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 20.18 можно переписать следующим образом (листинг 20.19).

**Листинг 20.19. 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 20.18) используются два (см. листинг 20.19). 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.

# Заключение

Несколько лет назад вышла наша первая книга "Самоучитель PHP 5", которая неоднократно допечатывалась. Второе издание этой книги было полностью переработано и увидело свет в июне 2006 года. Книга, последнюю главу которой вы только что прочитали, по своей сути является третьим изданием "Самоучителя PHP". Но в реальности от первых двух книг в ней сохранился только стиль изложения "от простого к сложному", т. к. весь материал книги был полностью переработан с учетом нововведений, анонсированных разработчиками PHP для шестой версии.

Однако, как уже отмечалось во *введении*, невозможно в одной книге рассмотреть все аспекты программирования на PHP. Поэтому нами написано немало книг, дополняющих эту книгу.

Так, в книге "PHP. Практика создания Web-сайтов"[1] рассматривается создание "с нуля" полноценного Web-сайта, отвечающего всем современным требованиям. Книга "PHP 5 на примерах"[2] — это по сути "сборник рецептов" PHP-разработчика: в ней рассмотрены короткие и эффективные приемы, наиболее часто применяющиеся при создании сайтов.

Ни один современный сайт не обходится без использования СУБД (в РФ, как правило, MySQL), и три наших книги "Самоучитель MySQL 5"[3], "MySQL 5 на примерах"[4] и "MySQL 5"[5] (серия "В подлиннике") посвящены этой популярной СУБД.

Книга "Головоломки на PHP для хакера"[6] — это задачник по PHP с подробными решениями и объяснениями и с уклоном в безопасность создаваемых приложений. Цель этой книги — показать на конкретных примерах методы защиты Web-приложений от различных атак.

Начиная с пятой версии PHP, в языке появилась полноценная объектно-ориентированной модель. В книге "Объектно-ориентированное программирование на PHP"[7] мы даем ответы на вопросы: зачем нужно объектно-ориентированное

---

[1] Кузнецов М., Симдянов И. PHP. Практика создания Web-сайтов. — СПб.: БХВ-Петербург, 2008.
[2] Кузнецов М., Симдянов И. PHP 5 на примерах. — СПб.: БХВ-Петербург, 2005.
[3] Кузнецов М., Симдянов И. Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2006.
[4] Кузнецов М., Симдянов И. MySQL на примерах. — СПб.: БХВ-Петербург, 2007.
[5] Кузнецов М., Симдянов И. MySQL 5. — СПб.: БХВ-Петербург, 2005.
[6] Кузнецов М., Симдянов И. Головоломки на PHP для хакера / 2-е изд. — СПб.: БХВ-Петербург, 2008.
[7] Кузнецов М., Симдянов И. Объектно-ориентированное программирование на PHP. — СПб.: БХВ-Петербург, 2007.

программирование в PHP, когда его следует применять, а когда его применение не целесообразно и даже вредно. В этой книге на примере построения большого Web-приложения (CMS) демонстрируется, как добиться повторного использования кода в реальных проектах.

Процесс программирования сам по себе очень увлекателен и многие начинающие программисты хотели бы связать свою профессиональную жизнь именно с программированием. Однако если создаваемый код не выдерживает конкуренции или его не удается успешно продать, рано или поздно приходится уходить из любимой области. Тому, как успешно работать в сфере программирования, посвящена наша книга "Программирование: ступени успешной карьеры"[8].

# Online-поддержка

С момента выхода самой первой нашей книги мы взяли себе за правило оказывать читателям наших книг online-поддержку на форумах нашей студии SoftTime. Дело в том, что современные языки программирования обрастают объемным окружением, превращаясь не просто в языки общения с компьютером, а в целые технологии, которые просто невозможно описать в одной книге, в результате чего за бортом остаются целые, подчас немаловажные, разделы. И эту недостающую информацию всегда можно найти на наших сайтах, о которых рассказано далее.

В настоящий момент у IT-студии SoftTime, руководителями которой являются авторы книги, есть несколько сайтов, на которых вы можете найти различные дополнительные материалы:

- **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.

Кратко расскажем о каждом из наших интернет-проектов.

## Портал по программированию *SoftTime.ru*

Уже около пяти лет назад на сайте нашей студии был создан форум, посвященный программированию на PHP (**http://www.softtime.ru/forum/**).

[8] Кузнецов М., Симдянов И. Программирование: ступени успешной карьеры. — СПб.: БХВ-Петербург, 2006.

Как оказалось, форума только по PHP мало, поэтому постепенно вводились другие форумы. Сейчас на портале **www.softtime.ru** существует несколько форумов, посвященных тем или иным аспектам Web-программирования:

- **Форум PHP**, на котором обсуждаются вопросы программирования на PHP, а также установка, настройка и модификация Web-приложений, распространяемых IT-студией SoftTime;
- **Форум Apache**, посвященный установке и конфигурированию Web-сервера Apache;
- **Форум по регулярным выражениям**, на котором обсуждаются как вопросы работы с регулярными выражениями вообще, так и особенности их применения в PHP, MySQL и JavaScript в частности;
- **Форум MySQL**, предназначенный для обсуждения вопросов взаимодействия MySQL и PHP, а также различных аспектов администрирования СУБД и рассмотрения хитрых SQL-запросов;
- **Форум HTML+CSS+JavaScript**, на котором разговариваем о HTML-верстке, использовании стилевых таблиц CSS и программировании на языке JavaScript. На этом форуме также обсуждаются вопросы дизайна и компьютерной графики;
- **Форум "Задачи на PHP".** На этом форуме вы можете попрактиковаться в решении задач на PHP, которые составляют как авторы, так и другие форумчане. За первое место в решении особо интересных конкурсных задач читателей ждет приз: любая книга авторов с автографами.

Таким образом, если по мере чтения у вас возникнут технические вопросы, к примеру, такие как "Что означает та или иная ошибка?", "Как установить PHP?" и т. д., обсудить их можно на форумах портала **SoftTime.ru** по адресу **http://www.softtime.ru/forum/**.

Кроме того, на портале **www.softtime.ru**, по многочисленным просьбам форумчан и читателей нашей книги-задачника по C++ [9], был создан форум, посвященный вопросам программирования на языке C++.

На сайте **www.softtime.ru** вы также найдете большое количество самых разнообразных скриптов и статей по Web-программированию, написанных как нами, так и нашими читателями.

Как уже отмечалось выше, осветить все функции и расширения PHP, а также смежные дисциплины, необходимые для создания Web-приложений, в одной книге

[9] Кузнецов М., Симдянов И. C++. Мастер-класс в задачах и примерах. — СПб.: БХВ-Петербург, 2007. — 480 с.

невозможно — объем их чрезвычайно велик. Поэтому мы регулярно выпускаем книги по Web-программированию, затрагивающие самые разнообразные вопросы, начиная с проблем безопасности и баз данных и заканчивая психологическими и юридическими аспектами профессии программиста. С полным списком наших книг можно ознакомиться по ссылке **http://www.softtime.ru/php5/index.php**.

## Портал *Softtime.org*

Если на **SoftTime.ru** освещаются преимущественно различные вопросы программирования, то у портала **www.Softtime.org** совершенно иная задача. Со временем стало ясно, что в рамках только программистского портала и нам, и нашим читателям становилось уже тесно. Нам — потому что в студии за время ее существования появилось достаточно много проектов, не связанных непосредственно с программированием, размещать же всю информацию на одном сайте означало бы "свалить все в одну кучу", а этого, конечно, делать не нужно. У наших посетителей тоже становилось все больше и больше вопросов, к программированию как таковому не относящихся. В основном это вопросы правового и психологического характера. Что понятно: наша профессиональная жизнь не ограничивается рамками конкретной профессии, какой бы она не была, и почти всем нам по долгу службы приходится сталкиваться с различными юридическими и психологическими вопросами. Особенно тем, кто уже стал руководителем или занимается собственным бизнесом (а таковых не мало среди наших читателей).

В связи с вышеизложенными причинами и было принято решение создать портал **www.Softtime.org**, на котором и авторы и форумчане в меру своей компетенции делятся друг с другом различными советами по преодолению тех или иных проблем юридического и психологического характера. Поэтому вопросы "Как превратить увлечение программированием в профессию?", "Как зарабатывать при помощи PHP?", "Как организовывать свое дело?", "Нужно ли официально регистрировать Web-студию?", "Как юридически правильно оформить домен?" и многие другие, обсуждаются на форуме **http://www.softtime.org/forum/**.

## Сайт *Softtime.biz*

На сайте **www.Softtime.biz** расположена информация о коммерческих услугах, оказываемых IT-студией SoftTime. Если вы первый раз покупаете нашу книгу, то имейте в виду, что авторы — действующие программисты, зарабатывающие себе на жизнь разработкой Web-порталов самой различной направленности и уровня сложности. Потому и в этой, и во всех других наших книгах нет мифических и "высосанных из пальца" примеров — все примеры реальные и взяты из нашей каждодневной практики Web-программирования. Важность этого аспекта сложно

переоценить: думаем, что советы от авторов, которые сами делают по нескольку сайтов ежемесячно, более ценны, чем рекомендации человека, пусть и прекрасно знающего язык программирования, но не имеющего каждодневной реальной практики применения своих знаний (последний просто не знает, какие требования ставят перед разработчиком современные реалии). У нас такая практика, причем многолетняя и успешная, есть, в чем вы можете убедиться, зайдя на сайт **www.softtime.biz**. И нашим опытом мы с удовольствием делимся с вами в этой книге.

И мы искренне надеемся, что после того, как вы прочитали книгу, мы с вами не расстанемся, а встретимся на наших форумах.

Всего вам доброго!

# 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 дист-

рибутивов 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 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 **Dowload 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 CR (candidate release);
- 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 (x86) — урезанная версия дистрибутива, из которой удалены все вспомогательные утилиты, по сути это "голый" сервер MySQL;
- Windows (x86) — полная версия дистрибутива MySQL, включающая автоматический установщик;
- Without installer (unzip in C:\) — полная версия MySQL без автоматического установщика.

Рекомендуется выбрать дистрибутив Windows (x86). 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` (рис. П1.1). 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 предлагает 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 треере.

### 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. Он 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 обнару-

жить 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.

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. П1.6.** Службы Windows

**Рис. П1.7.** Окно свойств 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 по 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, представление 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, если 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
```

# П1.10. Общая настройка конфигурационного файла php.ini

После того как конфигурационный файл httpd.conf настроен, необходимо откорректировать несколько директив конфигурационного файла php.ini. В первую очередь следует выставить директиве `error_reporting` значение `E_ALL & ~E_NOTICE`:

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

Данная директива отвечает за уровень тревожности интерпретатора PHP при отображении ошибок. Слишком высокий уровень тревожности будет приводить к выводу замечаний (notice), которые являются своеобразными советами. Управлять уровнем тревожности интерпретатора PHP можно не только глобально, но и для каждого скрипта в отдельности: для этого в его начало следует поместить вызов функции `error_reporting()`, передав ей в качестве аргумента уровень тревожности (листинг П1.8).

**Листинг П1.8. Использование функции `error_reporting()`**

```
<?php
  error_reporting(E_ALL & ~E_NOTICE);
?>
```

Далее необходимо включить директиву `display_errors`, которая, начиная с PHP 5, отключена:

```
display_errors = On
```

Данная директива разрешает (`On`) или запрещает (`Off`) вывод ошибок и предупреждений в окно браузера. Если отключить данную директиву, то при возникновении любой ошибки будет выводиться пустое окно браузера. Для того чтобы все-таки выяснить, какая ошибка возникла, в конфигурационном файле php.ini необходимо включить директиву `log_errors`:

```
log_errors = On
```

А также снять комментарий напротив директивы:

```
error_log = syslog
```

Директива `error_log = syslog` позволяет регистрировать ошибки в системном журнале Windows, доступном по пути **Пуск | Настройка | Панель управления | Администрирование | Просмотр событий**. В Linux сообщения об ошибках будут помещаться в журнальный файл демона syslog.

Директива `short_open_tag` определяет, разрешено (`On`) или запрещено (`Off`) использование коротких тегов `<?` вместо `<?php`:

```
short_open_tag = On
```

Если данная директива отключена (принимает значение `Off`), то использование коротких тегов `<?` будет приводить к ошибке. Следует иметь в виду, что, начиная с PHP 6, данная директива будет исключена, и использование коротких PHP-тегов будет недопустимо.

Директива `output_buffering` позволяет помещать весь вывод из окна браузера в буфер и отправлять его только после того, как скрипт заканчивает работу. Для того чтобы включить такое поведение, следует присвоить директиве значение `On`:

```
output_buffering = On
```

Директива `output_buffering` позволяет эффективно бороться с ошибкой "Cannot modify header information — headers already sent by", которая возникает, если функции `session_start()`, `setcookie()` или `header()` вызываются после того, как в окно браузера уже отправлена информация.

Однако при разработке сайтов, которые будут размещаться на сайте хостпровайдера, лучше не менять значение данной директивы, т. к. она может быть отключена. В большинстве случаев ошибку "Cannot modify header information — headers already sent by" можно обойти либо программно, либо применяя функции управления выводом, которые также передают вывод в предварительный буфер.

Директива `max_execution_time` определяет максимальное количество секунд, которые отводятся на выполнение скрипта:

```
max_execution_time = 30
```

При этом не учитывается время, затрачиваемое на ожидание ответа от базы данных или из сети. По умолчанию директива принимает значение, равное 30 секундам.

Если директива принимает значение, равное 0, то считается, что все ограничения по времени сняты. Управлять значением директивы `max_execution_time` можно не только глобально, редактируя конфигурационный файл php.ini, но и локально, поместив в начало PHP-скрипта вызов функции `set_time_limit()` и передав в качестве аргумента функции количество секунд, отводимых скрипту на исполнение (листинг П1.9).

**Листинг П1.9. Снятие временных ограничений при помощи функции `set_time_limit()`**

```
<?php
  set_time_limit(0);
?>
```

Директива `memory_limit` определяет максимальный объем памяти, выделяемый одному сценарию:

```
memory_limit = 8M
```

По умолчанию значение директивы равно 8 Мбайт. Это значение следует увеличить, если планируется обрабатывать файлы объемом около 10 Мбайт, т. к. в этом случае не удастся даже прочитать содержимое файла в переменную или массив, например, при помощи функций `file_get_contents()` или `file()`, и 10-мегабайтный файл придется обрабатывать либо по частям, либо построчно, что замедлит работу скрипта.

Директива `post_max_size` определяет максимальный размер данных, переданных методом `POST`:

```
post_max_size = 8M
```

По умолчанию директива принимает данные объемом 8 Мбайт. Если планируется загружать на сервер файлы с большим размером, то значение данной директивы следует увеличить.

Директива `file_uploads` разрешает (`On`) или запрещает (`Off`) загрузку файлов на сервер:

```
file_uploads = On
```

Директива `upload_max_filesize` задает максимальный размер загружаемых на сервер файлов:

```
upload_max_filesize = 2M
```

По умолчанию директива принимает значение, равное 2 Мбайт. Если планируется загружать на сервер более объемные файлы, следует увеличить значение данной директивы.

Директива `allow_url_fopen` разрешает (`On`) или запрещает (`Off`) использование загрузки файлов с удаленных хостов, т. е. использование адресов http:// и ftp:// в файловых функциях:

```
allow_url_fopen = On
```

Директива `session.save_path` позволяет задать путь к каталогу, в который будут сохраняться файлы с сессионными данными. Использование данной директивы удобно для отладки сессий, т. к. позволяет визуально контролировать создание новой сессии:

```
session.save_path = C:/tmp
```

Директива `session.cookie_lifetime` определяет время жизни cookie в секундах. Если директива принимает значение 0, то срок жизни cookie продолжается до закрытия браузера пользователем:

```
session.cookie_lifetime = 0
```

# П1.11. Настройка и проверка работоспособности расширений PHP

По умолчанию, начиная с PHP 5, все 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 имена 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. П1.9).

**Рис. П1.9.** PHP не может загрузить расширение php_mysql.dll

## 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. Данное приложение посвящено 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 Брандмауэр Windows (Windows 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 брандмауэре 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 (x86).

**Рис. П2.1.** Стартовое окно автоматического установщика

**Рис. П2.2.** Выбор режима установки MySQL

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.

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 будут 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\.

**Рис. П2.3.** Выбор компонентов для установки

**Рис. П2.4.** Выбор каталога,
в который будет установлен MySQL-сервер

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).

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 прединсталляционной настройки выводится окно, представленное на рис. П2.5. Нажатие кнопки **Install** начинает процесс копирования файлов, представленный на рис. П2.6.

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.

**Рис. П2.5.** Завершающее окно перед процессом инсталляции

**Рис. П2.6.** Копирование файлов

**Рис. П2.7.** Предложение зарегистрироваться на сайте **http://www.mysql.com**

**Рис. П2.8.** Завершение 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.

## 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, изображенного на рис. П2.9. После нажатия кнопки **Next** открывается окно, представленное на 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

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. П2.10.** Выбор режима настройки

**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.

**Рис. П2.12.** Выбор типа таблиц по умолчанию

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. П2.13.** Выбор пути для хранения файла под таблицы InnoDB

**Рис. П2.14.** Выбор максимального количества активных соединений с сервером

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.

На реальных серверах не следует устанавливать это число более 200, обычно если оперативной памяти выделено достаточно и производительность процессора (про-

цессоров) 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 нет других версий MySQL, рекомендуется сохранить значение по умолчанию, т. к. порт 3306 является стандартным для MySQL.

***Замечание***

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.

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. Флажок **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 придет-

ся выполнять вручную. Как будет показано далее, можно изменить режим работы сервиса в любой момент в консоли управления сервисами.

**Рис. П2.15.** Выбор порта, по которому сервер MySQL будет слушать клиентские запросы

**Рис. П2.16.** Выбор кодировки на MySQL-сервере по умолчанию

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 производите 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 пользова-

теля. 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).

### *Замечание*

Для выхода введите команду `EXIT`.

**Рис. П2.20.** Командная строка утилиты MySQL Command Line Client

**Рис. П2.21.** Результат выполнения команды `SELECT VERSION()`

### *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.

**Рис. П2.22.** Консоль управления сервисами

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`.

# П2.2. Установка MySQL под Linux

Большинство дистрибутивов Linux содержат 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 стандартной установки выполните команды, предварительно войдя в систему под учетной записью `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 имен-

но 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` |

***Таблица П2.1*** *(окончание)*

| Имя секции | Описание |
|---|---|
| [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 |
| [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.

# Предметный указатель

## $

$this 448

## C

Cookies 425

## H

HTTP-заголовкок 411—412

## S

Structured Query Language (SQL) 514

## А

Абстрактные типы данных 441
Аксессор 461

## Б

База данных:
  information_schema 520
  mysql 520
  реляционная 515
  система управления 514
  создание 519
  удаление 520

## В

Виртуальный хост 602
Выражение 18
Выражение составное 19

## Д

Деструктор 459
Директива 638
Директива httpd.conf:
  AddDefaultCharset 607
  CustomLog 603
  DirectoryIndex 611

DocumentRoot 603
ErrorLog 603
NameVirtualHost 603—604
PHPIniDir 610
ServerAdmin 603
ServerName 603
Директива my.ini, default-character-set 633
Директива php.ini:
allow_url_fopen 616
asp_tags 16
display_errors 613
error_reporting 33, 613
extension 616
extension_dir 617
file_uploads 615
log_errors 614
magic_quotes_gpc 248
max_execution_time 614
memory_limit 28, 615
output_buffering 614
post_max_size 615
precision 27
session.cookie_lifetime 616
session.save_path 431, 616
short_open_tag 16, 614
upload_max_filesize 220, 615

# З

Замечание (Notice) 33
Запрос на выборку данных 549

# И

Индексы:
PRIMARY KEY 548
UNIQUE 548
ключ:
внешний 519
логический 517
первичный 517, 539
суррогатный 517—518
Инкапсуляция 442, 445
Интерполяция 450
Интерфейс 483
наследование 486
Исключение 509

# К

Каталог:
данных 519
определение 363
Класс 443
final 496
базовый 471
производный 471
Клонирование объекта 497
Ключевое слово:
clone 497
final 494
Кодировка 521
Коды состояния 415
Комментарий 21
# 21
/* ... */ 21
// 21
в конфигурационном файле 638
Константа:
__CLASS__ 60, 493
__FILE__ 60, 493
__FUNCTION__ 60, 493
__LINE__ 60, 493
__METHOD__ 60, 493
FALSE 35, 55
M_1_PI 338
M_2_PI 338

M_2_SQRTPI 338
M_E 337
M_EULER 338
M_LN10 337
M_LN2 337
M_LNPI 338
M_LOG10E 337
M_LOG2E 337
M_PI 337
M_PI_2 337
M_PI_4 337
M_SQRT1_2 338
M_SQRT2 338
M_SQRT3 338
M_SQRTPI 338
NULL 55
RAND_MAX 341
TRUE 35, 55
динамическая 58
класса 491
объявление 53
предопределенная 60, 493
проверка существования 59
Конструктор 454
Конструкция:
array() 117, 121, 176
catch 509
declare 188
echo 64, 250
empty() 38
global 174
include 107
include_once 107
isset() 37, 140, 463
list() 131, 177
require 107
require_once 107
static 176
throw 509
try 509
unset() 36, 435, 460, 465
Конструкция SQL:
ALL 564, 566
AS 558
AUTO_INCREMENT 544
BETWEEN 555
DEFAULT 538
DEFAULT CHARACTER SET 521
DISTINCT 564
DISTINCTROW 564
GROUP BY 564
IGNORE 543
IN 555
INTERVAL 541
LIMIT 546, 562—563
NOT BETWEEN 555
NOT IN 556
ORDER BY 558
ORDER BY ... ASC 561
ORDER BY ... DESC 559
SET 547
VALUES 544, 548
WHERE 546—547, 551, 556
Контролируемый блок 510

# M

Массив:
$_SERVER 159
$GLOBALS 175
ассоциативный 124
индексный 124
многомерный 129
создание 117
Метод:
__autoload() 452, 460
__call() 452, 466
__clone() 452, 499
__construct() 452, 455
__destruct() 452, 459
__get() 452, 461
__isset() 452, 463
__set() 452, 461

__set_state() 452
__sleep() 452, 500
__toString() 452, 468
__unset() 452, 464
__wakeup() 452
абстрактный 481
динамический 466
класса 443
перегрузка 476
статический 490

# Н

Наследование 442, 471

# О

Объект 443
интерполяция 468
клонирование 497
Округление числа 346
Оператор:
! 82, 87
!= 76
!== 76
% 66
& 70
&& 82
* 66
. 63
.= 64
/ 66
@ 113
^ 70, 72
| 70, 71
|| 82
~ 70, 73
+ 66
++ 66, 68
< 76
<< 70, 73
<<< 32
<= 76
= 24
== 76, 78
=== 76, 78
=> 118
> 76
-> 446
>= 76
>> 70, 74
and 82
break 90, 97
class 443
continue 94
do … while 101
extends 471
for 102
foreach 138
if 79
implements 484
interface 483
new 455
or 82
return 109, 176
switch 90
while 95
x ? y : z 89
разрешения области видимости :: 487, 490, 492
Оператор SQL:
> 551
ALTER TABLE 533
AND 553
CREATE DATABASE 519
CREATE TABLE 529
DELETE 545, 546, 583
DESCRIBE 533
DROP DATABASE 520
INSERT 543, 548, 583
многострочный 544
однострочный 536

NOT LIKE 557
OR 553
REPLACE 548
SELECT 551, 565—566, 583
SET NAMES 571
SHOW DATABASES 519
SHOW TABLES 530
TRUNCATE TABLE 546
UNION 565, 566
UNION ALL 566
UPDATE 547, 583
Определитель точности 254

# П

Параметр 637
mysqld:
--defaults-file 637
--standalone 634
Переключатель 215
Переменная:
глобальная 174
динамическая 51
инициализация 23
интерполяция 29
неинициализированная 33
определение 23
проверка существования 36
статическая 175
уничтожение 36
Подавление кэширования 419
Поле пароля 208
Полиморфизм 442, 478
Порт 628
Последовательность * 550
Почтовое сообщение, отправка 437
Права доступа UNIX 399

# Р

Рассылка писем 439
Регулярные выражения:
квантификаторы 285
метасимволы 284
модификаторы 287
позиционные проверки 287
специальные символы 286
функции 288
Реляционная база данных 518

# С

Сервис 628, 633
Сессия 431
Скрипт 13
Скрытое поле 210
Спецификатор доступа:
private 445, 473
protected 475
public 445, 473
var 445
Список 213
Стандарт языка SQL 515
Столбец:
псевдоним 558
уникальные значения 564

# Т

Таблица:
InnoDB 626
MyISAM 626
объединение 565
результирующая 551
Тег:
<? … ?> 16
<?= … ?> 16
<?php … ?> 16
Текстовая область 209

Текстовое поле 207
Тип данных:
array 25, 53, 121
BIGINT 524
BIT 524
BLOB 527
BOOL 524
boolean 25, 35, 53, 524
CHAR 527
DATE 526, 540
DATETIME 526
DEC 524
DECIMAL 524
double 25, 27, 524
DOUBLE PRECISION 524
ENUM 527
float 25, 27, 53, 524
INT 524
int64 25
integer 25, 53, 524
LONGBLOB 527
LONGTEXT 527
MEDIUMBLOB 527
MEDIUMINT 524
MEDIUMTEXT 527
NULL 25
NUMERIC 524
object 25, 53
REAL 524
resource 25, 53
SET 527
SMALLINT 524
string 25, 53
TEXT 527
TIME 526
TIMESTAMP 526
TINYBLOB 527
TINYINT 524
TINYTEXT 527
VARCHAR 527
YEAR 526
неявное приведение 44
строковый 539
явное приведение 46

# У

Удаление изображений 300
Утилита:
apachectl 609
ApachtMonitor 600
MySQL Command Line Client 631
MySQL Server Instance Configuration Wizard 626
mysqldump 638

# Ф

Файл:
db.opt 519
hosts 605
абсолютный путь 364
бинарный режим 384
блокировка 390
временный 369
дозапись 389
загрузка на сервер 217
закрытие 387
копирование 370
определение 363
относительный путь 366
переименование 371
режим трансляции 384
сетевой путь 367
создание 363
удаление 370
чтение 376
Флажок 211
Функция:
abs() 346
acos() 353
acosh() 353
addcslashes() 247, 249
addslashes() 247—248

array_combine() 126
array_count_values() 144—145
array_fill() 121
array_key_exists() 142—143
array_multisort() 150, 156
array_rand() 147—148
array_search() 142—143
array_sum() 146
arsort() 150, 153
asin() 353
asinh() 353
asort() 149, 152
atan() 353
atan2() 354
atanh() 354
base_convert() 342—343
base64_decode() 257
base64_encode() 257—258
basename() 269—270
bin2hex() 265
bindec() 343—345
call_exists() 453
call_user_func() 181, 453
call_user_func_array() 180, 453
call_user_method() 453
call_user_method_array() 453
ceil() 346—347
chdir() 404—405
checkdate() 323, 326
chgrp() 400
chmod() 399—400
chop() 232
chown() 400
chr() 221—222
chroot() 404
chunk_split() 271, 274
closedir() 404
compact() 126—127
constant() 58
convert_cyr_string() 256—257
convert_uudecode() 257
convert_uuencode() 257
copy() 370
cos() 353
cosh() 353
count() 144
COUNT() 558
count_chars() 221, 223
create_function() 180
current() 134
date() 322, 323
date_default_timezone_get() 316—317
date_default_timezone_set() 316—317
date_parse() 323, 332
date_sun_info() 317, 321
date_sunrise() 316, 320
date_sunset() 317
decbin() 343—344
dechex() 343—344
decoct() 343—344
define() 53, 57
defined() 57
deg2rad() 343, 345
die() 193
dirname() 269, 271
doubleval() 49
each() 134, 135
empty() 193
end() 134
eval() 51, 193
exit() 193, 574
exp() 349, 350
explode() 272, 278
expm1() 349
extension_loaded() 183, 186
extract() 126, 128
fclose() 364, 373, 387
feof() 373
fflush() 375, 388
fgetc() 373
fgets() 373, 378
fgetss() 374
file() 374, 382, 615
file_get_contents() 374, 380, 615
file_put_contents() 375, 385
filegroup() 400

fileowner() 400
fileperms() 400—401
filesize() 379
floatval() 49
flock() 375, 390
floor() 346, 348
fmod() 346, 348
fopen() 364, 367, 373
fpassthru() 374
fprintf() 250
fputs() 375
fread() 374, 378
fscanf() 374
fseek() 395, 397
ftell() 395
ftruncate() 375
func_get_arg($arg_num) 172
func_get_args() 172
func_num_args() 172
function_exists() 183
fwrite() 375, 383
get_class() 453
get_class_methods() 453
get_class_vars() 453
get_declared_class() 453
get_declared_interfaces() 453
get_defined_constants() 58, 60
get_defined_functions() 183—185
get_defined_vars() 39
get_extension_funcs() 183, 185
get_headers() 418
get_html_translation_table() 239
get_included_files() 112
get_loaded_extensions() 183
get_magic_quotes_gpc() 248
get_object_vars() 453
get_parent_class() 453
get_resource_type() 39
getcwd() 404—405
getdate() 323, 329
getenv() 194
getrandmax() 341—342
gettimeofday() 310, 313
gettype() 39—40
glob() 404, 409
gmdate() 322
gmmktime() 310
gmstrftime() 323
header() 413, 614
headers_list() 413, 417
headers_sent() 413, 416
hebrev() 257
hebrevc() 257
hexdec() 343, 345
html_entity_decode() 239
htmlentities() 239, 246
htmlspecialchars() 238, 241
htmlspecialchars_decode() 239, 245
http_build_query() 200
hypot() 354
idate() 322
implode() 272, 278
in_array() 141, 142
interface_exists() 453
intval() 49—50, 67
is_a() 454
is_array() 39
is_bool() 39
is_callable() 39, 454
is_double() 39, 42, 355
is_executable() 400
is_finite() 356, 359
is_float() 39, 355, 357
is_infinite() 356, 359
is_int() 39, 41—42, 355—356
is_integer() 40, 355
is_long() 40, 355
is_nan() 356, 358
is_null() 40
is_numeric() 40, 43, 355, 357
is_object() 40, 454
is_readable() 400
is_real() 40, 355
is_resource() 40, 44
is_scalar() 40
is_string() 40

is_subclass_of() 454
is_uploaded_file() 372
is_writable() 400
is_writeable() 400
isset() 194
join() 272
key() 134
krsort() 150, 154
ksort() 150, 153
lcg_value() 341—342
lchgrp() 400
lchown() 400
levenshtein() 262
localtime() 323, 335
log() 349
log10() 349
log1p() 349
ltrim() 232
mail() 437
max() 339
mb_convert_encoding() 259
method_exists() 454
microtime() 309—310
min() 339
mkdir() 403
mktime() 310
money_format() 251
move_uploaded_file() 372
mt_getrandmax() 341
mt_rand() 340
mt_srand() 341
mysql_close() 571
mysql_connect() 569
mysql_erorr() 574
mysql_fetch_array() 574, 580
mysql_fetch_assoc() 574, 577, 580
mysql_fetch_object() 574, 582
mysql_fetch_row() 574, 576, 580
mysql_num_fields() 583
mysql_num_rows() 583—584
mysql_query() 573—575
mysql_result() 574—575
mysql_select_db() 572
natcasesort() 150
natsort() 150, 154
next() 134
nl2br() 238—239
NOW() 541
number_format() 251, 256
octdec() 343, 345
opendir() 404
ord() 221, 223
pack() 265—266
parse_str() 272
parse_url() 199—200
pathinfo() 269
php_uname() 194—195
phpinfo() 194, 612
phpversion() 194—195
pi() 353, 354
pos() 134
pow() 349
preg_grep() 289
preg_match() 288—289
preg_match_all() 289, 294
preg_quote() 289, 306
preg_replace() 289, 297
preg_replace_callback() 289, 302
preg_split() 289, 304
prev() 135
print() 65, 250, 251
print_r() 118, 194, 454, 580
printf() 250, 252
property_exits() 454
quoted_printable_decode() 248
quotemeta() 248
rad2deg() 343, 346
rand() 147, 340—341
range() 121, 122
rawurldecode() 199
rawurlencode() 199
readdir() 404, 406
readfile() 374
realpath() 269, 270
register_shutdown_function() 188
register_tick_function() 189

rename() 370—371
reset() 135
rewind() 395
rewinddir() 404
rmdir() 404
round() 346—347
rsort() 149, 152
rtrim() 232
scandir() 404, 408
serialize() 265, 267
session_cache_expire() 422
session_cache_limiter() 421
session_destroy() 433, 435
session_get_cookie_params() 433
session_id() 432, 435
session_is_registered() 432
session_name() 433
session_regenerate_id() 432
session_register() 432
session_save_path() 432
session_start () 432
session_start() 614
session_unregister() 432
set_time_limit() 615
setcookie() 425, 614
settype () 49
settype() 49
shuffle() 148
similar_text() 262, 264
sin() 353—354
sinh() 353
sizeof() 144
sleep() 193
sort() 149, 150
sprintf() 250
sqrt() 349
srand() 340—341
sscanf() 273, 279
str_ireplace() 231
str_pad() 271, 273
str_repeat() 271, 273
str_replace() 231—232
str_shuffle() 221
str_split() 274
str_word_count() 272, 277
strcasecmp() 260
strchr() 226
strcmp() 260, 262
strcoll() 260
strcspn() 261
strftime() 323, 327
strip_tags() 239, 246
stripcslashes() 247
stripos() 225
stripslashes() 247
stristr() 226, 230
strlen() 221, 222
strnatcasecmp() 261
strnatcmp() 261, 263
strncasecmp() 261
strncmp() 261
strpbrk() 226, 231
strpos() 225, 227
strptime() 323, 331
strrchr() 226, 230
strrev() 221, 224
strripos() 226
strrpos() 226, 228
strspn() 261
strstr() 226, 229
strtok() 272, 276
strtolower () 237
strtotime() 310, 314
strtoupper() 237
strtr () 231
strtr() 231, 234
strval() 49
substr() 225—226
substr_compare() 261
substr_count() 225, 227
substr_replace() 231, 233
tan() 353
tanh() 353
tempnam() 364, 369
time() 309—310
time_nanosleep() 193

timezone_abbreviations_list() 316, 318
timezone_identifiers_list() 316, 317
timezone_name_from_abbr() 316, 319
tmpfile() 364, 369
touch() 364
trim() 232, 235
uasort() 150
ucfirst() 237
ucwords() 237
uksort() 150
umask() 400
unlink() 370
unpack() 265
unregister_tick_function() 189
unserialize() 265, 267
unset() 194
urldecode() 199
urlencode() 199
usleep() 193
usort() 150
vfprintf() 250
vprintf() 250
vsprintf() 251
wordwrap() 272, 280
анонимная 180
вложенная 179
динамическая 179
неявное выполнение 188
объявление 165
параметры 168
проверка существования 182
рекурсивная 177

# Ч

Член:
класса 443
проверка существования 463
уничтожение 464

# Э

Экранирование 29

# Я

Язык SQL 514, 515