

# Разработка веб-приложений на **PHP 8**

Синтаксис PHP. Нововведения PHP 8.x

Объектно-ориентированное программирование на PHP 8.x

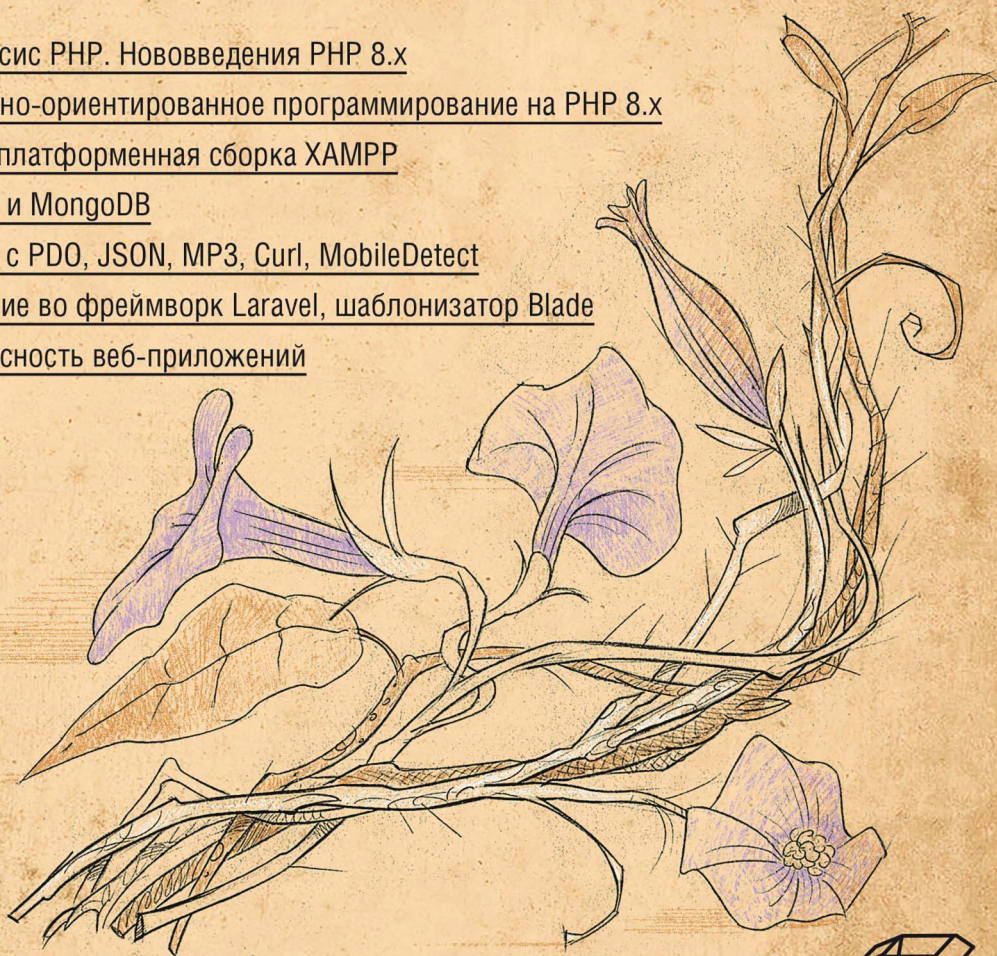
Кросс-платформенная сборка XAMPP

MySQL и MongoDB

Работа с PDO, JSON, MP3, Curl, MobileDetect

Введение во фреймворк Laravel, шаблонизатор Blade

Безопасность веб-приложений



**Дмитрий Колесниченко**



Материалы  
на [www.bhv.ru](http://www.bhv.ru)

Дмитрий Колесниченко

# Разработка веб-приложений на PHP 8

Санкт-Петербург  
«БХВ-Петербург»

2024

УДК 004.438 PHP  
ББК 32.973.26-018.1  
К60

**Колесниченко Д. Н.**

К60 Разработка веб-приложений на PHP 8. СПб.: БХВ-Петербург, 2024. — 496 с.: ил. — (Профессиональное программирование)  
ISBN 978-5-9775-1830-7

На практических примерах описано создание веб-приложений на языке PHP версии 8.x. Даны начала разработки на PHP: установка и настройка Apache 2.4, PHP, MySQL и кросс-платформенной сборки XAMPP, выбор редактора PHP-кода, синтаксис языка, самые полезные функции и нововведения PHP 8.x. Рассмотрено создание веб-приложений с использованием популярного фреймворка Laravel и шаблонизатора Blade. В качестве хранилища данных использованы два сервера — самая современная версия MySQL и набирающая популярность СУБД MongoDB. Раскрыты особенности создания индикатора загрузки файла и разыменовывания массивов, приведены примеры устранения типичных SEO-ошибок, допускаемых программистами, описана работа с PDO, JSON, MP3, Curl, MobileDetec. Особое внимание уделяется безопасности веб-приложений — рассматривается, как уберечь их от основных атак, как установить SSL-сертификат и уберечь сам сервер от неприятностей.

На сайте издательства находятся дополнительные главы, листинги из книги, а также необходимое программное обеспечение.

*Для веб-программистов*

УДК 004.438 PHP  
ББК 32.973.26-018.1

**Группа подготовки издания:**

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн обложки	<i>Зои Канторович</i>

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

ISBN 978-5-9775-1830-7

© ООО "БХВ", 2024  
© Оформление. ООО "БХВ-Петербург", 2024

# Оглавление

- Предисловие ..... 15**
- Немного истории ..... 15
- Основные нововведения в PHP версии 8 ..... 17
  - Улучшена производительность благодаря компиляции Just-In-Time ..... 17
  - Именованные аргументы ..... 17
  - Атрибуты вместо аннотаций PHPDoc ..... 17
  - Объявление свойств в конструкторе ..... 18
  - Выражение *Match* и оператор *nullsafe* ..... 18
  - Новые классы, интерфейсы и функции ..... 19
  - Улучшения в системе типов и обработке ошибок ..... 19
  - Прочие улучшения синтаксиса ..... 20
- ЧАСТЬ I. ТЕОРИЯ ..... 21**
- РАЗДЕЛ 1. БЫСТРЫЙ СТАРТ ..... 23**
- Глава 1. Установка необходимого программного обеспечения ..... 25**
  - 1.1. Нужно ли устанавливать программное обеспечение? ..... 25
  - 1.2. Выбор PHP-редактора и FTP-клиента ..... 27
  - 1.3. Установка связки Apache + PHP + MySQL в Windows ..... 30
    - 1.3.1. Для опытных пользователей: установка вручную ..... 30
      - Установка веб-сервера Apache ..... 30
      - Установка PHP ..... 32
    - 1.3.2. Установка для новичков (рекомендуется) ..... 39
- Глава 2. Программа на PHP ..... 43**
  - 2.1. Ваша первая программа ..... 43
  - 2.2. Запуск PHP-программы ..... 44
  - 2.3. Вывод текста без *echo* ..... 45
- Глава 3. Основы синтаксиса PHP ..... 48**
  - 3.1. Переменные ..... 48
    - 3.1.1. Правила объявления переменных. Имена переменных ..... 48
    - 3.1.2. Типы данных переменных ..... 49

3.1.3. Булевы переменные .....	51
3.1.4. Операции над переменными .....	51
3.1.5. Ссылки .....	52
3.2. Константы .....	53
3.3. Выражения и операции .....	57
3.3.1. Что такое <i>выражение</i> ? .....	57
3.3.2. Арифметические операции .....	57
3.3.3. Логические выражения .....	58
3.3.4. Битовые операции .....	58
3.3.5. Приоритеты операций .....	59
3.3.6. Операторы эквивалентности <code>==</code> и <code>===</code> .....	60
3.3.7. Оператор <code>??</code> — сокращенная форма тернарной условной операции .....	61
3.3.8. Операции со строками .....	61
3.3.9. Оператор <i>nullsafe</i> .....	62
3.4. Условный оператор .....	63
3.5. Циклы .....	64
3.5.1. Цикл со счетчиком <i>for</i> .....	64
3.5.2. Цикл <i>while</i> .....	65
3.5.3. Цикл <i>do-while</i> .....	65
3.5.4. Принудительное завершение цикла и пропуск итерации .....	65
3.6. Оператор выбора <i>switch-case</i> .....	66
3.7. Выражение <i>match</i> в PHP 8 .....	67
<b>Глава 4. Файл конфигурации <i>php.ini</i> .....</b>	<b>68</b>
4.1. Каталог <i>/etc/php</i> .....	68
4.2. Параметры памяти .....	69
4.3. Zend OPcache .....	69
4.4. Максимальное время выполнения .....	71
4.5. Загрузка файлов .....	72
4.6. Обработка сессий .....	72
4.7. Буферизация вывода .....	73
4.8. Директива <i>error_reporting()</i> .....	73
4.9. Отключение потенциально опасных функций .....	73
4.10. Директива <i>allow_url_open</i> .....	74
<b>РАЗДЕЛ 2. ПЕРЕДАЧА ПАРАМЕТРОВ PHP-ПРОГРАММАМ .....</b>	<b>75</b>
<b>Глава 5. Методы <i>GET</i> и <i>POST</i> .....</b>	<b>77</b>
5.1. Интерфейс CGI .....	77
5.2. Метод <i>GET</i> .....	79
5.3. Метод <i>POST</i> .....	79
<b>Глава 6. Протокол HTTP и интерфейс CGI .....</b>	<b>81</b>
6.1. Специальные переменные окружения CGI .....	81
6.2. Заголовки протокола HTTP .....	82
6.3. Коды ответов протокола HTTP .....	83
<b>Глава 7. Передача параметров посредством HTML-формы .....</b>	<b>85</b>
7.1. Создание простейшей формы и ее обработка в сценарии .....	85

7.2. Создание пользовательского интерфейса с помощью формы .....	88
7.2.1. Ввод текста. Теги <i>INPUT</i> и <i>TEXTAREA</i> .....	88
7.2.2. Зависимые и независимые переключатели.....	90
7.2.3. Списки выбора.....	91
7.2.4. Форма для передачи файлов .....	93
7.2.5. Кнопки .....	93
7.3. Проверка параметров формы.....	94
7.3.1. Проверка корректности e-mail.....	95
7.3.2. Проверка правильности номера банковской карты .....	96
7.3.3. Удаление лишних пробелов.....	97
7.4. Директива <i>@csrf</i> шаблонизатора Blade .....	97

## **Глава 8. Не забываем о поисковой оптимизации .....**

8.1. «Дружественные» интернет-адреса.....	99
8.1.1. Организация SEF URL с помощью файла <i>.htaccess</i> .....	100
8.1.2. Использование фреймворков.....	102
8.2. Идентификаторы сеансов.....	102
8.3. Производительность сценария .....	102
8.3.1. Использование шаблонизатора. Сокращение количества инструкций <i>echo</i> .....	103
8.3.2. Включение OPcache.....	103
8.3.3. Включение HTTP 2.0.....	104
8.3.4. Обновление версии PHP .....	104

## **РАЗДЕЛ 3. МАССИВЫ И СПИСКИ.....**

## **Глава 9. Основные операции над массивами и списками .....**

9.1. Массив и список. Цикл <i>foreach</i> .....	107
9.2. Функции <i>list()</i> и <i>array()</i> .....	110
9.3. Удаление массива .....	112
9.4. Слияние массивов.....	112
9.5. Функция <i>print_r()</i> .....	114
9.6. Разыменовывание массива.....	115

## **Глава 10. Функции сортировки массивов.....**

10.1. Сортировка массивов .....	117
10.2. Функция <i>sort()</i> — сортировка списка.....	117
10.3. Функция <i>asort()</i> — сортировка массива по значениям .....	118
10.4. Функция <i>ksort()</i> — сортировка по ключам .....	119
10.5. Функции <i>array_reverse()</i> и <i>shuffle()</i> .....	120
10.6. Собственная функция сортировки.....	120
10.7. Натуральная сортировка .....	121

## **Глава 11. Особые операции над массивами.....**

11.1. Добавление и удаление элементов массива.....	123
11.2. Упаковка переменных в массив и их извлечение .....	124
11.3. Получение части массива.....	126
11.4. Функции автоматического заполнения массива .....	126
11.5. Сравнение массивов .....	127

11.6. Полезные операции над массивом .....	128
11.6.1. Вычисление суммы и произведения всех элементов массива .....	128
11.6.2. Проверка существования элемента в массиве.....	128
11.6.3. Получение случайного элемента из массива.....	129
11.6.4. Удаление дубликатов из массива .....	129
11.6.5. Получение значений и ключей массива.....	130
11.6.6. Замена местами значений и ключей.....	130
11.6.7. Подсчет значений в массиве .....	130
11.6.8. Замена в массиве.....	131
11.6.9. Поиск в массиве .....	131
11.6.10. Прогулка по массиву .....	132

## **РАЗДЕЛ 4. ФУНКЦИИ В PHP ..... 135**

### **Глава 12. Полезные стандартные функции. Работа с датой..... 137**

12.1. Генератор случайных чисел.....	137
12.2. Дата и время.....	138
12.2.1. Кратко о timestamp.....	138
12.2.2. Функции <i>strtotime()</i> и <i>checkdate()</i> .....	138
12.2.3. Вывод даты.....	139
12.2.4. Использование <i>type="date"</i> .....	141
12.2.5. Функция <i>checkdate()</i> : проверка даты на корректность .....	141
12.2.6. Класс <i>DateTime</i> : удобная работа с датой и временем.....	143
Создание даты по строке .....	143
Вычисление количества дней между двумя датами.....	144
Работа с временными интервалами .....	145
12.2.7. Настройка PHP для корректной работы с датами.....	146
12.3. Математические функции.....	146

### **Глава 13. Функции для работы со строками ..... 148**

13.1. Основные строковые функции .....	148
13.2. Специальные функции замены .....	152
13.3. Функции преобразования строки .....	154
13.4. Функции преобразования кодировок .....	156
13.5. Функции для работы с отдельными символами строки. Разыменовывание строки .....	157
13.6. Функция <i>md5()</i> и другие функции шифрования/хеширования. API хеширования паролей .....	158
13.7. Функции <i>explode()</i> и <i>implode()</i> : работа с подстроками .....	159
13.8. Статистические функции .....	160
13.9. Функции вывода текста.....	161
13.10. Установка локали.....	163
13.11. Форматирование чисел и денежных величин.....	164
13.12. Преобразование систем счисления .....	164
13.13. Строки в PHP 7/8 .....	165

### **Глава 14. Работаем с файлами и каталогами ..... 166**

14.1. Права доступа в UNIX.....	166
14.2. Чтение файла.....	168
14.2.1. Функции <i>fopen()</i> и <i>fread()</i> .....	169

14.2.2. Функция <i>file()</i> : построчное чтение файла .....	171
14.2.3. Чтение всего файла: функция <i>file_get_contents()</i> .....	172
14.3. Запись файла .....	172
14.4. Создание временных файлов .....	173
14.5. Работа с CSV-файлами .....	173
14.6. Специальные функции для работы с файлами .....	175
14.6.1. Функции для работы с именами файлов .....	175
14.6.2. Работа с правами доступа .....	176
14.6.3. Копирование, переименование и удаление файлов .....	177
14.6.4. Время доступа к файлу .....	178
14.6.5. Другие полезные функции .....	178
14.7. Совместный доступ к файлу .....	179
14.8. Функции для работы с каталогами .....	180
<b>Глава 15. Вывод графических изображений средствами PHP .....</b>	<b>182</b>
15.1. Библиотека GD .....	182
15.1.1. Получение информации об изображении .....	182
15.1.2. Конвертирование графических форматов .....	185
15.1.3. Вывод текста поверх картинки. Задание цвета .....	187
15.1.4. Прозрачность .....	190
15.1.5. Рисование графических примитивов .....	191
15.1.6. Поворот изображения .....	193
15.2. Изменение размера изображения .....	193
15.3. Создание водяных знаков .....	195
15.4. Поддержка графического формата WebP .....	197
<b>Глава 16. Работа с сетевыми сокетами в PHP. Сетевые функции .....</b>	<b>199</b>
16.1. Еще раз о том, что такое сокет .....	199
16.2. Функция <i>fsockopen()</i> .....	199
16.3. Примеры работы с сокетами .....	200
16.3.1. Работаем с протоколом HTTP .....	200
16.3.2. Отправка почты с использованием сокетов .....	202
16.3.3. Простейший клиент/сервер .....	204
16.4. Блокирующий и неблокирующий режимы сокета .....	206
16.5. DNS-функции .....	206
<b>Глава 17. Собственные функции .....</b>	<b>208</b>
17.1. Зачем нужны собственные функции? .....	208
17.2. Особенности функций в PHP .....	208
17.3. Объявление функции .....	209
17.4. Области видимости функции .....	211
17.5. Вложенность функций .....	211
17.6. Переменное число аргументов .....	213
17.7. Передача массивов в качестве параметров .....	214
17.8. Передача аргументов по ссылке .....	216
17.9. Генераторы .....	217
17.10. Полезные примеры .....	219
17.10.1. Получение реального IP-адреса клиента .....	219
17.10.2. Генерирование сложного пароля .....	220

17.10.3. Рекурсивное удаление каталога.....	220
17.10.4. Отправка файла в браузер.....	221
17.10.5. Сжатие файла «на лету».....	222
<b>РАЗДЕЛ 5. БАЗА ДАННЫХ MYSQL .....</b>	<b>225</b>
<b>Глава 18. Установка MySQL на VDS .....</b>	<b>227</b>
18.1. Несколько вводных слов.....	227
18.2. Установка сервиса MySQL .....	227
18.3. Настройка MySQL .....	228
18.4. Создание MySQL-пользователя.....	229
18.5. Запуск и останов сервера .....	230
<b>Глава 19. Основы SQL .....</b>	<b>231</b>
19.1. Немного истории .....	231
19.2. Преимущества SQL .....	231
19.3. Как выглядят запросы?.....	232
19.4. Что такое база данных? .....	233
19.5. Создание таблиц .....	234
19.6. Добавление записей в таблицу .....	238
19.7. Обновление записей .....	238
19.8. Выборка записей.....	239
19.9. Удаление записей.....	240
19.10. Встроенные функции.....	241
19.11. Группировка записей. Сложные запросы .....	242
19.12. Копирование записей из одной таблицы в другую .....	245
19.13. Кеширование запросов.....	245
<b>Глава 20. Функции для работы с MySQL.....</b>	<b>248</b>
20.1. Способы работы с базой данных.....	248
20.2. Расширение <i>mysqli</i> .....	249
20.2.1. Подключение к серверу MySQL.....	249
20.2.2. Передача запросов серверу.....	250
20.2.3. Метод <i>real_escape_string()</i> .....	252
20.3. Расширение PDO .....	252
20.3.1. Соединение с базой данных.....	252
20.3.2. Выполнение запросов и чтение результата .....	254
20.3.3. Получение данных.....	256
20.3.4. Особенности использования операторов <i>LIKE</i> , <i>LIMIT</i> и <i>IN</i> .....	257
20.3.5. Имена таблиц и полей при работе с PDO .....	258
20.3.6. Запросы вставки и обновление .....	258
<b>Глава 21. Работа с базой данных в Laravel.....</b>	<b>260</b>
21.1. Способы работы с базой данных.....	260
21.2. Сырые (прямые) запросы.....	261
21.3. Конструктор запросов .....	262
21.4. Система Eloquent .....	265

<b>РАЗДЕЛ 6. ИНСТРУМЕНТЫ ДЛЯ СОЗДАНИЯ СЛОЖНЫХ ПРОЕКТОВ .....</b>	<b>269</b>
<b>Глава 22. Разработка собственного шаблонизатора .....</b>	<b>271</b>
22.1. Организация файлов и каталогов проекта .....	271
22.2. Выносим параметры в отдельный файл .....	273
22.3. Подключение дополнительных файлов .....	274
22.3.1. Инструкции <i>include</i> и <i>require</i> .....	274
22.3.2. Альтернативный способ подключения сценариев .....	275
22.3.3. Инструкции <i>include_once</i> и <i>require_once</i> .....	277
22.4. Шаблоны .....	277
<b>Глава 23. Шаблонизатор Blade .....</b>	<b>282</b>
23.1. Введение в Blade .....	282
23.2. Вывод значений скалярных переменных .....	283
23.3. Директивы Blade .....	283
23.3.1. Директива <i>@if</i> .....	283
23.3.2. Директивы <i>@for</i> , <i>@foreach</i> и <i>@while</i> .....	284
23.3.3. Директивы <i>@forelse</i> и <i>@endforelse</i> .....	284
23.4. Включение представлений. Директива <i>@include</i> .....	285
23.5. Директива <i>@csrf</i> .....	287
<b>Глава 24. Объектно-ориентированное программирование .....</b>	<b>288</b>
24.1. Основы ООП .....	288
24.2. Классы и объекты .....	289
24.3. Конструкторы и деструкторы класса .....	291
24.4. Наследование классов. Полиморфизм .....	293
24.5. Область видимости членов класса .....	294
24.6. Абстрактные классы и методы .....	295
24.7. Служебное слово <i>final</i> .....	295
24.8. Клонирование объектов .....	296
24.9. Константы — члены класса .....	297
24.10. Статические члены класса .....	297
24.11. Оператор <i>instanceof</i> .....	298
24.12. Итераторы .....	298
24.13. Пространства имен .....	299
24.13.1. Общая концепция .....	299
24.13.2. Объявление пространства имен .....	300
24.13.3. Псевдонимы .....	301
24.14. Типажи .....	302
24.15. Вызов метода или свойства класса выражением .....	303
24.16. Генераторы .....	303
24.17. Атрибуты .....	305
<b>Глава 25. Хранение данных в Cookies и сессиях .....</b>	<b>306</b>
25.1. Зачем нужны Cookies и сессии? .....	306
25.2. Cookies или хранение данных на стороне клиента .....	306
25.2.1. Что такое Cookies? .....	306
25.2.2. Установка Cookies .....	307

25.2.3. Удаление Cookies.....	308
25.2.4. Организация корзины с помощью Cookies.....	309
25.3. Механизм сессий.....	311
25.3.1. Для чего нужны сессии? .....	311
25.3.2. Автоматическое создание сессии.....	312
25.3.3. Хранение данных в сессии.....	312
<b>Глава 26. Обработка исключений.....</b>	<b>313</b>
26.1. Введение в обработку исключений.....	313
26.2. Блок <i>catch</i> .....	314
26.3. Блок <i>finally</i> .....	315
26.4. Глобальный обработчик исключений.....	317
<b>Глава 27. Контроль версий.....</b>	<b>318</b>
27.1. Выбор системы контроля версий.....	318
27.2. Первоначальная настройка .....	319
27.3. Создание нового репозитория или получение его по существующему URL-адресу.....	319
27.4. Операции с файлами. Перемещение и удаление версий файлов репозитория.....	320
27.5. Сохранение и восстановление незавершенных изменений.....	321
27.6. Просмотр изменений и создание коммитов (фиксация изменений) .....	321
27.7. Коллективная работа .....	322
27.8. Просмотр и изучение истории изменений файлов проекта .....	322
27.9. Откат изменений. Удаление ошибок и корректировка созданной истории .....	323
27.10. Синхронизация с удаленным репозиторием.	
Регистрация удаленного репозитория и обмен изменениями .....	323
<b>Глава 28. Тестирование RНР-сценариев.....</b>	<b>324</b>
28.1. Программа работает, но не так, как нам нужно .....	324
28.2. «Самодельные» точки останова.....	325
28.3. Система автоматического тестирования.....	327
28.4. Директива <i>error_reporting</i> .....	330
<b><u>ЧАСТЬ II. ПРАКТИКА</u> .....</b>	<b>331</b>
<b>РАЗДЕЛ 7. РАЗРАБОТКА ОСНОВНЫХ ЭЛЕМЕНТОВ САЙТА .....</b>	<b>333</b>
<b>Глава 29. Загрузка файлов на сервер .....</b>	<b>335</b>
29.1. Что нужно знать о загрузке файлов на сервер?.....	335
29.2. Реализация загрузки файла .....	338
29.3. Загрузка нескольких файлов.....	340
29.4. Индикатор загрузки файла .....	342
29.4.1. Некоторые теоретические предпосылки.....	342
29.4.2. Пример практической реализации .....	345
29.5. Проблемы при загрузке файлов.....	352
<b>Глава 30. Использование FTP-функций .....</b>	<b>353</b>
30.1. Функции для работы с FTP .....	353
30.2. Примеры использования FTP-функций .....	356

<b>Глава 31. Отправка и прием почты.....</b>	<b>359</b>
31.1. Отправка почты средствами PHP: функция <i>mail()</i> .....	359
31.1.1. Использование функции .....	359
31.1.2. Подробно о настройке сервера .....	360
31.2. Класс <i>PHPMailer</i> . Разработка сценария автоматической рассылки прайс-листа .....	361
31.3. Получение писем по протоколу POP3 .....	365
31.4. Получение писем по протоколу IMAP .....	367
<b>Глава 32. Введение в PEAR.....</b>	<b>370</b>
32.1. Серьезные проекты и PEAR.....	370
32.2. Пример использования класса <i>DB</i> .....	372
<b>Глава 33. Импорт и экспорт данных .....</b>	<b>375</b>
33.1. Импорт прайс-листов из формата CSV в базу данных MySQL .....	375
33.2. Преобразование файлов Excel в CSV с помощью PHP. Импорт прайсов из Excel .....	379
33.3. Работа с XML-файлами .....	380
33.3.1. Парсинг XML-файла .....	380
33.3.2. Генерирование XML-файла .....	382
<b>Глава 34. Работаем с MP3 .....</b>	<b>384</b>
34.1. Формат MP3 .....	384
34.2. Библиотека PEAR .....	385
34.3. Вывод ID3-тегов .....	386
34.4. Редактирование ID3-тегов .....	388
34.5. Удаление тега.....	388
<b>Глава 35. Расширение cURL: практические примеры .....</b>	<b>390</b>
35.1. Этот загадочный cURL.....	390
35.2. Авторизация на сайте и загрузка файла после нее.....	392
35.3. Замена функции <i>file_get_contents()</i> с помощью cURL.....	395
35.4. Загрузка файла через FTP .....	395
35.5. Проверка доступности сайта.....	396
<b>РАЗДЕЛ 8. ВВЕДЕНИЕ В LARAVEL.....</b>	<b>399</b>
<b>Глава 36. Фреймворк или чистый PHP-код? .....</b>	<b>401</b>
36.1. Что такое фреймворк? .....	401
36.2. Обзор популярных PHP-фреймворков.....	402
36.2.1. Zend Framework и The Laminas Project.....	402
36.2.2. Laravel.....	403
36.2.3. Symfony.....	403
36.2.4. Yii.....	403
36.2.5. CodeIgniter.....	404
<b>Глава 37. Установка Laravel на VDS с Ubuntu Linux.....</b>	<b>405</b>
37.1. Выбор места для установки .....	405
37.2. Установка необходимого ПО .....	405
37.2.1. Установка веб-сервера Apache .....	405
37.2.2. Установка PHP и его расширений.....	407
37.2.3. Установка Laravel .....	409

<b>Глава 38. Определяем маршруты .....</b>	<b>412</b>
38.1. Запросы протокола HTTP .....	412
38.2. Модель. Представление. Контроллер .....	413
38.3. Простейшие маршруты. Сопоставление маршрута с контроллером .....	414
38.4. Параметры в маршрутах .....	415
38.5. Имена маршрутов .....	416
38.6. Префиксы маршрутов .....	417
38.7. Маршруты новостной страницы .....	417
<b>Глава 39. Пишем контроллер .....</b>	<b>418</b>
39.1. Создание контроллера.....	418
39.2. Простой метод: <i>List()</i> .....	419
39.3. Методы с параметрами: <i>View()</i> и <i>GetArchiveContent()</i> .....	420
39.4. Обработка POST-запроса .....	421
<b>Глава 40. Создаем представление .....</b>	<b>422</b>
40.1. Каталог <i>resources/views</i> .....	422
40.2. Получение данных из контроллера .....	423
40.3. Очистка кеша страниц и кеша представлений .....	424
<b>Глава 41. Запрос и ответ .....</b>	<b>425</b>
41.1. Работаем с запросами. Класс <i>Request</i> .....	425
41.1.1. Основные методы .....	425
41.1.2. Получение информации о пользователе/запросе.....	426
41.1.3. Работа с файлами.....	427
41.1.4. Собирая все вместе.....	427
41.2. Класс <i>Response</i> .....	428
<b>Глава 42. Работа с данными.....</b>	<b>430</b>
42.1. Файловая система .....	430
42.1.1. Конфигурация фасада <i>Storage</i> .....	430
42.1.2. Методы фасада <i>Storage</i> .....	431
42.1.3. Загрузка файлов на сервер .....	432
42.1.4. Метод <i>download()</i> .....	433
42.2. Сессии.....	433
42.3. Работа с Cookies.....	435
<b>РАЗДЕЛ 9. БЕЗОПАСНОСТЬ САЙТА.....</b>	<b>437</b>
<b>Глава 43. Как взламываются сайты и как этому помешать?</b>	
<b>Основные сведения.....</b>	<b>439</b>
43.1. Основные способы взлома сайта.....	439
43.2. Два самых распространенных метода взлома .....	441
43.2.1. Межсайтовый скриптинг.....	441
43.2.2. SQL-инъекции .....	443
43.3. Остальные методы.....	446
<b>Глава 44. SSL-сертификат для сайта.....</b>	<b>447</b>
44.1. Выбор сертификата .....	447
44.1.1. Основные типы сертификатов .....	447

44.1.2. Какой тип сертификата выбрать? .....	448
44.1.3. Особенности SSL-сертификатов разных типов .....	448
44.2. Где купить SSL-сертификат? .....	451
44.3. Установка сертификата на веб-сервер .....	452
44.3.1. Веб-сервер Apache2 .....	452
44.3.2. Веб-сервер Nginx .....	453
<b>Глава 45. Защита PHP с помощью конфигурационного файла .....</b>	<b>454</b>
45.1. Конфигурационный файл <i>php.ini</i> .....	454
45.2. Отключение потенциально опасных функций .....	455
45.3. Рекомендованные значения некоторых конфигурационных директив .....	455
<b>РАЗДЕЛ 10. ПОЛЕЗНЫЕ СВЕДЕНИЯ .....</b>	<b>457</b>
<b>Глава 46. Устанавливаем визуальный редактор Summernote .....</b>	<b>459</b>
46.1. Знакомство с редактором .....	459
46.2. Интеграция Summernote и Laravel .....	459
<b>Глава 47. Работа с MongoDB средствами PHP .....</b>	<b>462</b>
47.1. Что такое MongoDB? .....	462
47.2. Настройка интерпретатора PHP .....	463
47.3. Добавление данных в MongoDB .....	465
47.4. Чтение информации из базы данных .....	467
47.5. Преобразование объекта в массив .....	469
<b>ПРИЛОЖЕНИЯ .....</b>	<b>473</b>
<b>Приложение 1. Шаблоны проектирования .....</b>	<b>475</b>
П1.1. Введение в шаблоны проектирования .....	475
П1.2. Шаблон «Стратегия» .....	476
П1.3. Шаблон «Адаптер» .....	477
П1.4. Шаблон «Фабрика» .....	478
П1.5. Шаблон «Одиночка» .....	479
<b>Приложение 2. Профайлинг .....</b>	<b>481</b>
П2.1. Что такое профайлинг? .....	481
П2.2. Типы профайлеров .....	481
П2.3. Профайлер Xdebug .....	482
П2.4. XHProf .....	482
<b>Приложение 3. Виртуальная машина HHVM .....</b>	<b>485</b>
П3.1. Что такое HHVM? .....	485
П3.2. Подойдет ли HHVM именно для вас? .....	486
П3.3. Установка HHVM .....	487
П3.4. Настройка HHVM .....	487
П3.5. Язык Hack .....	489
<b>Приложение 4. Описание электронного архива .....</b>	<b>490</b>
<b>Предметный указатель .....</b>	<b>492</b>

# Предисловие

RНР (Hypertext Preprocessor) — один из самых популярных языков программирования, используемый для разработки веб-приложений. В настоящее время РНР поддерживается подавляющим большинством хостинг-провайдеров, что делает его чуть ли не основным языком, с помощью которого можно разработать любой интернет-проект — от простенького сайта до крупного портала.

На РНР написано огромное количество как отдельных скриптов, так и завершенных проектов: форумов, систем управления контентом и др.

Электронный архив с информацией, расширяющей и дополняющей материал «бумажной» книги, можно скачать с сервера издательства «БХВ» по ссылке: <https://zip.bhv.ru/9785977518307.zip>, а также со страницы книги на сайте <https://bhv.ru>. Подробная информация об архиве находится в *приложении 4*.

## Немного истории

В 1994 году датский программист Расмус Лердорф (Rasmus Lerdorf) создал надстройку (набор скриптов) над Perl/CGI для вывода и учета посетителей своего сайта и назвал ее Personal Home Page (отсюда язык РНР и ведет свое название).

Но Perl — довольно медленный интерпретатор, и вскоре его производительности перестало хватать, поэтому разработчик написал на языке С новый интерпретатор и назвал его РНР/FI (Personal Home Page / Form Interpreter). В новом интерпретаторе четко прослеживались черты Perl — например, символ доллара в начале имени переменной. Тогда же были заложены черты современного РНР, такие как автоматическая обработка форм, встраиваемость в HTML и др.

В 1997 году появилась вторая версия интерпретатора — РНР/FI 2.0. Она была установлена примерно на 50 тыс. серверов. С одной стороны, цифра внушительная, но это всего лишь 1% от общего числа интернет-серверов на тот момент.

В 1998 году появился РНР 3.0. Именно с этой версии аббревиатура РНР стала рекурсивным акронимом РНР — Hypertext Preprocessor. Третья версия пользовалась существенно бóльшим успехом, чем вторая, — она была установлена на 10% всех интернет-серверов, а это уже не мало!

PHP 3.0 частенько критиковали за медленное ядро, и действительно — сценарии выполнялись медленно. Поэтому разработку версии PHP 4 посвятили переработке ядра. И теперь, начиная с PHP 4, интерпретатор сначала транслирует PHP-код во внутреннее представление, а потом выполняет его (а не выполняет сценарий строка за строкой, как в PHP 3), благодаря чему существенно повысилась производительность сценариев. Четвертая версия PHP появилась в мае 2000 года, обновления для нее выпускались до конца 2007 года, но с августа 2008 года поддержка PHP 4 была прекращена.

Пятая версия PHP появилась в июле 2004 года. Она отличалась переработанным ядром Zend, что увеличило эффективность интерпретатора, добавлением поддержки XML, переработанной поддержкой объектно-ориентированного программирования (ООП). С этого момента PHP стал полноценным объектно-ориентированным языком программирования, а его объектная модель во многом приблизилась к модели Java. Далеко не все хостинг-провайдеры успели в свое время перейти на версию PHP 5.4, выпущенную в 2012 году, поэтому как минимум еще несколько лет на хостингах встречалась версия 5.3, выпущенная в 2009 году.

Начиная с 2005 года велась разработка PHP 6 (что случилось с этой версией, вы узнаете чуть позже), но вместо нее в июне 2013 года вышла версия 5.5. В августе 2014 года была выпущена версия 5.6, но она не получила особого распространения.

Версия PHP 7 вышла 3 декабря 2015 года, и в последующие несколько лет ее развитие продолжалось в версиях до 7.4.

Восьмая версия PHP была выпущена 26 ноября 2020 года, и на текущий момент самой последней версией PHP является версия 8.2.5, появившаяся 23 апреля 2023 года.

Вернемся к версии PHP 6. Куда же она подевалась? Как уже отмечалось, в 2015 году миру была показана версия PHP 7. Но, как все мы помним, официального релиза PHP 6 так и не состоялось. Почему же разработчики не продолжили существующую нумерацию и не выпустили PHP 6, а «перешагнули» через шестую версию и явили миру сразу PHP 7?

Причин тому две. Релиз PHP 6 должен был состояться еще в 2005 году. Но из-за проблем с реализацией Unicode<sup>1</sup> разработчики из года в год откладывали выпуск шестой версии. Вместо нее все новинки включались в ветку PHP 5.x — появлялись версии 5.3–5.6 (самая последняя из ветки 5.x). Очевидно, разработчики не хотели, чтобы новая версия языка ассоциировалась с неудачами. Кроме того, разработчикам хотелось показать революционность новой версии, подчеркивая таким образом, что в ней очень много важных изменений. Вспомним, Microsoft поступила так же с Windows — все ждали Windows 9, а вышла сразу «десятка».

---

<sup>1</sup> Получить список всех функций, поддерживающих Unicode, можно по адресу: [http://www.php.net/~scoates/unicode/render\\_func\\_data.php](http://www.php.net/~scoates/unicode/render_func_data.php).

## Основные нововведения в PHP версии 8

### Улучшена производительность благодаря компиляции Just-In-Time

Каждая следующая версия PHP хоть немного, но быстрее, чем предыдущая. Но в случае с версией 8 улучшение производительности произошло не из-за очередной оптимизации, а благодаря появлению механизма компиляции Just-In-Time (JIT). В предыдущей версии, PHP 7, разработчики выжали максимум производительности, поэтому дальше без JIT-компилятора было уже никак не обойтись.

Нужно отметить, что для веб-приложений использование JIT не сильно улучшает скорость обработки запросов (а в некоторых случаях скорость будет даже меньше, чем без нее). Зато JIT существенно повышает скорость выполнения математических операций. В официальном релизе (<https://www.php.net/releases/8.0/ru.php>) сказано, что на синтетических бенчмарках PHP 8 показывает улучшение производительности примерно в 3 раза.

Включить JIT можно в настройках файла `php.ini`:

```
opcache.jit_buffer_size=32M
opcache.jit=1235
```

Эти две строчки не только включают JIT, но и устанавливают размер буфера. Подробнее о том, что такое JIT, можно прочитать в следующей статье (там же вы найдете тесты производительности): <https://medium.com/jp-tech/try-out-jit-compiler-with-php-8-0-a020e6aeb3e5>.

### Именованные аргументы

Именованные аргументы позволяют задавать только необходимые аргументы в ваших функциях, причем в произвольном порядке — нужно указать название аргумента и передать значение. Подробнее о том, как использовать именованные аргументы, будет рассказано в *главе 17*.

### Атрибуты вместо аннотаций PHPDoc

Вместо устаревших аннотаций PHPDoc вы теперь можете использовать *атрибуты* — структурные метаданные с нативным синтаксисом PHP. Например, до версии 8 аннотации выглядели так:

```
class PostsController
{
    /**
     * @Route("/api/posts/{id}", methods={"GET"})
     */
    public function get($id) { /* ... */ }
}
```

Сейчас же мы можем использовать атрибуты так:

```
class PostsController
{
    #[Route("/api/posts/{id}", methods: ["GET"])]
    public function get($id) { /* ... */ }
}
```

## Объявление свойств в конструкторе

Объявление свойств в конструкторе позволяет использовать меньше шаблонного кода для определения и инициализации свойств. Например:

```
class Player {
    public function __construct(
        public float $x = 0.0,
        public float $y = 0.0,
        public float $z = 0.0,
    ) {}
}
```

Подробнее новинка будет рассмотрена в *главе 24*.

## Выражение *Match* и оператор *nullsafe*

Новое выражение `Match` похоже на оператор `switch` в предыдущих версиях PHP, но вся его прелесть в том, что это выражение, а не оператор, следовательно, результат этого выражения может быть сохранен в переменной или возвращен.

Оператор `nullsafe` делает проверку на `null` не только безопасной, но и компактной. Вы только взгляните на код в предыдущих версиях PHP:

```
$country = null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```

В PHP 8 этот код можно заменить одной строкой:

```
$country = $session?->user?->getAddress()?->country;
```

Работает новый оператор следующим образом: когда один из элементов в последовательности возвращает `null`, выполнение прерывается и вся последовательность возвращает `null`.

## Новые классы, интерфейсы и функции

Разумеется, компилятором JIT и парой синтаксических нововведений дело не обошлось. В PHP 8 появился ряд новых функций и классов:

- ❑ функции `str_contains()`, `str_starts_with()`, `str_ends_with()` делают работу со строками проще. Странно, что эти функции появились только в версии 8, — до этого много лет разработчики писали свои варианты этих функций;
- ❑ класс `WeakMap` — предоставляет общий вариант использования слабых карт, когда данные связываются с отдельными экземплярами объектов, не предотвращая сборку мусора объектов, которые используются в качестве ключей. Если ключ объекта будет удален сборщиком мусора, он будет просто удален с карты;
- ❑ интерфейс `Stringable` — обозначает класс, реализующий метод `__toString()`. Его основное значение — разрешить функциям выполнять проверку типа на соответствие типу `union string/Stringable`, чтобы принимать либо строковый примитив, либо объект, который может быть преобразован в строку;
- ❑ функция `fdiv()` — реализует деление чисел с плавающей запятой в соответствии с семантикой IEEE-754.

Также был добавлен ряд других, менее полезных функций, которые вы вряд ли будете использовать на практике при создании веб-приложений.

## Улучшения в системе типов и обработке ошибок

В системе типов и обработке ошибок произошли следующие изменения:

- ❑ появился новый тип `mixed` — пока он используется только в подсказках в стиле RHPDoc, но общая идея такова, что этот тип должен позволить программистам добавлять типы к параметрам, свойствам класса и возвращаемым значениям функций, чтобы указать, что информация о типе не забыта, но сам тип еще не определен. В общем, пока его можно использовать как заглушку на случай, если пока с типом нет однозначности;
- ❑ более строгие проверки типов для арифметических/битовых операторов;
- ❑ генерирование фатальной ошибки при несовместимости сигнатур методов;
- ❑ наследование с приватными методами;
- ❑ возвращаемый тип `static`;
- ❑ оператор `@` больше не подавляет фатальные ошибки.

## Прочие улучшения синтаксиса

Наконец, произошли следующие незначительные улучшения синтаксиса:

- ❑ разрешена запятая в конце списка параметров `RFC` и в списке `use` замыканий;
- ❑ блок `catch` можно использовать без указания переменной;
- ❑ изменен синтаксис переменных;
- ❑ имена в пространстве имен рассматриваются как единый токен;
- ❑ выражение `Throw` (поскольку это выражение, а не оператор, то генерирование исключения возможно в любом контексте, где допускается использование выражения);
- ❑ добавление `::class` для объектов.



**ЧАСТЬ I**

---

**ТЕОРИЯ**



# I. РАЗДЕЛ 1

## Быстрый старт

Глава 1.	Установка необходимого программного обеспечения
Глава 2.	Программа на PHP
Глава 3.	Основы синтаксиса PHP
Глава 4.	Файл конфигурации <i>php.ini</i>



# ГЛАВА 1

## Установка необходимого программного обеспечения

### 1.1. Нужно ли устанавливать программное обеспечение?

Подготавливая книгу к печати, я вообще сомневался в целесообразности включения в нее такой главы. Почему? Если вы купили эту книгу, значит, со временем собираетесь профессионально заниматься PHP (разрабатывать сайты, сценарии) или же планируете создать собственный сайт на PHP. Следовательно, вам понадобится хостинг с поддержкой PHP. Учитывая, что PHP-хостинг сейчас стоит очень дешево (а можно найти и вовсе бесплатные варианты), как и безлимитный Интернет, то все, что требуется для PHP-разработки, — это какой-нибудь текстовый редактор с поддержкой PHP-синтаксиса.

Ведь вам всего-то необходимо написать сценарий. В свою очередь, сценарий — это обычный текстовый файл. Как только сценарий будет готов, его нужно загрузить по FTP или SSH на ваш хостинг и запустить через браузер. При этом совсем не имеет значения, какую операционную систему вы используете: Windows, Linux или даже Mac OS.

Ранее имело смысл устанавливать на свой компьютер связку Apache + PHP + сервер баз данных MySQL. Сейчас же можно производить разработку и отладку сценариев непосредственно на сервере. Так даже правильнее, поскольку конфигурации Apache и PHP на локальном и удаленном сервере в 99% случаев окажутся различными, поэтому при переносе сценариев могут возникнуть проблемы. Какие именно? Начиная с самых безобидных, вроде отличающихся параметров базы данных MySQL, до разницы в версии PHP. С параметрами базы данных все просто — при переносе сценария с одного сервера на другой не забывайте проверять параметры доступа, иначе ваши сценарии откажутся работать. Чуть позже мы продемонстрируем пример, позволяющий легко избавиться от этой проблемы. А вот с версией PHP и установленными PHP-расширениями проблемы могут возникнуть глобальные. Очень часто код, написанный для более новой версии PHP (например, для PHP8), оказывается несовместим с предыдущими версиями.

Иная распространенная проблема — отсутствие на другом сервере необходимых вам расширений. Например, вы создали галерею картинок, использующую функ-

ции из библиотеки GD. Но эта галерея не будет работать, если на сервере не установлено расширение GD (PHP-библиотека GD). Поэтому, прежде чем размещать сценарий на сервере, вызовите функцию `phpinfo()` и убедитесь, что на сервере установлены нужная версия PHP и нужные расширения.

Как узнать, под какую версию PHP готовить сценарии? Если вы только начали изучать PHP, тогда возьмите самую последнюю версию — PHP 8. Нет ни одной причины изучать старую версию. А вот если вы выбираете версию PHP для устанавливаемого веб-приложения, нужно сначала ознакомиться с его документацией, — вполне вероятно, что может понадобиться более старая версия.

И не забывайте, что в каждой версии PHP имеются изменения по сравнению с предыдущей. Например, новые параметры у функций и новые функции. Для изучения PHP по большому счету все равно, какую взять версию. Главное, чтобы вы знали, какая она именно, дабы не возникало потом вопросов, почему та или иная функция не работает так, как следует.

Как правило, когда вы покупаете хостинг, в настройках панели управления есть возможность выбора версии PHP. На рис. 1.1 показан экран модуля **PHP Selector** панели управления DirectAdmin. Как видите, текущей является версия 7.4, но вы можете выбрать версию 8.1, хотя самой последней является 8.2. Кстати, по тестам производительности именно 8.2 — самая быстрая версия PHP на текущий момент. На этом же экране можно не только выбрать версию PHP, но и расширения PHP.

### ПРИМЕЧАНИЕ

Не хочется делать никому рекламу, именно поэтому я не раскрываю, панель управления каким хостингом здесь показана. Но если вы ее узнали, то мое мнение об этом хостинге скорее отрицательное. Панель управления хорошая, а вот производительность сервера оставляет желать лучшего.

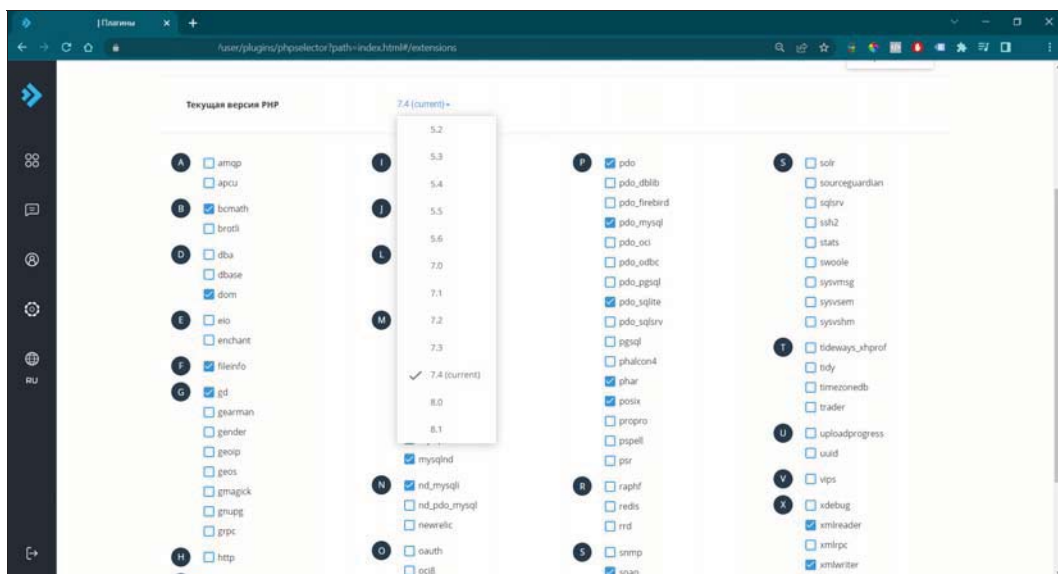


Рис. 1.1. Панель управления хостингом: выбор версии PHP

Чтобы узнать, какая версия PHP используется на том или ином хостинге:

- ❑ если хостинг уже куплен, напишите сценарий, вызывающий функцию `phpinfo()`, и запустите его. Кроме версии PHP, вы увидите массу другой полезной информации, в том числе список доступных расширений;
- ❑ если вы только планируете покупать хостинг, сведения о версии PHP имеются на веб-сайте хостера. Можно также обратиться в его службу поддержки — там подскажут, какие версии PHP доступны на их серверах.

Если вам нужен хостинг именно с поддержкой PHP 8, воспользуйтесь поиском Google, — множество компаний уже начали предоставлять такой хостинг. Также уточните наличие версии 8 у вашего хостера — вполне возможно, что новая версия уже поселилась на одном из его серверов и хостер сможет переместить ваш сайт на такой сервер.

Итак, если вы не собираетесь устанавливать веб-сервер Apache, интерпретатор PHP и сервер баз данных на свой компьютер, вам понадобятся две программы: текстовый редактор и FTP-клиент.

Начиная с версии 5.4, у PHP имеется свой встроенный веб-сервер, но я рекомендую все же использовать Apache, поскольку в большинстве случаев на сервере хостера будет установлен именно этот сервер.

## 1.2. Выбор PHP-редактора и FTP-клиента

Из всех PHP-редакторов, которыми я пользовался, больше всех мне понравились два редактора: PHP Expert Editor и Zend Studio (вы без проблем найдете в Интернете сайты их разработчиков). Оба редактора коммерческие, но они стоят своих денег. PHP Expert Editor (рис. 1.2) обеспечивает не только подсветку синтаксиса, но и умеет проверять синтаксис сценария. Правда, для проверки синтаксиса вам придется все же установить PHP (об этом позже) и указать его в настройках программы.

### ПРИМЕЧАНИЕ

Условно-бесплатную версию программы PHP Expert Editor вы найдете в каталоге *software* сопровождающего книгу электронного архива (см. *приложение 4*). Последнюю версию программы также всегда можно скачать с сайта разработчиков: <http://phpexperteditor.com/>.

Чтобы указать PHP в настройках программы PHP Expert Editor, выполните команду **Запуск | Настройки**, в открывшемся окне перейдите на вкладку **Интерпретаторы скриптов** (рис. 1.3) и укажите путь к интерпретатору PHP. Для проверки синтаксиса нажмите клавишу `<F7>`.

Возможности Zend Studio (рис. 1.4) куда шире — это не просто редактор, а настоящая IDE (среда разработки). Чего стоит только автодополнение кода, когда программа «читает» ваши мысли и дописывает за вас PHP-код. Если вы работали с IDE — например, с Visual Studio, то поймете, о чем я говорю. Поверьте, это очень удобно. Кстати, кроме Zend Studio вы также можете использовать редактор Visual Studio Code от Microsoft — он абсолютно бесплатный и также поддерживает синтаксис PHP.

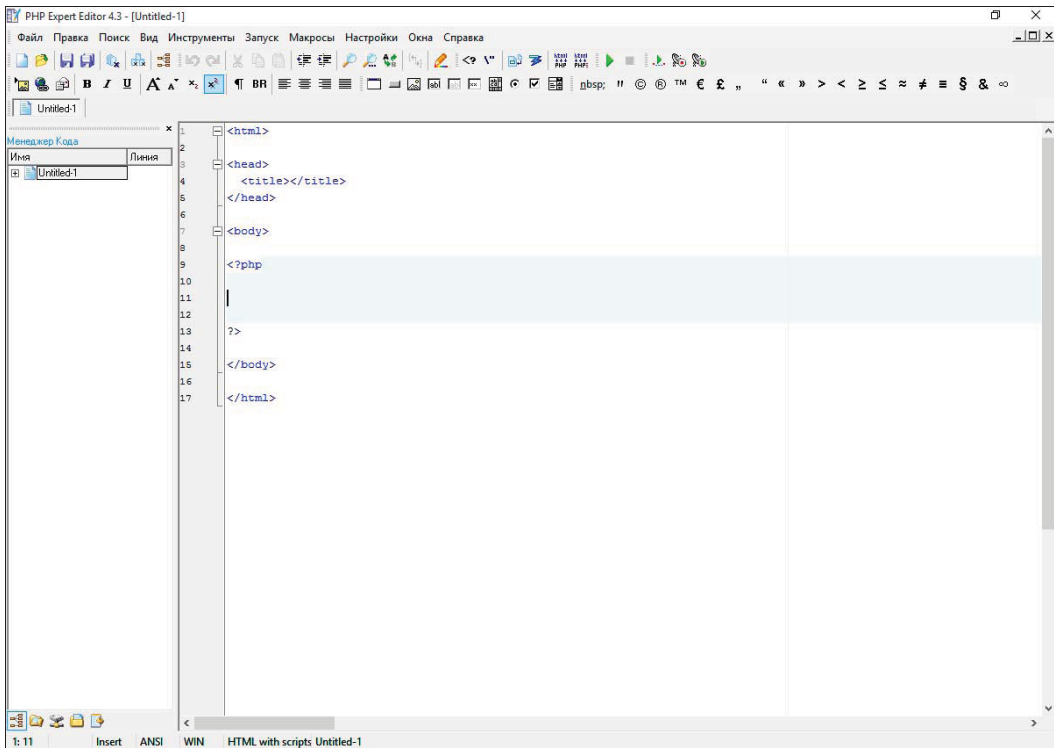


Рис. 1.2. Программа PHP Expert Editor

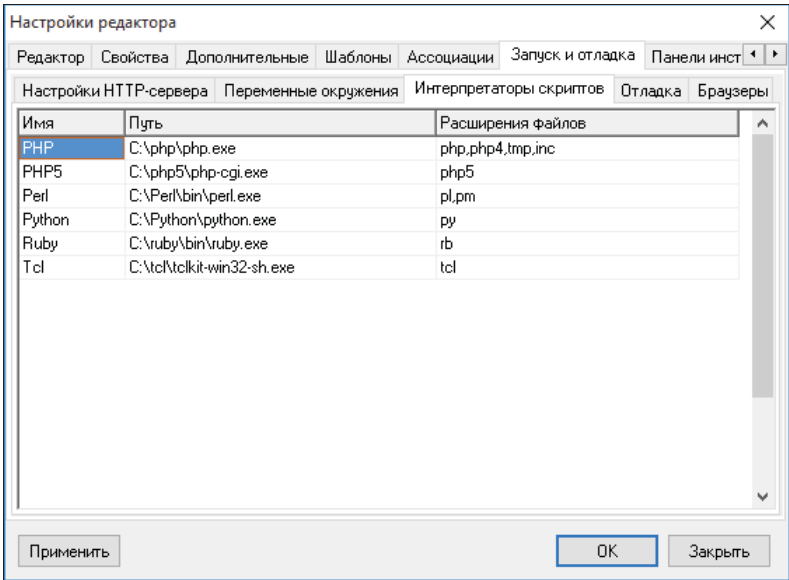
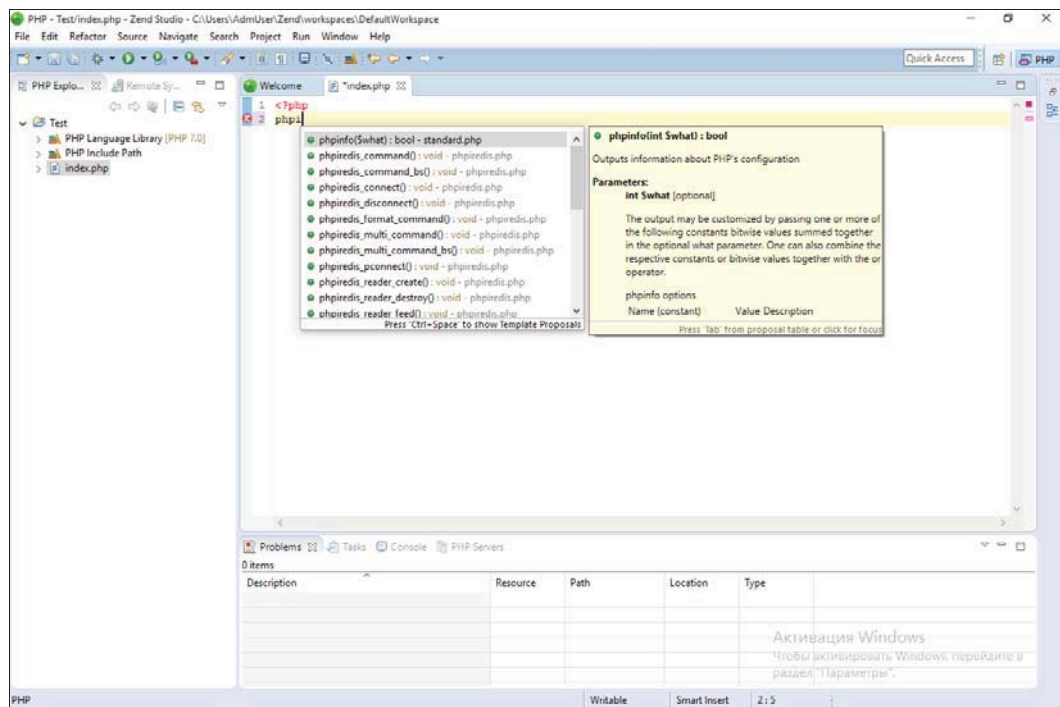


Рис. 1.3. Путь к интерпретатору PHP



**Рис. 1.4.** Zend выводит не только имена функций, но и краткую справку по каждой функции

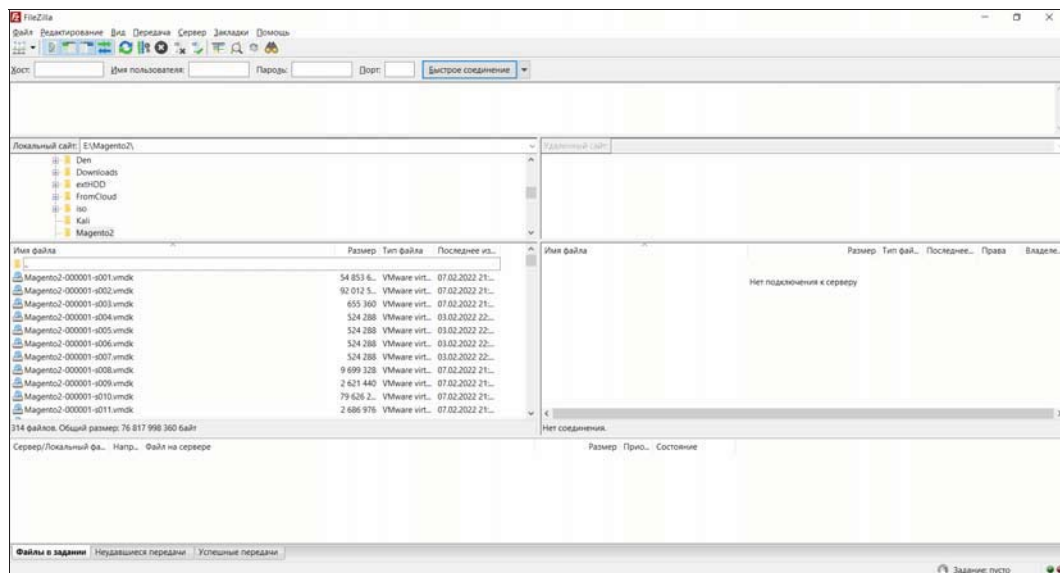


Рис. 1.5. Программа FileZilla

В качестве FTP-клиента могу порекомендовать бесплатную программу FileZilla (рис. 1.5), которая доступна как для Windows, так и для Linux. Кроме того, можно использовать FTP-клиент, встроенный в файловый менеджер Total Commander.

В случае с VDS<sup>1</sup>, правда, вам понадобится SSH-клиент — например, Bitvise SSH Client. Но пока вы — начинающий программист, вряд ли у вас будет собственный VDS. Начинать придется с хостинга.

## 1.3. Установка связки Apache + PHP + MySQL в Windows

### 1.3.1. Для опытных пользователей: установка вручную

Сразу хочу вас предупредить: этот способ требует затрат времени, и его описание приводится только в образовательных целях, — на тот случай, если вы желаете знать, как все устроено. Гораздо проще установить пакет XAMPP, который уже содержит всё необходимое (Apache, MySQL, PHP 7), и наслаждаться сэкономленным временем, поскольку сразу же после установки XAMPP вы сможете приступить к созданию сценариев. Но если у вас есть время и желание, тогда предлагаю продолжить чтение этого раздела. Будет интересно!

#### Установка веб-сервера Apache

Начнем с установки веб-сервера Apache. В настоящий момент имеются две его актуальные версии: 2.2 и 2.4. Грубо говоря: старая и новая. Так вот, первые версии PHP 7.0 прекрасно поддерживали версию Apache 2.2. Однако в последних сборках оказалось, что файл `php7apache2_2.dll`, позволяющий связать Apache 2.2 и PHP 7.0, просто отсутствует. Найти в Интернете мне его тоже не удалось. Отсюда можно сделать вывод, что поддержку Apache 2.2 из PHP 7.0 исключили. Следовательно, говорить о поддержке старой версии Apache более новыми версиями PHP (новее 7.0) не приходится. Поэтому выбираем версию 2.4.

#### ПРИМЕЧАНИЕ

Навигация на сайте Apache оставляет желать лучшего. Чтобы упростить вашу задачу, подсказываю ссылку, откуда можно скачать версию 2.4 для Windows: <https://www.apachelounge.com/download/> (именно с этого сайта предлагается скачать Windows-версию на официальном сайте Apache).

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете версии 2.2 и 2.4 для Windows. Linux-версии в архиве нет, поскольку Apache входит в состав любого дистрибутива Linux, и вам нужно будет его только установить.

Установить Apache 2.4 довольно просто. Прежде всего распакуйте архив `httpd-2.4.23-win32-VC14.zip` или `httpd-2.4.23-win64-VC14.zip` (в зависимости от разрядности Windows) в корневой каталог диска C:\ (или любого другого, но далее я буду предполагать, что вы распаковали архив на диск C:).

---

<sup>1</sup> VDS (Virtual Dedicated Server) или VPS (Virtual Private Server) — хостинг-услуга, где пользователю предоставляется виртуальный сервер с максимальными привилегиями.

## ОСОБЕННОСТИ УСТАНОВКИ В WINDOWS 10 И WINDOWS 11

Установку Apache, PHP и MySQL в Windows 10 и в Windows 11 желательно производить от имени администратора. Для этого щелкните на установочном файле правой кнопкой мыши и выберите команду **Запуск от имени администратора**.

Теперь запустите командную строку *от имени администратора*, перейдите в каталог `c:\Apache24\bin` и введите команду:

```
httpd.exe -k install
```

После этого вы увидите сообщение о том, что сервис Apache2.4 успешно установлен, а также окно, предлагающее настроить правила брандмауэра (рис. 1.6).

Управлять сервисом Apache можно или через приложение Apache Monitor (рис. 1.7, а), или через оснастку **Службы** (рис. 1.7, б) — как вам больше нравится.

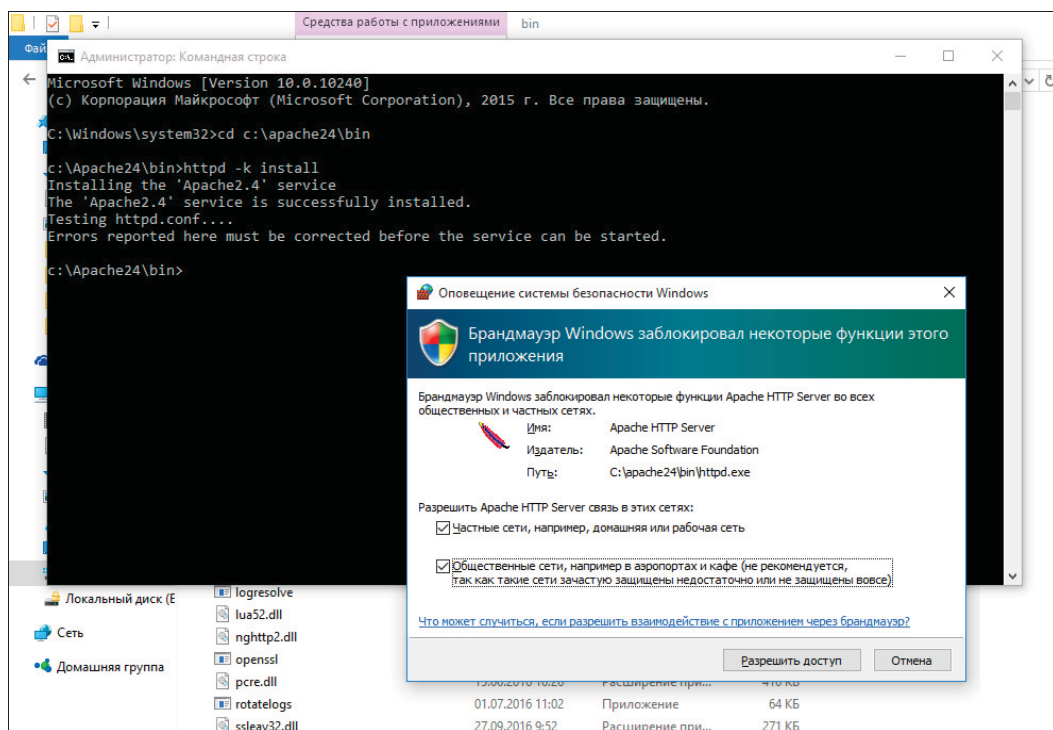
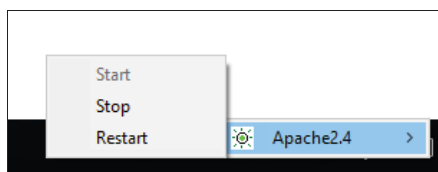


Рис. 1.6. Установка Apache 2.4

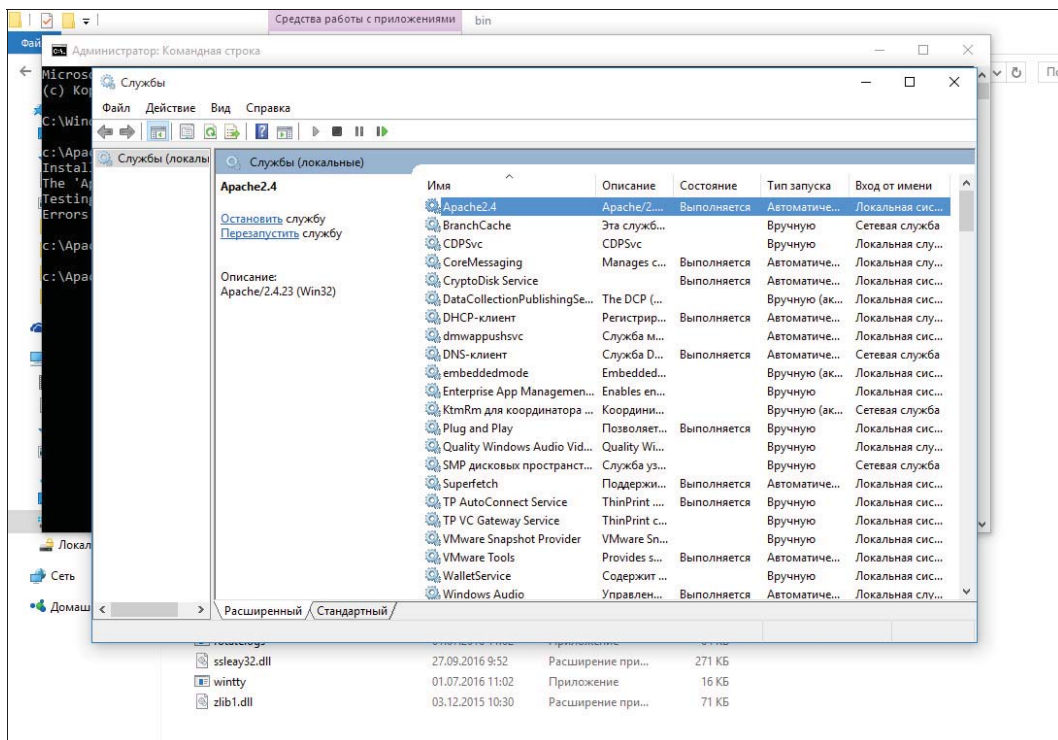
После установки сервиса Apache его нужно запустить, как уже было сказано, или через Apache Monitor, или с помощью оснастки **Службы**. Затем запустите браузер и введите следующий URL:

```
http://localhost
```

Если в ответ вы увидите сообщение **It works!** (рис. 1.8), значит, установка Apache прошла успешно.



а



б

Рис. 1.7. Приложение Apache Monitor (а); оснастка Службы (б)

## Установка PHP

По умолчанию сервер Apache не поддерживает PHP, поэтому PHP придется установить отдельно. Ранее (до версии 5.4) в PHP был доступен инсталлятор, который существенно упрощал процесс установки, помогал внести все необходимые изменения в конфигурационные файлы Apache и изменить параметры Windows. Сейчас все это придется делать вручную. Если вам этим заниматься не с руки, удалите Apache и установите пакет XAMPP, где в одном флаконе содержится все необходимое: Apache, PHP и MySQL, и вам вручную ничего настраивать не придется.

### ПРИМЕЧАНИЕ

В каталоге *software* сопровождающего книгу электронного архива (см. приложение 4) вы найдете различные версии PHP.

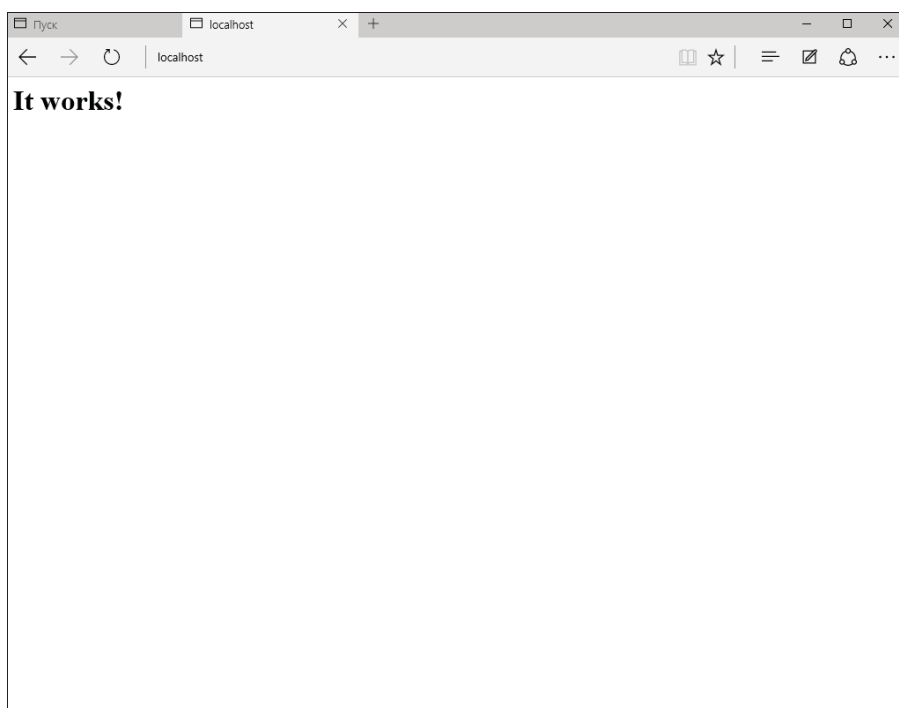


Рис. 1.8. Веб-сервер работает!

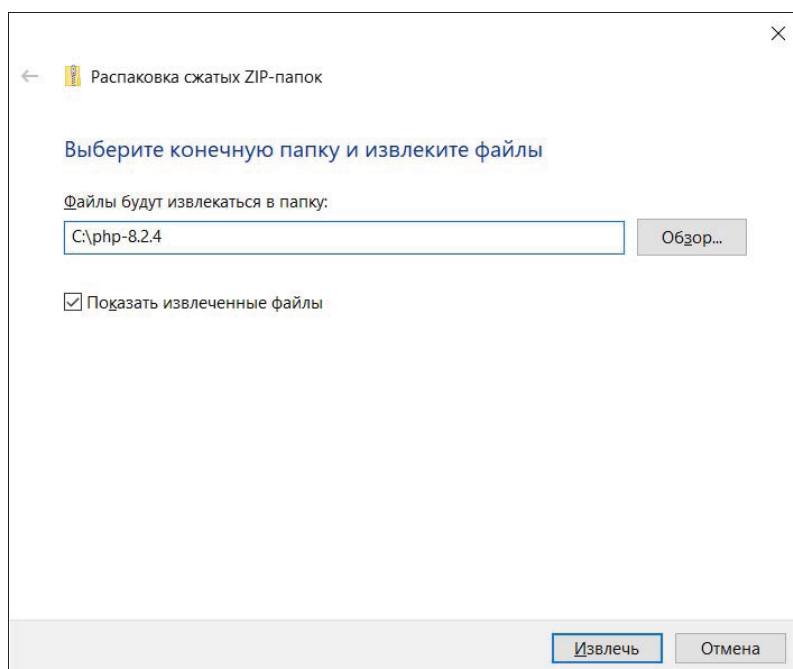


Рис. 1.9. Распаковка PHP

Итак, распакуйте архив с РНР в каталог C:\php8-2.4 (рис. 1.9).

После этого добавьте каталог C:\php-8.2.4 в путь поиска приложений. Для этого выполните следующие действия:

1. Откройте окно **Система** (рис. 1.10).
2. В окне **Система** на панели слева выберите команду **Дополнительные параметры системы**.

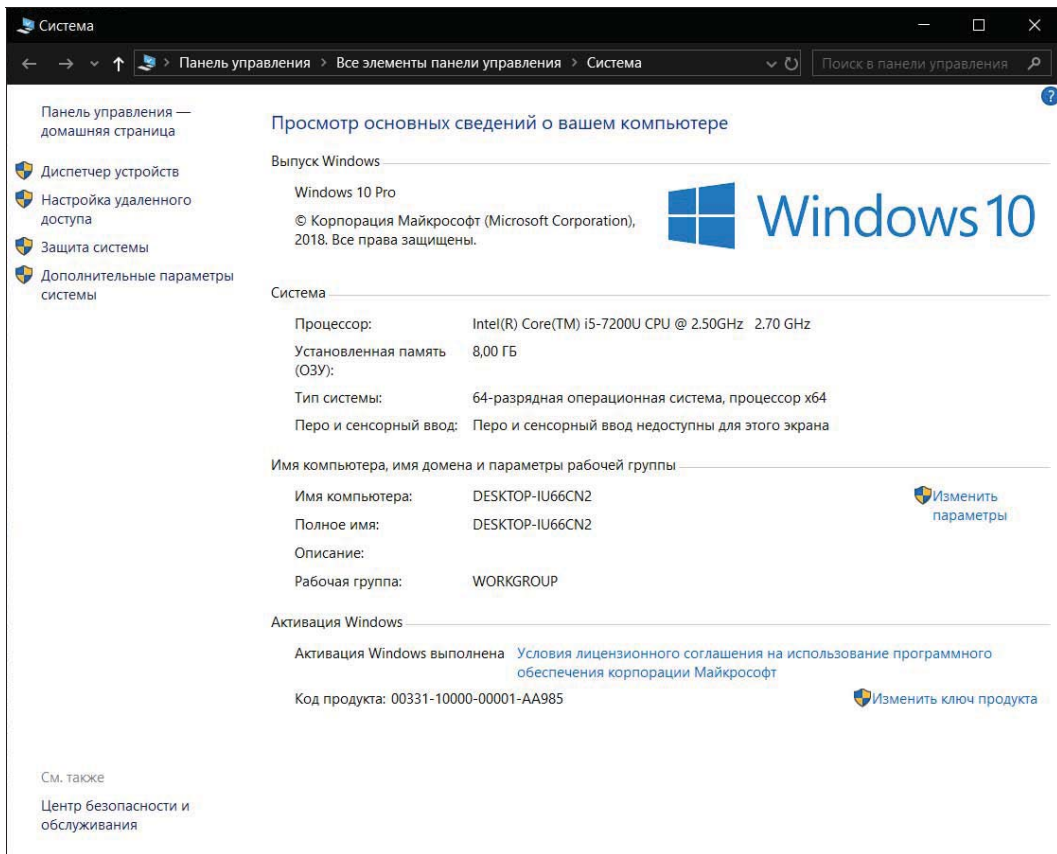


Рис. 1.10. Окно Система

3. Нажмите кнопку **Переменные среды**.
4. В окне **Переменные среды** выберите переменную **Path** и нажмите кнопку **Изменить** (рис. 1.11).
5. Добавьте каталог C:\php-8.2.4 в путь поиска программы (рис. 1.12).
6. Закройте последние три окна, последовательно нажимая кнопки **ОК**.
7. Закройте окно **Система**.

Откройте любой файловый менеджер (подойдет и Проводник, но удобнее использовать приложение вроде Total Commander). Перейдите в каталог C:\php-8.2.4 и

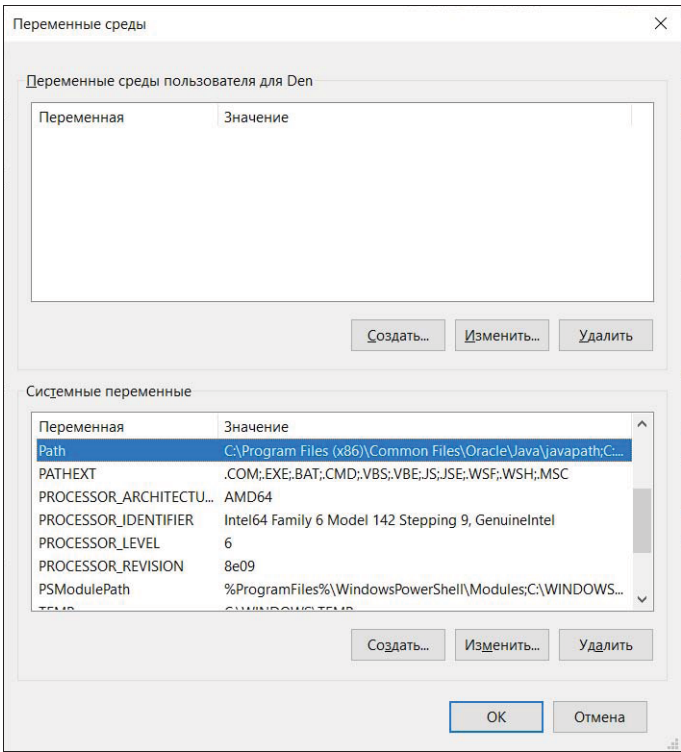


Рис. 1.11. Переменные среды

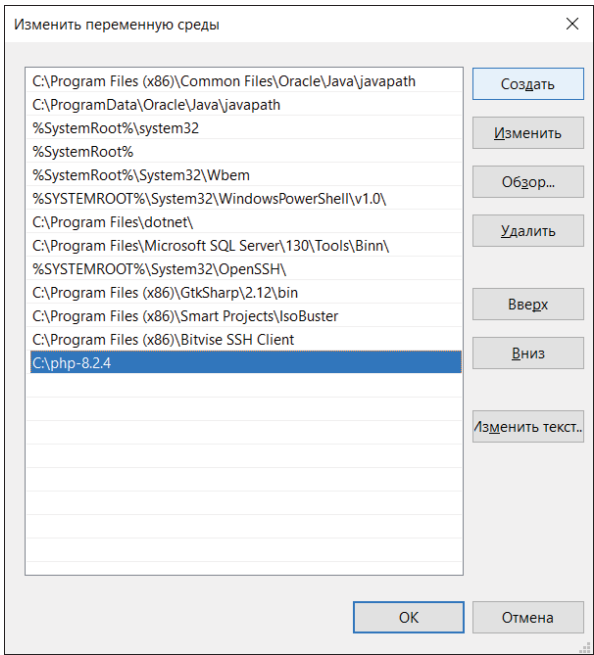


Рис. 1.12. Редактирование переменной Path

сохраните файл `php.ini-development` как `php.ini`. Откройте файл `php.ini` и найдите в нем строку:

```
; extension_dir = "ext"
```

Эту строку нужно раскомментировать, чтобы получилось так:

```
extension_dir = "ext"
```

Сохраните файл `php.ini` с произведенным изменением.

#### ПРИМЕЧАНИЕ

В дистрибутиве вы найдете две заготовки файла `php.ini`: `php.ini-development` и `php.ini-production`. Для разработки и отладки кода лучше использовать настройки для разработчика, т. е. файл `php.ini-development`. Файл `php.ini-production` содержит так называемые *производственные параметры*. Разработчику от них мало пользы, т. к. сообщения об ошибках выводиться не будут.

Теперь проверим, все ли вы сделали правильно. Откройте командную строку и введите команду `php -v`. Если вы увидите сообщение с номером версии РНР и другой информацией (рис. 1.13), тогда вы все сделали как надо — т. е. добавили каталог с РНР в путь поиска программ и создали файл `php.ini` с основными настройками.

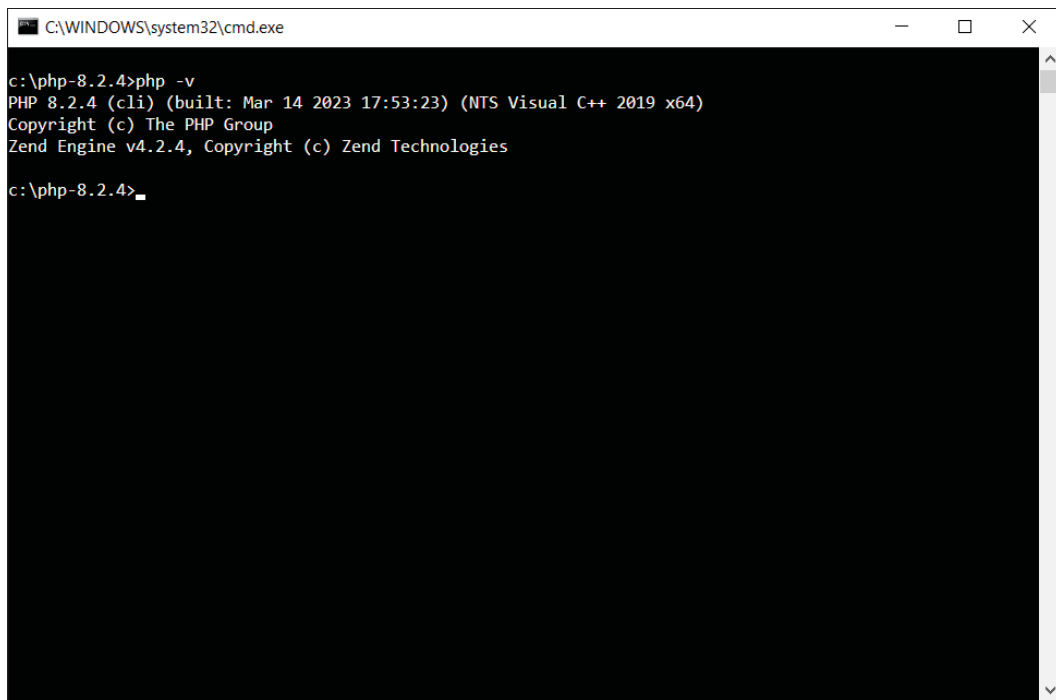


Рис. 1.13. РНР 8 установлен и работает

Однако если на вашем компьютере не установлен распространяемый пакет Visual C++ для Visual Studio 2015, вы увидите совсем другое сообщение (рис. 1.14). К счастью, исправить ситуацию достаточно просто — нужно лишь загрузить (с адреса

<http://www.microsoft.com/ru-RU/download/details.aspx?id=48145>) и установить недостающее ПО. Правда, появление такого сообщения возможно лишь для довольно древних систем, которые давно не обновлялись.

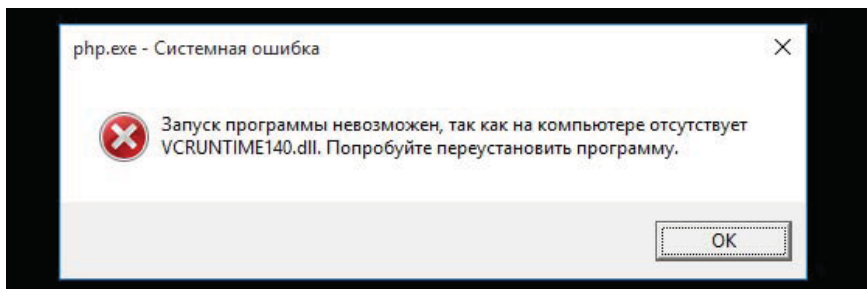


Рис. 1.14. Для запуска PHP 8 не хватает нужного DLL-файла

Установив PHP, нужно связать его с Apache. Раньше все эти операции делал инсталлятор, сейчас, повторяю, приходится выполнять их вручную.

Перейдите в каталог, куда вы установили Apache (в нашем случае — это C:\Apache24\), затем — в подкаталог conf. Откройте файл конфигурации Apache — httpd.conf. Найдите в этом файле секцию LoadModule. Каждая строка LoadModule загружает какой-то модуль Apache. Пролистайте файл, пока не закончатся строки LoadModule. После последней строки LoadModule добавьте следующие строки:

```
LoadModule php_module "c:/php-8.2.4/php8apache2_4.dll"  
AddType application/x-httpd-php .php
```

Обратите внимание на пути — возможно, они у вас будут другими, если вы распаковали PHP 8 в иной каталог.

После этого найдите в файле httpd.conf следующие строки:

```
<IfModule dir_module>  
DirectoryIndex index.html  
</IfModule>
```

Добавьте сюда поддержку файла index.php:

```
<IfModule dir_module>  
DirectoryIndex index.php index.html  
</IfModule>
```

На этом все... Файл конфигурации Apache можно сохранить. Далее с помощью приложения Apache Monitor, которое устанавливается вместе с Apache, перезапустите Apache, выбрав команду **Restart**.

В каталоге C:\Apache24\htdocs создайте файл info.php со следующим содержимым:

```
<?php phpinfo(); ?>
```

Откройте браузер и введите URL:

<http://localhost/info.php>

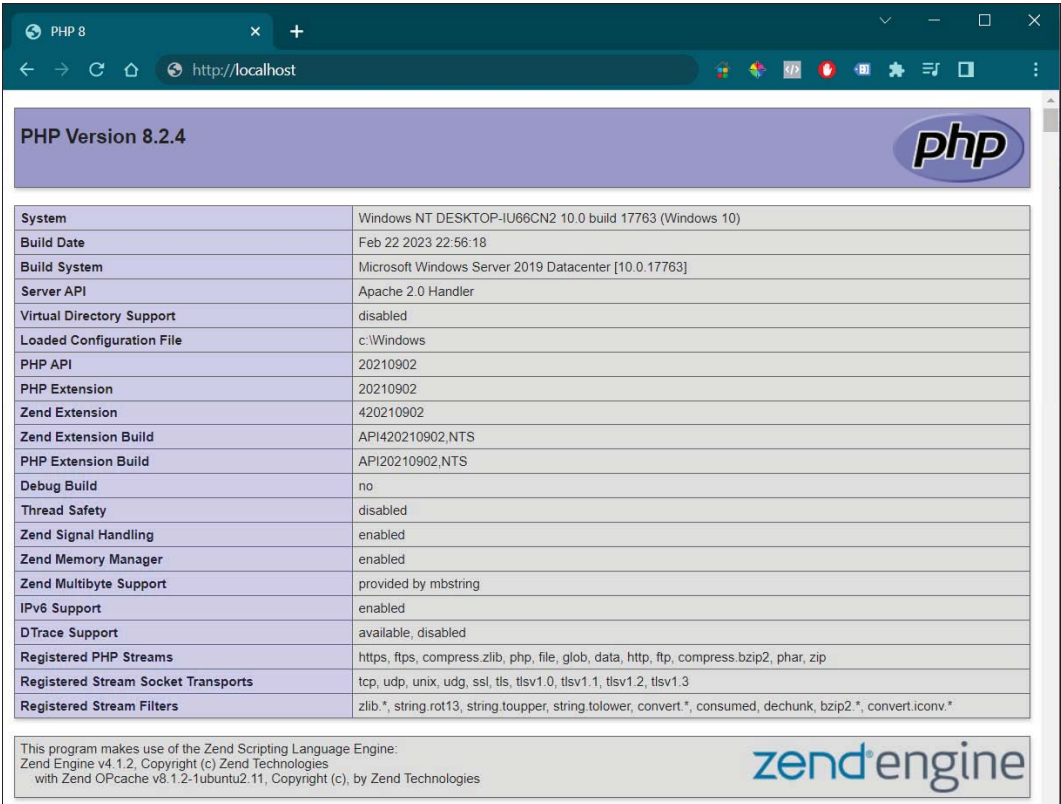


Рис. 1.15. Информационная страница

Если вы все сделали правильно, то увидите информационную страницу, сообщающую о конфигурации PHP (рис. 1.15).

Если же вы вместо информационной таблицы увидели содержимое файла info.php, то PHP не работает вместе с Apache. Чтобы исправить это, выполните следующие действия:

1. Перезапустите Apache (ранее было показано, как это сделать) и повторите попытку. Возможно, вы просто забыли перезапустить веб-сервер. Если же перезапуск сервера не дал результатов, перейдите к следующему шагу.
2. Ранее было показано, какие строки нужно было добавить в файл httpd.conf. Убедитесь, что вы все сделали без ошибок, и снова перезапустите веб-сервер. Если и сейчас у вас ничего не получилось, скачайте и установите XAMPP (см. далее) — в 99.99% у вас не будет никаких проблем, и вы сразу получите все необходимое программное обеспечение, готовое к использованию.

Теперь вам осталось скачать и установить сервер баз данных MySQL (<http://mysql.com/>). Его установка проходит вообще без каких-либо нюансов, поэтому ее мы здесь подробно рассматривать не станем.

На мой взгляд, установка в Windows всех компонентов, необходимых для организации связки Apache + PHP + MySQL, в настоящее время напоминает установку

такой связки в Linux лет 10 лет назад. Сейчас же в Linux, наоборот, все стало гораздо проще — там достаточно только установить необходимые пакеты (apache, php, mysql) — об этом мы поговорим в *главе 37*, когда будем устанавливать все эти пакеты на VDS под управлением Linux.

### 1.3.2. Установка для новичков (рекомендуется)

Если вам не с руки устанавливать все компоненты связки отдельно, вы можете установить пакет XAMPP, содержащий Apache, PHP и MySQL, как говорится, «в одном флаконе». Скачать последнюю версию XAMPP можно по адресу <https://www.apachefriends.org/>.

#### ПРИМЕЧАНИЕ

В каталоге *software* сопровождающего книгу электронного архива (см. *приложение 4*) вы найдете последнюю (на момент написания этих строк) версию XAMPP (файл *xampp-windows-x64-8.2.0-0-VS16-installer*). Это версия, содержащая самое новое ПО, включая PHP 8.2.0.

Основное преимущество XAMPP — простота установки и управления всеми сервисами, входящими в его состав (веб-сервером, FTP-сервером и MySQL).

Установка XAMPP проблем не вызывает. Программа нормально работает в актуальных версиях Windows: 10 и 11. Программу установки XAMPP нужно запускать от имени администратора.

Желательно устанавливать XAMPP в корневой каталог диска C:\, а не в каталог C:\Program Files, чтобы не возникло каких-либо проблем с правами доступа, — так рекомендуют разработчики XAMPP. Да и вам добираться до своих файлов будет проще (рис. 1.16).

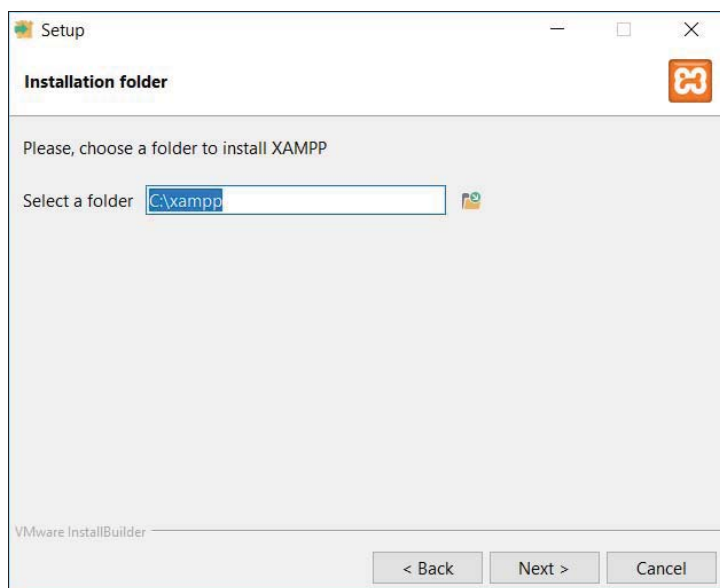


Рис. 1.16. Установка XAMPP

При установке Apache и MySQL вас спросят, как вы хотите запускать эти программы: в качестве сервисов (служб) Windows или вручную. Если вы собираетесь программировать на PHP каждый день, тогда удобнее запускать Apache и MySQL как сервисы. Но помните, что, даже когда они вам не нужны, они все равно занимают память и потребляют ресурсы процессора. Поэтому, если вы собираетесь программировать на PHP время от времени, лучше запускать эти программы вручную. Для этого запустите приложение XAMPP Control Panel — откроется панель управления XAMPP. Она позволяет запустить, остановить и настроить подконтрольные сервисы (рис. 1.17):

- ❑ **Apache** — это и есть веб-сервер;
- ❑ **MySQL** — сервер баз данных;
- ❑ **FileZilla** — простейший FTP-сервер для Windows;
- ❑ **Mercury** — простейший почтовый сервер (точнее, агент доставки почты — MTA) для Windows;
- ❑ **Tomcat** — программа-контейнер сервлетов, написанная на языке Java и реализующая спецификацию сервлетов и спецификацию JavaServer Pages (JSP).

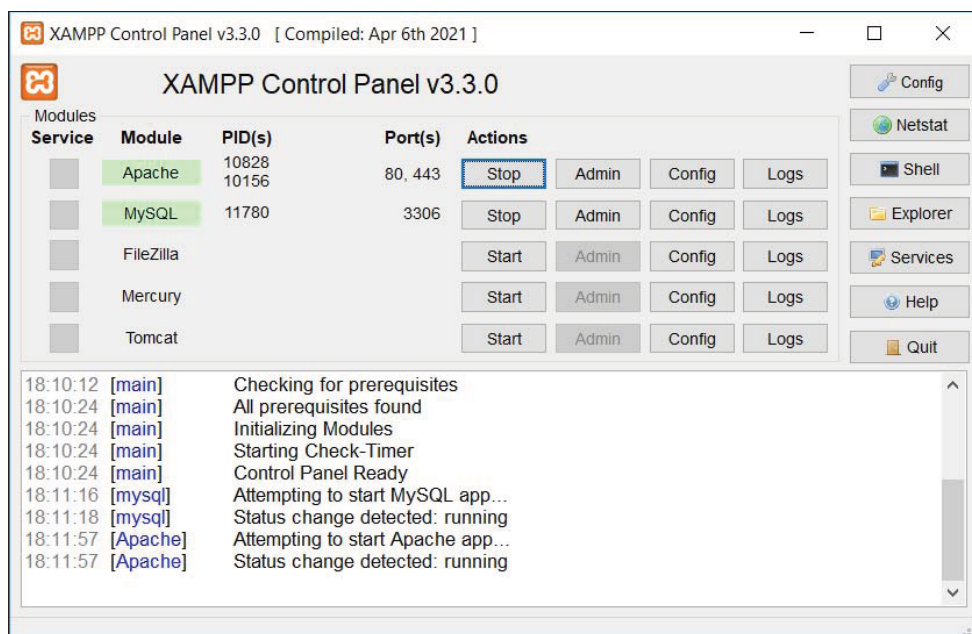


Рис. 1.17. Панель управления XAMPP

Все сервисы запускать не надо — вам хватит первых двух. Третий сервис нужно запустить, если вы хотите написать PHP-сценарий, взаимодействующий с FTP-сервером, — вам не придется искать какой-нибудь FTP-сервер, все будет у вас под рукой. Система **Mercury** подойдет для проектов, подразумевающих отправку поч-

ты, — без запуска Mercury ваши проекты не смогут отправлять почту стандартной функцией `mail()`, или же вам придется использовать дополнительные классы вроде `PHPMailer` для отправки почты через сторонние SMTP-серверы.

Для запуска сервиса нажмите кнопку **Start** напротив его имени. В случае успешного запуска кнопка **Start** превратится в кнопку **Stop** (служит для останова сервиса), а между названием сервиса и кнопкой **Stop** появится надпись **Running**.

При первом запуске каждой из служб вы увидите окно брандмауэра, предлагающее разрешить или запретить приложению доступ в сеть (рис. 1.18), — установите флажок **Общественные сети** и нажмите кнопку **Разрешить доступ**.

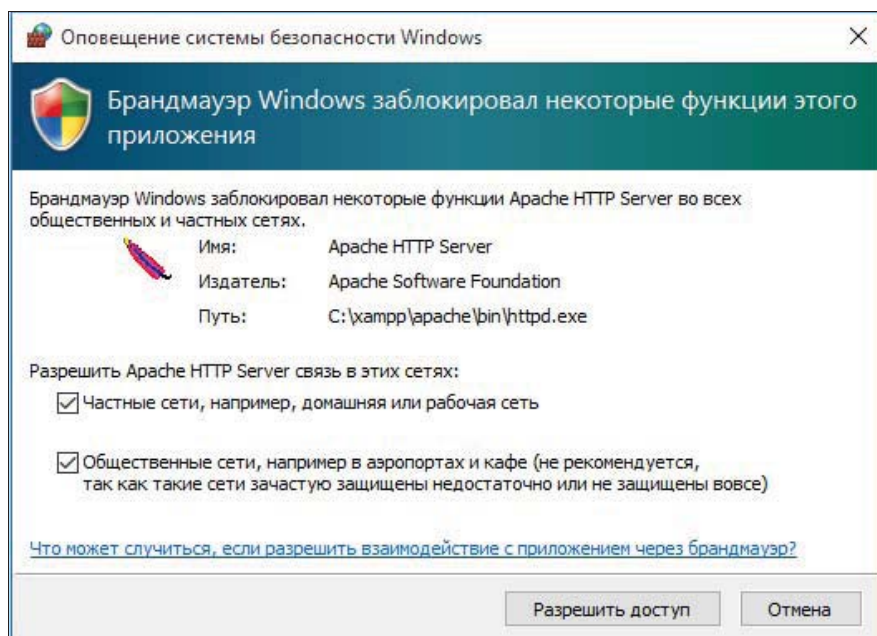


Рис. 1.18. Оповещение системы безопасности Windows

Кнопка **Admin**, которая становится активной после запуска сервиса, позволяет настраивать выбранный сервис. Так, нажатие кнопки **Admin** для сервиса **MySQL** приводит к вызову программы `phpMyAdmin`, которая предоставляет удобный веб-интерфейс для управления вашими базами данных.

Для выхода из панели управления XAMPP служит кнопка **Exit** (см. рис. 1.17). Помните, что при выходе из панели запущенные сервисы не будут остановлены. Если вы хотите завершить работу сервисов, нажмите кнопку **Stop** каждого запущенного сервиса, а затем уже нажимайте кнопку **Exit**.

Свои PHP-сценарии при использовании XAMPP вам нужно поместить в подкаталог `htdocs` каталога установки XAMPP. Например, если вы установили XAMPP в каталог `C:\xampp`, то PHP-сценарии следует поместить в каталог `C:\xampp\htdocs`.

При написании сценариев, использующих MySQL, вы можете применить следующие параметры доступа:

- ☐ сервер MySQL — localhost;
- ☐ имя пользователя — root;
- ☐ пароль — отсутствует;
- ☐ база данных по умолчанию — test (но вы можете создать другую БД с помощью программы phpMyAdmin).

Вот теперь можно перейти ко *второй главе*, в которой вы напишете свою первую программу.



## ГЛАВА 2

# Программа на PHP

### 2.1. Ваша первая программа

Традиционно многие книги по программированию начинаются с создания простейшей программы, выводящей строку **Hello world!**. В этой книге такой программы не будет. Вместо этого мы напишем программу, тестирующую наше MySQL-соединение. Этим мы сразу «убьем двух зайцев»: напишем первую PHP-программу и проверим созданную в *главе 1* конфигурацию.

Логика нашей будущей программы проста — если удалось подключиться к MySQL-серверу, то все настроено правильно, иначе будет выведено сообщение об ошибке. В последующих главах книги будут приведены более сложные примеры работы с MySQL.

Итак, приступим к написанию нашего первого PHP-сценария (листинг 2.1). На момент выполнения этой программы доступ root к базе данных должен быть разрешен, пароля у root — нет, а база данных называется `test`. Именно таким образом выглядят параметры подключения к MySQL по умолчанию. Если вы изменили эти параметры, то придется внести изменения в программу — задать название базы данных, имя пользователя и его пароль. Даже если вы это не сделаете, программа все равно будет работать, просто она выведет строку **Подключение не удалось**.

#### Листинг 2.1. Первая PHP-программа

```
<?php
// параметры доступа к MySQL, возможно, придется изменить:
$dsn = "mysql:host=localhost;dbname=test;charset=utf8";

try {
    // имя пользователя root, пароль не задан
    $pdo = new PDO($dsn, "root", "");
} catch (PDOException $e) {
    die('Подключение не удалось: ' . $e->getMessage());
}
```

```
/* это пример многострочного
   комментария */

# пример комментария в стиле UNIX bash:
/* если программа не остановлена die(), тогда выводим
   сообщение об успешном подключении */

echo "Соединение установлено успешно";
?>
```

Разберемся, что мы узнали, написав эту программу:

- ❑ PHP-код заключается в теги `<?php` и `?>`. Закрывающий тег `?>` в последнее время принято опускать, если нет прямой необходимости его закрыть, — например, когда вы встраиваете PHP-код внутри HTML-кода и хотите явно указать, что блок PHP-кода завершен. В ранних версиях PHP (до версии 7) использовался короткий синтаксис: `<? и ?>`, в последних версиях PHP он не поддерживается (если вы явно не отредактируете файл конфигурации для его включения), и вы получите сообщение об ошибке;
- ❑ инструкция `die()` выводит текст ошибки и останавливает выполнение сценария, когда нам это нужно (в нашем случае, если не получилось подключиться к MySQL-серверу);
- ❑ комментарии в PHP-программах могут быть трех типов. Лично мне удобнее использовать однострочные комментарии в стиле C: `// Комментарий`;
- ❑ для вывода текста служит инструкция `echo`;
- ❑ после каждого оператора нужно указывать точку с запятой.

### О КОРОТКИХ PHP-ТЕГАХ

Теги вида `<? ?>` называются *короткими*, или *short tags* (полная версия тегов выглядит так: `<?php ?>`). Чтобы можно было использовать сокращенную версию тегов, в файле `php.ini` нужно включить директиву `short_open_tag` (присвоить ей значение `On`). По умолчанию директива `short_open_tag` выключена (ее значение: `Off`). Некоторые хостеры включают эту директиву, другие — нет. В XAMPP она выключена по умолчанию. Старайтесь привыкать к полной версии PHP-тегов — так ваши сценарии станут более универсальными. В этой книге мы иногда будем использовать полную версию, иногда — сокращенную. Но не удивляйтесь, если вдруг сценарии, использующие сокращенную версию тегов, не станут работать — нужно или включить упомянутую директиву, или использовать полную версию тегов.

## 2.2. Запуск PHP-программы

Запустить программу очень просто. Сохраните ее как PHP-файл (например, `test.php`) и загрузите на ваш веб-сервер (предположим, в его корневой каталог). После чего откройте браузер и введите URL:

`http://сервер/test.php`

**ВНИМАНИЕ!**

Файл *test.php* нужно загрузить именно в корневой каталог веб-сервера, а не в корневой каталог вашей учетной записи на FTP-сервере хостинг-провайдера. Обычно это каталог *public\_html* или *www*. За более точной информацией обратитесь к вашему хостинг-провайдеру.

Если у вас локальный веб-сервер, то запустить программу можно так:

`http://localhost/test.php`

**ВНИМАНИЕ!**

Если вы настраивали локальный веб-сервер так, как показано в *главе 1*, тогда файл *test.php* вам нужно поместить или в каталог *C:\Apache24\htdocs* (при настройке вручную), или в каталог *c:\xampp\htdocs* (при использовании XAMPP).

Также можно запустить нашу программу и без веб-сервера. Достаточно открыть терминал (командную строку в Windows), перейти в каталог, в котором находится наш файл *test.php*, и ввести команду:

```
php test.php
```

Если вы при запуске сценария из листинга 2.1 увидели сообщение об ошибке, проверьте параметры доступа к MySQL: имя сервера, имя пользователя и пароль. В большинстве случаев проблема именно в них. Если вы сами устанавливали сервер MySQL, то нужно использовать следующие параметры доступа:

- ☐ имя сервера — `localhost`;
- ☐ имя пользователя — `root`;
- ☐ пароль — пустой (отсутствует).

Если же вы запускаете сценарий на сервере хостинг-провайдера, то параметры доступа будут другими.

## 2.3. Вывод текста без *echo*

Создайте на основании листинга 2.2 файл *html.php*.

**Листинг 2.2. Сценарий *html.php* (вариант 1)**

```
<html>
<head><title>Зароловок</title></head>
<body>
```

Как видите, этот сценарий ничем не отличается от PHP-кода. Попробуйте его запустить — PHP не найдет PHP-кода в файле (поскольку в нем нет тегов `<?php` и `?>`) и выведет HTML-код «как есть». Но самое интересное далее — PHP-сценарии могут содержать как HTML-код, так и PHP-код. Модифицируем наш файл *html.php* (листинг 2.3).

**Листинг 2.3. Сценарий html.php (вариант 2)**

```
<html>
<head><title>Заголовок</title></head>
<body>
<?php
echo "<h1>Выводим текст</h1>";
?>
</body>
</html>
```

В результате выполнения этого сценария будет сформирована следующая HTML-страница (рис. 2.1).

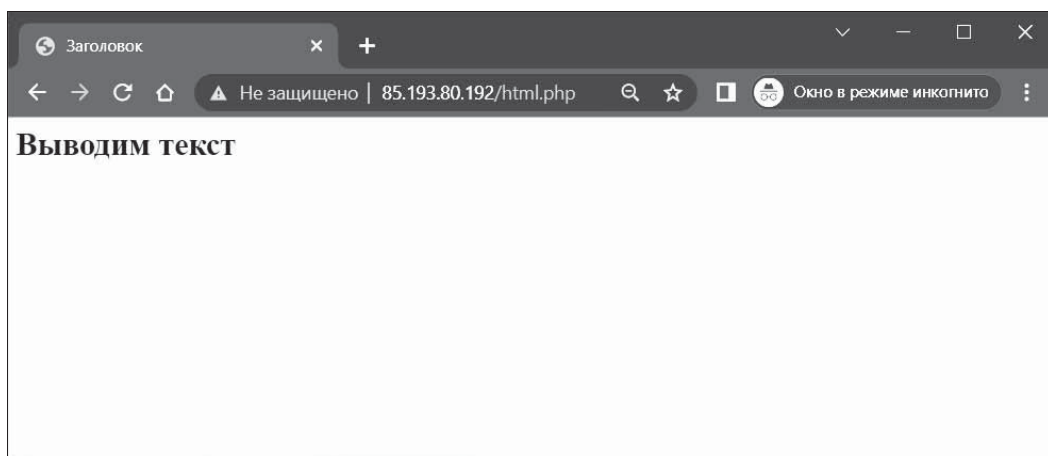


Рис. 2.1. Результат выполнения листинга 2.3

Пойдем дальше. Вы можете включать в HTML-код неограниченное количество PHP-кода (листинг 2.4).

**Листинг 2.4. Сценарий html.php (вариант 3)**

```
<html>
<head><title>Заголовок</title></head>
<body>
<?php
echo "<h1>Выводим текст</h1>";
?>
</body>
<?php
echo "</html>";
?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *2-4.php*), представленный в каталоге *Glava\_2* сопровождающего книгу электронного архива (см. приложение 4).

Приведенные сценарии *html.php* демонстрируют, как можно включать HTML-код в состав PHP-сценариев. Но я вам советую избегать подобной практики: *HTML-код и PHP-код должны храниться отдельно*. Ведь когда созданный сайт будет передан заказчику, его станет поддерживать веб-дизайнер, который может не иметь представления о PHP и легко допустить ошибку, из-за которой сценарий вообще не запустится. Поэтому лучше всего хранить HTML-код в шаблонах, заполняемых нашим PHP-сценарием или специальным шаблонизатором. Работа с шаблонами будет описана в главах 22 и 23. В главе 22 мы рассмотрим простенький шаблонизатор, который подойдет для самых простых проектов. В главе 23 мы обратимся уже к более серьезному продукту — шаблонизатору Blade, который используется фреймворком Laravel.

Забегая вперед, скажу, что HTML-код за пределами PHP-тегов таит в себе угрозу. Если до PHP-тегов будет обнаружен любой текст (даже пробелы или пробельные символы), то он будет выведен в браузер до выполнения самого кода, а потом PHP-сценарий не сможет установить файлы cookies, поскольку они должны быть установлены до первого вывода в браузер.



## ГЛАВА 3

# Основы синтаксиса РНР

### 3.1. Переменные

#### 3.1.1. Правила объявления переменных. Имена переменных

Если вы ранее работали с другими языками программирования, то наверняка знакомы с понятием *переменной*, которое присутствует и в РНР. Приводя формальное определение, можно сказать, что *переменная* — это *поименованная область памяти*. Области памяти присваивается выбранное программистом имя, по которому потом программа обращается к данным. Все данные, с которыми вы станете работать, будут храниться в переменных.

В других языках программирования переменные нужно объявлять до их первого использования — например, в языке Pascal так:

```
var  
    I : integer;
```

РНР не требует явного объявления переменных. Но если переменные не объявлены, как тогда РНР узнает, что есть переменная, а что — обычный символ? Для выделения переменных из кода программы РНР требует, чтобы перед именем переменной указывался символ `$`. Например: `$i`, `$string`.

То, что РНР не требует объявления переменной, не освобождает вас от ее *инициализации*, т. е. от задания переменной начального значения. Инициализация не обязательна, но весьма желательна. Предположим, вы используете переменную `$i` как счетчик с шагом 2, т. е. после определенного события — например, после каждой итерации цикла, выполняется следующий оператор:

```
$i = $i + 2;
```

Если вы забыли присвоить переменной `$i` начальное значение до выполнения этого оператора, результат будет непредсказуемым. Поэтому возьмите за правило инициализировать каждую переменную, которую вы собираетесь использовать, например, так:

```
$i = 0;
```

Вернемся к именам переменных. Как уже было сказано, перед именем переменной нужно обязательно указывать символ доллара. Например:

```
i = $i + 2;           // этот код неправильный!  
i = i + 2;           // этот код неправильный!  
$i = $i + 2;         // этот код ПРАВИЛЬНЫЙ
```

Имя переменной может состоять из латинских символов, знака подчеркивания и цифр, причем не может начинаться с цифры.

Имена переменных чувствительны к регистру символов, т. е. `$a` и `$A` — две разные переменные:

```
$a = 1;  
$A = 2;  
  
echo $a;  
echo $A;
```

Наш код выведет, как и следовало ожидать, **12**. Если бы переменные не были чувствительны к регистру символов, код вывел бы **11**.

### 3.1.2. Типы данных переменных

Обратите внимание, что при объявлении переменной на языке Pascal были указаны имя переменной и ее тип. PHP не требует явного объявления типа переменной, но это не означает, что в PHP нет типов переменных. Просто PHP по-особому работает с переменными.

Предположим, вы объявили две переменные. Одна переменная (`$a`) содержит число, а вторая переменная (`$b`) — массив. Присвоим второй переменной значение первой переменной:

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

Вы только задумайтесь, что происходит. Вы присваиваете число массиву! В любом другом языке программирования вы бы получили ошибку несовместимости типов переменных, но не в PHP. PHP возьмет да и скопирует в переменную `$b` не только значение переменной `$a`, но и тип (структуру) переменной. Да, в результате массив будет потерян, а `$b` станет обычной целой переменной, содержащей значение переменной `$a`.

Такая «всеядность» PHP весьма удобна, но, с другой стороны, если вы проявите невнимательность, она может стать причиной трудно отслеживаемых ошибок.

Вот основные типы переменных в PHP:

- ❑ `int` — *целой* переменной в 32-битных системах вы можете присвоить значения от `-2 147 483 648` до `2 147 483 647`, в 64-битных: от `-9 223 372 036 854 775 808` до `9 223 372 036 854 775 807` (что соответствует `long long int` в C);
- ❑ `double` — переменная, содержащая *вещественное число большой точности*;
- ❑ `string` — наверное, самый важный тип данных в PHP, поскольку очень много операций связано именно с обработкой *строк*;

- ❑ `array` — массивам и спискам далее посвящен целый раздел книги;
- ❑ `object` — PHP является объектно-ориентированным языком программирования, что, впрочем, требуется далеко не всегда — вы можете написать превосходный сценарий, работающий и без объектов;
- ❑ `bool` — логическая переменная: может принимать значения `true` или `false`;
- ❑ `link` — ссылка, особый тип, о котором мы поговорим отдельно.

Для проверки типа переменной служат функции, приведенные в табл. 3.1.

Таблица 3.1. Функции проверки типа переменной

Функция	Описание
<code>is_int(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — целое число
<code>is_double(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — вещественное число
<code>is_string(\$x)</code>	Проверяет, является ли переменная строкой
<code>is_numeric(\$x)</code>	Проверяет, является ли переменная числом. Возвращает <code>true</code> , если это так
<code>is_array(\$x)</code>	Проверяет, является ли переменная массивом
<code>is_object(\$x)</code>	Возвращает <code>true</code> , если указанная переменная — объект
<code>is_bool(\$x)</code>	Проверяет, является ли переменная логической переменной
<code>gettype(\$x)</code>	Возвращает строку, которая описывает тип переменной: boolean — логическое значение; integer — целое значение; double — вещественное значение; string — строка; array — массив; object — объект; resource — ресурс (например, результат обработки MySQL-запроса); NULL — переменная не имеет значения (не инициализирована), следовательно, невозможно установить ее тип; unknown type — неизвестный тип (тип определить не удалось)

**ВНИМАНИЕ!**

Начиная с PHP 7, строки, содержащие шестнадцатеричные числа, теперь всегда обрабатываются как строки, а не как числа: `is_numeric("0xFF")` теперь `false`.

В PHP 8 улучшено сравнение строк и чисел. Например, такое положение дел было в PHP 7:

```
0 == 'foobar'           // true
```

А вот в PHP 8 это выражение вернет `false`, поскольку при сравнении с числовой строкой PHP 8 использует сравнение чисел. В противном случае число преобразуется в строку и используется сравнение строк: строка `'0'` не равна строке `'foobar'`.

### 3.1.3. Булевы переменные

Отдельного разговора заслуживают логические переменные. Мы привыкли, что в других языках программирования значения `true` и `false` ассоциируются соответственно со значениями 1 и 0. Но в PHP значение `false` ассоциируется с пустой строкой, а `true` — с любым ненулевым значением, например:

```
$a = 10001;  
if ($a == true) echo 'Переменная $a — истинна';
```

#### ПРИМЕЧАНИЕ

Обратите внимание, что в операторе `echo` используются одинарные кавычки, а не двойные.

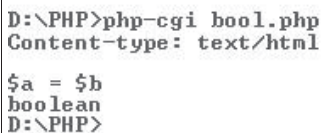
Поскольку со значением `true` ассоциируется любое ненулевое значение, то мы увидим следующую строку:

Переменная \$a — истинна

Рассмотрим еще один пример:

```
$a = 1001; $b = true;  
if ($a == $b) echo '$a = $b';           // сравниваем переменные  
echo "\n".gettype($b);                  // выводим тип переменной $b
```

Самое интересное здесь — сообщение сценария, а он сообщит, что переменные `$a` и `$b` равны (рис. 3.1). Это происходит потому, что переменной `$b` присвоен логический тип (это так и есть), и PHP считает, что и переменная `$a` тоже логического типа.



```
D:\PHP>php-cgi bool.php  
Content-type: text/html  
  
$a = $b  
boolean  
D:\PHP>
```

Рис. 3.1. Сравнение логической и целой переменных

### 3.1.4. Операции над переменными

Над переменной вы можете выполнить всего три операции:

- присвоить переменной значение: константу, результат выполнения функции или выражения:

```
$x = 1;  
$x = pi;  
$x = func($y);  
$x = ($b * 2) / 100;
```

□ уничтожить переменную — для этого служит функция `unset`:

```
unset($a);
```

□ проверить существование переменной — используется функция `isset()`:

```
if ($isset($x)) echo 'Переменная $x установлена';
```

### 3.1.5. Ссылки

Если вы знакомы с программированием на языке C, то, очевидно, знакомы с *указателями*. В PHP нет указателей, зато есть *ссылки*. Ссылки можно воспринимать как псевдонимы переменных. Предположим, что у вас есть переменная-массив `$arr1` объемом 1 Мбайт. Если вы присвоите ее значение другой переменной, `$arr2`, то у вас окажутся два одинаковых массива общим размером 2 Мбайт.

Ссылки позволяют обращаться к одним и тем же данным под разными именами:

```
$a = 100;
$link_on_a = &$a;

echo $link_on_a;

echo " ";      // пробел

$link_on_a = 5;
echo $a;
```

Сначала мы объявляем переменную `$a`. Затем объявляем ссылку `$link_on_a` на переменную `$a`. Синтаксис объявления ссылки следующий:

```
ссылка = &переменная;
```

После чего выводим переменную `$link_on_a`. Она равна 100, как и следовало ожидать. Теперь мы присваиваем ссылке `$link_on_a` значение 5 и выводим значение переменной `$a`. Сценарий выведет значение 5 (рис. 3.2).

```
D:\PHP>php-cgi links.php
Content-type: text/html

100 5
D:\PHP>
```

Рис. 3.2. Пример использования ссылок

Показанные здесь ссылки называются *жесткими*. Существуют еще *мягкие* (символические) ссылки. Символическая ссылка — это не псевдоним переменной, а переменная, содержащая имя другой переменной.

Работать с мягкими ссылками нужно следующим образом. Сначала надо создать переменную, на которую будет указывать ссылка. Затем другой переменной присвоить имя первой переменной. Чтобы создать мягкую ссылку, следует обратиться ко второй переменной так:

```
$$переменная
```

Рассмотрим следующий фрагмент кода:

```
$a = 100; // исходная переменная

$soft_link = "a"; // эта переменная будет играть роль ссылки

echo $$soft_link; // это уже ссылка, выведет 100

echo " "; // выводим пробел

echo $soft_link; // выведет просто 'a' (без кавычек)

$$soft_link = 5; // присваиваем ссылке значение 5

echo " "; // еще один пробел

echo $a; // выводим переменную $a, будет выведено 5
```

## 3.2. Константы

*Константы* содержат постоянные значения, изменить которые в процессе выполнения программы вы не можете. Для объявления констант используется функция `define()`, синтаксис ее таков:

```
define(имя, значение, чувствительность_к_регистру);
```

Вот примеры объявления констант:

```
define("const", "константа", true);
define("A", "1");
```

Первый параметр функции `define()` — имя константы, второй — значение. Третий параметр особый. Если он `true`, имя константы будет чувствительно к регистру. По умолчанию константы чувствительны к регистру, т. е. `CONST` и `const` — это две разные константы.

При использовании константы указывать знак доллара не нужно:

```
// выводим константы
echo const;
echo A;
```

В PHP 7 в качестве значения констант, объявляемых через функцию `define()`, теперь можно указывать массивы. Как видите, PHP стал еще более гибким языком.

В табл. 3.2 приведены предопределенные константы PHP.

Таблица 3.2. Предопределенные константы PHP

Константа	Значение
PHP_VERSION (string)	Текущая версия PHP в виде строки в формате major.minor.release[extra]

Таблица 3.2 (продолжение)

Константа	Значение
<code>PHP_MAJOR_VERSION</code> (integer)	Основная версия PHP, то есть 5 для версии 5.4.21
<code>PHP_MINOR_VERSION</code> (integer)	Промежуточная версия, то есть 4 для версии 5.4.21
<code>PHP_RELEASE_VERSION</code> (integer)	Номер релиза, то есть число 21 для версии 5.4.21
<code>PHP_VERSION_ID</code> (integer)	Числовой идентификатор версии PHP
<code>PHP_EXTRA_VERSION</code> (string)	Экстраверсия PHP, например, слово <code>-extra</code> для версии 5.2.7-extra
<code>PHP_ZTS</code>	Показывает, включена ли ZTS (Zend Thread Safety), — содержит значение 0, если ZFS выключена, и 1, если включена
<code>PHP_DEBUG</code>	Если константа содержит значение 1, то исполнимый файл интерпретатора PHP был собран в отладочной конфигурации. Может произойти, если вы «собирали» PHP самостоятельно. У всех загруженных с официальных сайтов версий PHP эта константа равна 0
<code>PHP_MAXPATHLEN</code> (integer)	Максимальная длина файловых имен (включая путь), поддерживаемая этой сборкой PHP
<code>PHP_OS</code> (string)	Тип операционной системы, под управлением которой работает PHP
<code>PHP_SAPI</code> (string)	Серверное API этой сборки PHP. Смотрите также <code>php_sapi_name()</code>
<code>PHP_EOL</code> (string)	Корректный символ конца строки, используемый на этой платформе
<code>PHP_INT_MAX</code> (integer)	Максимальное целое число, поддерживаемое этой сборкой PHP. Обычно это <code>int(2147483647)</code> для 32-битных систем и <code>int(9223372036854775807)</code> для 64-битных
<code>PHP_INT_MIN</code> (integer)	Наименьшее целое число, поддерживаемое этой сборкой PHP. Обычно это <code>int(-2147483648)</code> в 32-битных системах и <code>int(-9223372036854775808)</code> в 64-битных. Константа доступна, начиная с PHP 7.0
<code>PHP_INT_SIZE</code> (integer)	Размер типа <code>integer</code> в байтах в этой сборке PHP
<code>DEFAULT_INCLUDE_PATH</code> (string)	Include-путь по умолчанию (путь, по которому инструкции <code>include</code> и <code>require</code> будут искать PHP-файлы)
<code>PHP_EXTENSION_DIR</code> (string)	Каталог с расширениями PHP
<code>PHP_PREFIX</code> (string)	Значение опции <code>--prefix</code> , указанной при запуске <code>configure</code>
<code>PHP_BINDIR</code> (string)	Указывает путь установки бинарных файлов
<code>PHP_BINARY</code> (string)	Указывает путь к исполняемым файлам PHP во время выполнения скрипта
<code>PHP_MANDIR</code> (string)	Указывает путь установки страниц документации <code>man</code> (только для Linux)

Таблица 3.2 (продолжение)

Константа	Значение
PHP_LIBDIR (string)	Каталог библиотек
PHP_DATADIR (string)	Каталог с данными
PHP_SYSCONFDIR (string)	Каталог с файлом конфигурации
PHP_CONFIG_FILE_PATH (string)	Путь к файлу конфигурации
PHP_SHLIB_SUFFIX (string)	Суффикс, используемый для динамически линкуемых библиотек, таких как <code>so</code> (характерный для большинства UNIX-систем) или <code>dll</code> (для Windows)
<b>Обработка ошибок</b>	
E_ERROR (integer)	Фатальные ошибки времени выполнения. Это неустранимые средствами самого скрипта ошибки — такие как ошибка распределения памяти и т. п. Выполнение скрипта в таком случае прекращается
E_WARNING (integer)	Предупреждения времени выполнения (нефатальные ошибки). Выполнение скрипта в таком случае не прекращается
E_PARSE (integer)	Ошибки на этапе компиляции. Должны генерироваться только парсером
E_NOTICE (integer)	Уведомления времени выполнения. Указывают на то, что во время выполнения скрипта произошло что-то, что может указывать на ошибку, хотя это может происходить и при обычном выполнении программы
E_CORE_ERROR (integer)	Фатальные ошибки, которые происходят во время запуска PHP. Такие ошибки схожи с E_ERROR, за исключением того, что они генерируются ядром PHP
E_CORE_WARNING (integer)	Предупреждения (нефатальные ошибки), которые происходят во время начального запуска PHP. Такие предупреждения схожи с E_WARNING, за исключением того, что они генерируются ядром PHP
E_COMPILE_ERROR (integer)	Фатальные ошибки на этапе компиляции. Такие ошибки схожи с E_ERROR, за исключением того, что они генерируются скриптовым движком Zend
E_COMPILE_WARNING (integer)	Предупреждения на этапе компиляции (нефатальные ошибки). Такие предупреждения схожи с E_WARNING, за исключением того, что они генерируются скриптовым движком Zend
E_USER_ERROR (integer)	Сообщения об ошибках, сгенерированные пользователем. Такие ошибки схожи с E_ERROR, за исключением того, что они генерируются в коде скрипта средствами функции PHP <code>trigger_error()</code>
E_USER_WARNING (integer)	Предупреждения, сгенерированные пользователем. Такие предупреждения схожи с E_WARNING, за исключением того, что они генерируются в коде скрипта средствами функции PHP <code>trigger_error()</code>

Таблица 3.2 (окончание)

Константа	Значение
<code>E_USER_NOTICE (integer)</code>	Уведомления, сгенерированные пользователем. Такие уведомления схожи с <code>E_NOTICE</code> , за исключением того, что они генерируются в коде скрипта средствами функции PHP <code>trigger_error()</code>
<code>E_STRICT (integer)</code>	Включаются для того, чтобы PHP предлагал изменения в коде, которые обеспечат лучшее взаимодействие и совместимость кода
<code>E_RECOVERABLE_ERROR (integer)</code>	Фатальные ошибки с возможностью обработки. Такие ошибки указывают, что, вероятно, возникла опасная ситуация, но при этом скриптовый движок остается в стабильном состоянии. Если такая ошибка не обрабатывается функцией, определенной пользователем для обработки ошибок, выполнение приложения прерывается, как происходит при ошибках <code>E_ERROR</code>
<code>E_DEPRECATED (integer)</code>	Уведомления времени выполнения об использовании устаревших конструкций. Включаются для того, чтобы получать предупреждения о коде, который не будет работать в следующих версиях PHP
<code>E_USER_DEPRECATED (integer)</code>	Уведомления времени выполнения об использовании устаревших конструкций, сгенерированные пользователем. Такие уведомления схожи с <code>E_DEPRECATED</code> за исключением того, что они генерируются в коде скрипта с помощью функции PHP <code>trigger_error()</code>
<code>E_ALL (integer)</code>	Все поддерживаемые ошибки и предупреждения, за исключением ошибок <code>E_STRICT</code> до PHP 5.4.0
<code>__FILE__</code>	Имя текущего сценария (до и после слова <code>FILE</code> — по два знака подчеркивания)
<code>__DIR__</code>	Каталог, в котором находится сценарий
<code>__LINE__</code>	Строка, обрабатываемая в настоящий момент

Константы обработки ошибок можно использовать в файле конфигурации `php.ini`, а также в функции `error_reporting()`, которая устанавливает уровень отображаемых ошибок, например:

```
// Выключаем отображение всех ошибок:
error_reporting(0);
// Показываем только обычные ошибки времени выполнения:
error_reporting(E_ERROR | E_WARNING | E_PARSE);
// Показываем все ошибки:
error_reporting(E_ALL);
```

Обычно при отладке сценариев первой строкой (после `<?php`) добавляется вызов функции `error_reporting()`. Однако эту функцию можно также использовать, чтобы подавить любой вывод, что будет полезно, если ваш сценарий работает с сессией или файлами `cookies`. Как известно, `cookies` могут быть установлены только до

первого вывода из сценария. И если ваш сценарий выведет хотя бы один байт (например, уведомление интерпретатора), секция HTTP-заголовков будет закрыта, и сценарий больше не сможет устанавливать cookies (об установке cookies и работе с сессиями мы поговорим в *главе 25*).

Ознакомиться с другими PHP-константами можно по адресу:

**<http://php.net/manual/en/reserved.constants.php>.**

Просмотреть список констант и их значений можно с помощью функции `get_defined_constants()`:

```
print_r(get_defined_constants());
```

## 3.3. Выражения и операции

### 3.3.1. Что такое *выражение*?

*Выражение* — это часть инструкции или сама инструкция, возвращающая значение. Практически все инструкции программы являются выражениями — даже оператор присваивания:

```
$a = 1;
```

Результат этой операции — 1, он будет присвоен переменной `$a`. Само выражение в этом случае выглядит так:

```
1
```

Выражением может быть не только значение, но и инициализированная переменная:

```
$a = 1;  
$b = $a;
```

В PHP оператор присваивания может выглядеть довольно необычно:

```
$a = $b = 1;
```

Интересно, что в выражении можно даже инициализировать переменную:

```
$a = 50 * ($c = 7) * $c;
```

### 3.3.2. Арифметические операции

Как и в любом другом языке программирования, вы можете выполнять следующие арифметические операции:

- ☐ + — сложение;
- ☐ - — вычитание;
- ☐ \* — умножение;
- ☐ / — деление;
- ☐ A % B — остаток от деления A на B.

Кроме того, можно использовать операции инкремента и декремента:

- `$a++` — увеличить переменную `$a` на 1;
- `$b--` — уменьшить переменную `$b` на 1.

### 3.3.3. Логические выражения

*Логическим* называется выражение, результатом которого является истина (`true`) или ложь (`false`). Обычно в логических выражениях применяются следующие операторы сравнения:

- `<` — меньше;
- `>` — больше;
- `!=` — не равно;
- `==` — равно;
- `<=` — меньше или равно;
- `>=` — больше или равно;
- `<=>` — оператор `spaceship` («космический корабль»).

Особого внимания заслуживает последний оператор, появившийся в PHP 7. Выражение:

```
$a <=> $b
```

возвращает `-1`, `0` или `+1`, если `$a` соответственно меньше, равно или больше `$b`. Как уже отмечалось в *предисловии*, оператор `<=>` очень удобно использовать в `callback` для функции пользовательской сортировки `usort()`.

Также вы можете использовать следующие сугубо логические операции:

- `&&` — бинарная операция И (AND) истинна, если оба операнда истинны;
- `||` — бинарная операция ИЛИ (OR) истинна, если один из операндов `true`;
- `!` — унарная операция отрицания (NOT).

Примеры использования логических операций:

```
$a = true;
$b = false;
$c = $a || $b;      // $c = true;
$d = $a && $b;      // $d = false;
$e = !$a;           // $e = false;
```

### 3.3.4. Битовые операции

Эти операции предназначены для установки или снятия групп битов целочисленной переменной. Любое число можно рассматривать как последовательность битов. Целые числа в PHP являются 32-разрядными, поэтому для представления одного числа используются 32 бита:

0000 0000 0000 0000 0000 0000 0000 0000 – это 0  
0000 0000 0000 0000 0000 0000 0000 0001 – это 1  
0000 0000 0000 0000 0000 0000 0000 0010 – это 2  
0000 0000 0000 0000 0000 0000 0000 0011 – это 3  
...

К числу битовых операций относятся следующие:

- ☐ `a & b` — будут установлены те биты, которые установлены в `a` и `b` одновременно:  
`a = 1111 (15) b = 1100 (12) Результат = 1100 (12)`
- ☐ `a | b` — будут установлены только те биты, которые были установлены или в `a`, или в `b`:  
`a = 0100 (4) b = 0101 (5) Результат = 0101 (5)`
- ☐ `~a` — инвертирование битов:  
`a = 1001 Результат = 0110`
- ☐ `a << b` — поразрядный сдвиг битов `a` влево на `b` разрядов:  
`a = 0010 b = 0001 (1 разряд) Результат = 0100`
- ☐ `a >> b` — поразрядный сдвиг битов `a` вправо на `b` разрядов:  
`a = 0010 b = 0001 (1 разряд) Результат = 0001`

В PHP 7 оператор левого побитового сдвига (`<<`) при сдвиге на количество битов, превышающее количество битов в `integer`, теперь всегда возвращает 0. До этого результат зависел от архитектуры процессора. Аналогичный правый сдвиг всегда дает 0 или -1 в зависимости от знака исходного числа. При этом сдвиг не влияет на старший бит, отвечающий за знак.

### 3.3.5. Приоритеты операций

Операции в PHP выполняются в соответствии с их приоритетом, приведенным в табл. 3.3. Для изменения порядка выполнения операций можно использовать скобки, например:

`$a = (2 + 2) * 2;`

Таблица 3.3. Приоритет операций

Приоритет	Порядок выполнения	Операторы
13	справа налево	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> , <code>&gt;&gt;=</code> , <code>&lt;&lt;=</code> , <code>&amp;=</code> , <code>^=</code> , <code> =</code>
12	слева направо	<code>  </code>
11	слева направо	<code>&amp;&amp;</code>
10	слева направо	<code> </code>
9	слева направо	<code>^</code>

Таблица 3.3 (окончание)

Приоритет	Порядок выполнения	Операторы
8	слева направо	&
7	слева направо	== != === !== <> <=>
6	слева направо	<, <=, >, >=
5	слева направо	<<, >>
4	слева направо	+, -
3	слева направо	*, /, %
2	справа налево	++(префикс), --(префикс)
1	слева направо	(постфикс)++, (постфикс)--

**ПРИМЕЧАНИЕ**

Приоритет операций в табл. 3.3 представлен от низшего к высшему.

**ПРИМЕЧАНИЕ**

Операции >> и << называются *битовыми*. Мы их не рассматриваем, поскольку при веб-разработке они используются очень редко, а на роль системного языка программирования (как C) PHP явно не тянет!

3.3.6. Операторы эквивалентности == и ===

Для сравнения значений служит оператор равенства ==, например:

```
if ($a == $b) ...
```

Иногда этот оператор работает не совсем так, как мы ожидаем. Рассмотрим следующий код:

```
$x = 1;
$y = 1;

$a = 0;
$b = "";

if ($x == $y) echo 'Переменные $x и $y равны';
if ($a == $b) echo 'Переменные $a и $b равны';
```

Переменные \$x и \$y действительно равны, поэтому мы увидим сообщение:

```
Переменные $x и $y равны
```

Но переменные \$a и \$b вообще не могут быть равны, они даже разных типов, однако PHP сообщает, что они равны:

```
Переменные $a и $b равны
```

Почему так происходит? Просто PHP воспринимает переменные `$a` и `$b` как логические. А `0` и пустая строка соответствуют `false`, поэтому PHP и сообщает, что переменные равны.

Чтобы избежать подобной неразберихи, нужно использовать оператор сравнения `===` (три знака равенства):

```
if ($a === $b) echo 'Переменные $a и $b равны';
```

В этом случае строка 'Переменные `$a` и `$b` равны' не будет выведена.

Помните, что для оператора `===` не предусмотрен обратный оператор `!==`, вместо него вы должны использовать следующую конструкцию:

```
if (!($a === $b)) echo '$a <> $b';
```

### 3.3.7. Оператор `??` — сокращенная форма тернарной условной операции

Конструкция `?` : существует не только в PHP, но и в других языках программирования (правда, в немного видоизмененной форме). Она называется *тернарной условной операцией*. В PHP она имеет вид:

```
o1 ? o2 : o3
```

Работает она так: если `o1` истинно, тогда выполняется `o2`, иначе `o3`.

В PHP 7 появился новый оператор `??` (Null coalescing operator), позволяющий проверить переменную на существование и вернуть ее значение либо значение по умолчанию. Рассмотрим небольшой пример. Ранее нам приходилось писать вот такие длинные конструкции:

```
$action = isset($_POST['action']) ? $_POST['action'] : 'index';
```

Такой код можно часто встретить при обработке POST-переменных: если POST-переменная `action` (очевидно, она определяет действие, которое должна выполнить страница) задана, тогда локальная переменная `$action` получает ее значение. В противном случае переменная `$action` получит значение `'index'`.

Благодаря новому оператору эту конструкцию можно записать так:

```
$action = $_POST['action'] ?? 'index';
```

### 3.3.8. Операции со строками

Прежде всего нужно поговорить о *кавычках*, в которые заключаются строковые значения:

```
$a = 10;  
$string1 = "Значение переменной: $a";  
$string2 = 'Значение переменной: $a';
```

После выполнения этих операторов значение переменной `string1` будет следующим:

```
Значение переменной: 10
```

А значение переменной `string2` таким:

Значение переменной: `$a`

Как видите, между двойными и одинарными кавычками большая разница. Если строка заключена в одинарные кавычки, то все символы строки воспринимаются как есть, кроме последовательностей `\'` и `\\` (первая последовательность означает апостроф, а вторая — слеш). Строка, заключенная в двойные кавычки, позволяет вывести значение переменной.

В строках вы можете использовать следующие специальные символы:

- ❑ `\n` — символ новой строки;
- ❑ `\t` — символ табуляции;
- ❑ `\r` — символ возврата каретки;
- ❑ `\$` — знак доллара;
- ❑ `\"` — кавычка;
- ❑ `\\` — обратный слеш;
- ❑ `\xNN` — символ с кодом `NN` (в шестнадцатеричной системе);
- ❑ `\u{xxxxxx}` — произвольный Unicode-символ, где `xxxxxx` — код символа.

#### ПРИМЕЧАНИЕ

Синтаксис `\u{xxxxxx}` появился только в PHP 7. В PHP 5 он работать не будет.

Операций над строками всего две: обращение к символу строки и конкатенация (слияние строк). Обратиться к определенному символу строки можно так:

```
$имя[номер]
```

Например:

```
$string[1]           // обращение к первому символу строки
```

Конкатенация (слияние) строк выполняется с помощью оператора `.`:

```
$s1 = "Hello, ";  
$s2 = "world!";  
$string = $s1 . $s2; // $string = "Hello, world!"
```

При желании вы можете считать в строку вывод системной команды. Делается это с помощью обратных апострофов (которые находятся на одной клавише с тильдой `~`):

```
$dir = `dir`; // в Windows  
$dir = `ls`;  // в Linux  
echo $dir;
```

### 3.3.9. Оператор *nullsafe*

В PHP 8 разработчики решили избавить своих коллег от головной боли, а именно — от постоянной проверки на `null`. Представим, что нам нужно получить страну из сессии пользователя. Сначала надо убедиться, есть ли сама сессия, затем прове-

речь, есть ли в ней переменная `$user`, затем вернуть адрес методом `getAddress()` и, если возвращенный адрес не равен `null`, тогда выделить из него страну и вернуть ее. Код на PHP 7 выглядел бы так:

```
$country = null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```

А на PHP 8 все помещается в одну строчку:

```
$country = $session?->user?->getAddress()?->country;
```

Вместо проверки на `null`, вы можете использовать последовательность вызовов с новым оператором `nullsafe`. Когда один из элементов в последовательности возвращает `null`, выполнение прерывается, и вся последовательность возвращает `null`.

## 3.4. Условный оператор

Полный вид синтаксиса *условного оператора* в PHP выглядит следующим образом:

```
if (условие1)
    оператор1;
elseif (условие2)
    оператор2;
...
elseif (условиеN)
    операторN;
else
    операторN+1;
```

Условие — это логическое выражение. Как работает такой оператор, думаю, понятно. Если истинно первое условие, то выполняется `оператор1`, и действие условного оператора прекращается. Если истинно `условие2`, то выполняется `оператор2` и т. д. Если ни одно условие не истинно, тогда выполняется `операторN+1`.

Полная форма оператора используется далеко не всегда. Чаще она сокращается за счет блоков `elseif` до следующего вида:

```
if (условие1)
    оператор1;
else
    оператор2;
```

Минимальная форма условного оператора выглядит следующим образом:

```
if (условие1)
    оператор1;
```

Рассмотрим небольшой пример использования условного оператора:

```
$a = 10;
$b = 7;

if ($a > $b)
    echo "A > B";
else
    echo "A < B";
```

## 3.5. Циклы

PHP поддерживает четыре вида циклов:

- ❑ цикл со счетчиком `for`;
- ❑ цикл с предусловием `while`;
- ❑ цикл с постусловием `do-while`;
- ❑ специальный цикл переработки массивов `foreach`.

В этой главе мы рассмотрим первые три цикла, а о последнем цикле речь пойдет в *главе 9*.

### 3.5.1. Цикл со счетчиком *for*

Цикл со счетчиком `for` нужно использовать тогда, когда известно количество итераций цикла. Синтаксис цикла `for` следующий:

```
for (оператор_инициализация; условие; поститерационная_команда) {
    тело_цикла;
}
```

Перед первой итерацией выполняется оператор инициализации — обычно это инициализация счетчика цикла.

Затем проверяется условие. Если оно истинно, выполняется тело цикла, а после него — поститерационная команда.

Рассмотрим небольшой пример цикла `for`:

```
for ($i=1; $i <= 5; $i++) echo $i;
```

Цикл выведет: **12345**.

Если нужно выполнить несколько команд до запуска цикла и после итерации, их можно разделить запятыми:

```
for ($i=1, $d = 1; $i <= 5; $i++, $t--) echo $i;
```

### 3.5.2. Цикл *while*

Цикл `while` еще называется *циклом с предусловием*, поскольку сначала вычисляется условие, а потом выполняются операторы (тело цикла). Рассмотрим синтаксис цикла `while`:

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

Сначала вычисляется условие цикла. Если оно истинно, то выполняется тело цикла. Вот пример цикла `while`:

```
$i = 1;  
while ($i<6) {  
    echo $i;  
    $i++;  
}
```

Как и в предыдущем случае, цикл напечатает **12345**.

### 3.5.3. Цикл *do-while*

В отличие от цикла `while`, цикл `do-while` сначала выполняет тело цикла, а потом проверяет условие. Вот синтаксис цикла `do-while`:

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

Особенность этого цикла заключается в том, что тело цикла будет выполнено хотя бы один раз.

Рассмотрим небольшой пример:

```
$i = 1;  
do {  
    echo $i;  
    $i++;  
} while ($i <= 5);
```

Как вы уже догадались, наш цикл снова выведет строку **12345**.

### 3.5.4. Принудительное завершение цикла и пропуск итерации

Предположим, что у нас есть некая функция `func()`. Давайте напишем цикл, который будет вызывать эту функцию некоторое количество раз. Если возвращенное функцией `func()` значение будет равно 5, тогда нам нужно завершить цикл. Если же `func()` возвратит 4, то нам нужно пропустить итерацию, т. е. ничего не делать, и сразу перейти к следующей итерации цикла:

```
$i = 1;
while ($i < 10) {

    $k = func();

    if ($k == 4) continue;      // условие перехода на следующую итерацию
    if ($k == 5) break;        // условие останова цикла

    echo $k;                   // выводим $k

}
```

Оператор `break` принудительно завершает работу цикла. Обратите внимание на условие нашего цикла — он должен выполняться, пока `$i` меньше 10. Поскольку переменная `$i` равна 1 и не изменяется в процессе выполнения цикла, то наш цикл выполнялся бы вечно, если бы не условие останова цикла и оператор `break`.

Оператор `continue` завершает не весь цикл, а только текущую итерацию, и переходит к следующей итерации цикла.

## 3.6. Оператор выбора *switch-case*

Условный оператор довольно удобен, но когда условий слишком много, и все они одного типа, тогда намного удобнее использовать оператор выбора `switch-case`. Рассмотрим синтаксис этого оператора:

```
switch (выражение) {
    case значение1: операторы1; [break;]
    case значение2: операторы2; [break;]
    . . .
    case значениеN: операторыN; [break;]
    [default: операторы_по_умолчанию; [break;]]
}
```

Разберемся, как работает оператор `switch-case`. Сначала вычисляется значение выражения. Затем просматривается список значений. Если найдено совпадение, то выполняются соответствующие значению операторы. Если после этих операторов не будет указан оператор `break`, будут выполнены все последующие операторы и операторы по умолчанию.

Если совпадений не найдено, будут выполнены операторы по умолчанию, если они были заданы.

Теперь рассмотрим два примера кода. Первый — это пример обычного условного оператора `if`:

```
$x=2;

if ($x == 0) {
    echo "x=0";
} elseif ($x == 1) {
    echo "x=1";
}
```

```
} elseif ($x == 2) {  
    echo "x=2";  
}
```

Второй пример — это аналогичный ему оператор `switch-case`:

```
$x=2;  
switch ($x) {  
case 0:  
    echo "x=0";  
    break;  
case 1:  
    echo "x=1";  
    break;  
case 2:  
    echo "x=2";  
    break;  
}
```

## 3.7. Выражение *match* в PHP 8

Новое выражение `match` похоже на оператор `switch` со следующими особенностями:

- ☐ `match` — это выражение, его результат может быть сохранен в переменной или возвращен;
- ☐ условия `match` поддерживают только однострочные выражения, для которых не требуется управляющая конструкция `break`;
- ☐ выражение `match` использует строгое сравнение.

Посмотрим сначала на оператор `switch` в PHP 7:

```
switch (8.0) {  
    case '8.0':  
        $result = "О нет!";  
        break;  
    case 8.0:  
        $result = "То, что я и ожидал";  
        break;  
}  
echo $result;  
//> О нет!
```

А теперь — на выражение `match` в PHP 8:

```
echo match (8.0) {  
    '8.0' => "О нет!",  
    8.0 => "То, что я и ожидал",  
};  
//> То, что я и ожидал
```



## ГЛАВА 4

# Файл конфигурации *php.ini*

### 4.1. Каталог */etc/php*

Если у вас есть возможность редактировать файл `php.ini`, вам повезло — вы сможете настроить PHP так, как нужно вам. Обычно этот файл находится в одном из подкаталогов каталога `/etc/php`. В некоторых старых версиях PHP каталог конфигурации может называться `/etc/php5` или `/etc/php7`, но в современных версиях принято хранить конфигурацию PHP в каталоге `/etc/php/<номер версии>`, например `/etc/php/8.1`.

Перейдите в этот каталог. В нем вы найдете несколько подкаталогов, в которых хранятся разные конфигурации PHP — на все случаи жизни. Так, в каталоге `/etc/php/8.1/apache2` находится конфигурация PHP, когда он выполняется под управлением Apache. Проще говоря, когда сценарий запускается через браузер. Когда же сценарий запускается из командной строки, то используется конфигурация из каталога `/etc/php/8.1/cli`. В подкаталоге `fpm` хранится конфигурация на случай, если PHP выполняется как менеджер процессов PHP-FPM (FastCGI Process Manager). Такая конфигурация задействуется, когда в качестве веб-сервера используется Nginx, а не Apache.

Важно понимать, какой файл конфигурации вам нужно редактировать. Представим, что мы работаем с версией PHP 8.1, и рассмотрим примеры:

- ❑ используется веб-сервер Apache, и вам нужно увеличить количество памяти, выделяемой для сценария. Тогда вам надо редактировать файл `/etc/php/8.1/apache2/php.ini`. Чтобы конфигурация вступила в силу, следует перезагрузить Apache: `sudo systemctl restart apache2`;
- ❑ используется nginx — конфигурация находится в файле `/etc/php/8.1/fpm/php.ini`. После редактирования этого файла, чтобы конфигурация вступила в силу, нужно перезапустить PHP-FPM командой: `sudo systemctl restart php-fpm`;
- ❑ нужно отредактировать конфигурацию для консольных скриптов — редактируйте файл `/etc/php/8.1/cli/php.ini`. Никаких действий для применения конфигурации выполнять не требуется.

Несколько конфигураций используются для удобства и безопасности. Например, для консольных скриптов есть смысл не ограничивать время выполнения и разре-

шить выполнение некоторых функций (ту же `system()`, например), которые запрещено выполнять в режиме веб.

Далее мы рассмотрим самые полезные настройки PHP.

## 4.2. Параметры памяти

Директива `memory_limit` позволяет указать, сколько памяти будет доступно каждому PHP-процессу. Значение по умолчанию — 128 Мбайт, но этого размера оперативной памяти хватит только для приложений среднего размера. При превышении этого размера вы увидите сообщение о фатальной ошибке. Примерно такое:

```
Fatal error: Allowed memory size of 134217728 bytes exhausted  
(tried to allocate 8388608 bytes) in /<путь>/parser_02.php on line 114
```

Какое значение установить? Если вы планируете запускать самые простые приложения, то будет достаточно 64–128 Мбайт. Можно даже остановиться на 64 Мбайт, чтобы сэкономить ресурсы.

Для запуска приложений среднего размера будет достаточно 256–512 Мбайт. Но не стоит забывать об общем размере оперативной памяти. Пусть вы арендуете VDS с 2 Гбайт оперативной памяти. В этом случае я бы не рекомендовал использовать для `memory_limit` значения, превышающие 256 Мбайт. Ведь кроме PHP на вашем VDS запускаются и другие процессы — тот же сервер баз данных (MySQL), веб-сервер (Apache или nginx), да и для самой системы тоже нужна будет оперативка.

Как определить, сколько памяти потребляет один PHP-процесс? Если у вас есть доступ к командной строке, вы можете использовать утилиты `top` или `htop`, чтобы узнать это. В одном из терминалов откройте `top` или `htop`, а в другом запустите ваш сценарий (или же запустите его через браузер).

Если доступа к командной строке нет, вы можете использовать функцию `memory_get_peak_usage()` в конце PHP-сценария. Вот только проблема: если вы столкнетесь с фатальной ошибкой из-за превышения лимита памяти, до выполнения этой функции интерпретатор не дойдет, и результат вы не получите. Обычно PHP-сценарий потребляет 5–20 Мбайт данных, но при работе с загрузкой файлов, картинками или с большими объемами данных это значение будет выше.

Большое потребление памяти может также свидетельствовать о некорректной работе самого сценария. Если он работает с базой данных, возможно, есть смысл перепроверить запросы и объемы возвращаемых данных. Например, в таблице может быть много строк, и сценарий будет пытаться получить сразу все строки таблицы. Такое поведение не является правильным — для более эффективной работы сценария нужно получать строки из таблицы частями (`LIMIT`).

## 4.3. Zend OPcache

Разобравшись с памятью, нужно настроить расширение PHP Zend OPcache. В PHP встроен движок кеширования — OPcache, который сохраняет в памяти прекомпилированный байт-код скрипта.

Что такое *байт-код*? Давайте сначала узнаем, как обычный PHP-сценарий обрабатывается для каждого HTTP-запроса. Первым делом веб-сервер (Apache или nginx) отправляет запрос в PHP-FPM, а затем PHP-FPM назначает запрос дочернему PHP-процессу. Этот дочерний PHP-процесс находит соответствующий PHP-сценарий, читает его и компилирует его в так называемый *байт-код*, после чего происходит выполнение этого байт-кода.

Обратите внимание: PHP выполняет не тот текст, который вы пишете. Он сначала преобразует его в другой формат — байт-код, а потом уже выполняет этот самый байт-код. Тут просматривается аналогия с обычными компиляторами — написав программу на C, вы передаете ее компилятору, который создает исполнимый файл (\*.exe в Windows), и уже его вы можете запустить на выполнение. Аналогичная ситуация и в PHP — сначала создается байт-код, который потом запускается на выполнение.

Запустившись, байт-код генерирует HTTP-ответ, который потом возвращается веб-серверу, а тот, в свою очередь, отправляет его клиенту (в браузер пользователя). Такой процесс происходит для каждого HTTP-запроса.

Мы можем несколько ускорить этот процесс — с помощью кеширования откомпилированного байт-кода для каждого PHP-сценария. В результате мы добьемся, что движок PHP будет сразу выполнять байт-код, а не заниматься поиском сценария, его парсингом и преобразованием в байт-код.

Обратите внимание: OPcache — это механизм кеширования байт-кода, а не HTML. Сейчас поясню, в чем разница. Представим, что у нас есть сценарий `page.php`, который генерирует страницу на основании переданного параметра `id`, — например: `page.php?id=101`.

Пользователь обратился к сценарию так: `page.php?id=101`. Его ответ был прокеширован прокси-сервером (пусть тем же Squid) и помещен в кеш. Если этот или другой пользователь обратится к `page.php?id=101`, он, скорее всего, получит уже ранее прокешированную версию.

Но что если пользователь обратится к `page.php?id=102`? Ведь сценарий тот же, но параметр другой, и HTML-код для таких условий не был прокеширован ранее. Значит, придется выполнять `page.php` снова. Но если у нас используется OPcache, то байт-код сценария `page.php` уже создан — ведь ранее пользователь запускал этот сценарий, но с другим параметром. Этот байт-код и будет запущен на выполнение.

Расширение OPcache встроено в PHP, начиная с версии 5.5. Рассмотрим его опции:

```
opcache.memory_consumption = 64
opcache.interned_strings_buffer = 16
opcache.max_accelerated_files = 4000
opcache.validate_timestamps = 1
opcache.revalidate_freq = 0
opcache.fast_shutdown = 1
```

Первая опция (`opcache.memory_consumption`) говорит, сколько памяти в мегабайтах будет выделено под кеш байт-кода. Объем в 64 Мбайт достаточно для приложений

среднего размера. Если у вас большое приложение, состоящее из множества РНР-сценариев, можно увеличить это значение, скажем, до 128 Мбайт. Если же у вас небольшое приложение со всего несколькими сценариями, этот размер можно уменьшить до 16 Мбайт.

Вторая опция (`opcache.interned_strings_buffer`) задает размер памяти (в мегабайтах), используемой для хранения *интернированных строк*. Что это такое? Интерпретатор РНР обнаруживает идентичные строки в вашем приложении и сохраняет каждый отдельный экземпляр строки всего один раз, а во всех остальных случаях используются указатели на ту область памяти, где хранится оригинальная строка. По умолчанию буфер интернированных строк — отдельный для каждого РНР-процесса. Эта настройка позволяет всем процессам пула РНР-FPM хранить интернированные строки в одном разделяемом буфере, что дает возможность существенно экономить память. Значение по умолчанию — 4 Мбайт, но я рекомендую увеличить это значение до 128 Мбайт.

Третья опция (`opcache.max_accelerated_files`) задает максимальное число РНР-сценариев, которое может быть сохранено в кеше байт-кода. Вы можете указать любое значение: от 200 до 100 000. Убедитесь, что это значение больше количества сценариев в вашем приложении. Например, если в вашем приложении 3500 сценариев, то можно установить значение 4000.

Четвертая опция (`opcache.validate_timestamps`) заставляет РНР проверять РНР-сценарии на наличие изменений в интервале времени, указанном настройкой пятой опции (`opcache.revalidate_freq`). При разработке рекомендуется установить `opcache.validate_timestamps` в 1, а на продакш-серверах — в 0, т. е. отключить такую проверку. Это означает, что РНР не станет производить проверку изменений и вам придется очищать кеш байт-кода вручную, иначе РНР будет выполнять уже откомпилированный байт-код, и вы не увидите внесенных изменений.

Пятая опция (`opcache.revalidate_freq`) задает, как часто (в секундах) РНР будет проверять откомпилированные РНР-файлы на наличие изменений. Если РНР обнаружит изменения, он перекомпилирует РНР-файл. Значение 0 означает, что РНР должен перекомпилировать файлы при каждом запросе, но при условии, что включена четвертая опция (`opcache.validate_timestamps`).

Шестая опция (`opcache.fast_shutdown`) просит OPcache использовать быструю последовательность завершения работы для скорейшего освобождения памяти. Официальная документация по этой опции отсутствует. Все, что вам нужно знать, — она должна быть включена.

Для очистки кеша (если вы внесли изменения в сценарий, но видите, что они не применились) нужно перезапустить сервис РНР-FPM или веб-сервер Apache.

## 4.4. Максимальное время выполнения

Директива `max_execution_time` задает, сколько времени (в секундах) может выполняться РНР-сценарий, прежде чем он будет принудительно завершен. Значение по умолчанию — 30 секунд, но для некоторых приложений его нужно увеличить, по-

сколько сценарий может быть завершен прежде, чем он выполнит необходимые операции.

При запуске из командной строки `max_execution_time` по умолчанию равно 0, т. е. сценарий может выполняться бесконечно — пока не выполнит все запрограммированные действия. Пример:

```
max_execution_time = 60
```

Но тут можно применить небольшой трюк. При учете времени выполнения сценария не учитываются системные вызовы. Представьте, что вам нужно создать PDF-отчет и вернуть его в браузер. Создание отчета занимает 5 минут. Что делать? Увеличить время выполнения сценария до 300 секунд? Не хотелось бы, поскольку это повлияет и на другие сценарии. В этом случае можно запускать сценарий, генерирующий отчет, через функцию `exec()`. Время выполнения команды, указанной в `exec()`, не лимитируется. В свою очередь, если в `exec()` указать команду вроде `php create-report.php`, то сценарий `create-report.php` будет запущен как бы из командной строки, и, следовательно, время его выполнения вообще не будет лимитировано:

```
exec("php create-report.php");
```

## 4.5. Загрузка файлов

Если ваше приложение подразумевает загрузку файлов (например, фото), то на сей процесс влияют следующие опции:

```
file_uploads = On  
upload_max_filesize = 10M  
max_file_uploads = 5
```

По умолчанию PHP разрешает производить 20 загрузок (`max_file_uploads`) с размером каждой из них 2 Мбайт (`upload_max_filesize`). Мы же здесь задаем другие параметры: максимальное количество одновременных загрузок — 5, а размер каждого файла — 10 Мбайт. Таких параметров будет достаточно, скажем, для загрузки пяти фотографий, сделанных зеркальной камерой. Параметр `file_uploads = On` включает механизм загрузки как таковой. Если его выключить (`Off`), то загрузка файлов не будет поддерживаться.

## 4.6. Обработка сессий

По умолчанию обработчик сессий PHP очень медленный, поскольку сохраняет данные сессии на диск. Это создает нежелательный ввод/вывод, который и является причиной «подтормаживания». Вы можете использовать обработчик сессии `Memcached`, позволяющий хранить данные сессии не на диске, а в оперативной памяти. Для этого нужно установить расширение `PECL Memcached` (<http://pecl.php.net/package/memcached>), а затем «прописать» использование этого обработчика в `php.ini`:

```
session.save_handler = 'memcached'  
session.save_path = '127.0.0.2:11211'
```

## 4.7. Буферизация вывода

Сети работают более эффективно, когда отправляют больше данных за меньшее число блоков, чем если бы они отправляли меньше данных за большее число блоков. Другими словами, для повышения производительности нужно отправлять контент в браузер меньшим числом блоков, отправляя при этом больше данных.

Именно поэтому на помощь приходит *буферизация вывода*. По умолчанию буферизация вывода включена (исключение — консольные сценарии). Вот рекомендуемые параметры, которые нужно установить в *php.ini*:

```
output_buffering = 4096  
implicit_flush = false
```

## 4.8. Директива *error\_reporting()*

Одна из очень полезных директив файла конфигурации — *error\_reporting*. Она управляет сообщениями об ошибках и предупреждениями, которые будут выводиться интерпретатором. Во время разработки сценария нужно включить вывод всех ошибок и других сообщений интерпретатора. Для этого служит функция:

```
error_reporting(E_ALL);
```

Вызов этой функции должен идти в сценарии самым первым. Когда же сценарий уже отлажен и работает без ошибок, вывод всех ошибок следует выключить:

```
error_reporting(0);
```

Дело в том, что при выводе ошибки PHP показывает информацию, которую не нужно знать злоумышленнику. Если у вас есть доступ к файлу *php.ini*, то вы можете вообще выключить вывод сообщений интерпретатора в браузер, но включить их протоколирование. Для этого служат две следующие директивы:

```
display_errors = Off  
log_errors = On
```

После этого все ошибки будут протоколироваться в журнал Apache.

## 4.9. Отключение потенциально опасных функций

С помощью директивы *disable\_functions* можно отключить потенциально опасные функции (после этого они не будут доступны в сценариях). Вот их список:

```
disable_functions = system, exec, passthru, shell_exec, proc_open
```

Есть смысл отключить выполнение этих функций в конфигурациях `apache2` и `php-fpm`. В конфигурации `cli`, как правило, эти функции не отключают.

## 4.10. Директива `allow_url_open`

Если эта директива включена (`on`), функции для работы с файлами могут открывать удаленные ресурсы (по протоколам HTTP/HTTPS). Если такое поведение не требуется, можно ее выключить (значение `off`).

\* \* \*

Остальные параметры файла `php.ini` не так важны. Найти описание всех его директив можно в официальной документации: <http://www.php.net/ini>.



## I. РАЗДЕЛ 2

# Передача параметров PHP-программам

<b>Глава 5.</b>	Методы <i>GET</i> и <i>POST</i>
<b>Глава 6.</b>	Протокол HTTP и интерфейс CGI
<b>Глава 7.</b>	Передача параметров посредством HTML-формы
<b>Глава 8.</b>	Не забываем о поисковой оптимизации



## ГЛАВА 5

# Методы *GET* и *POST*

### 5.1. Интерфейс CGI

CGI (Common Gateway Interface) — общий интерфейс шлюза, используемый для «общения» веб-клиентов (браузеров) с веб-серверами. В этой главе мы рассмотрим интерфейс CGI вкратце, поскольку полученные здесь знания при программировании на PHP вам пригодятся лишь частично.

Итак, предположим, что у нас есть веб-сервер. Какой именно и с какой операционной системой он работает — не имеет значения. Это может быть как Apache, работающий под управлением FreeBSD, так и Microsoft Internet Information Server, запущенный в Windows 2003 Server. Не удивляйтесь столь старым версиям — во времена популярности CGI они были именно такими.

В каталог `cgi-bin` мы помещаем простейшую программу, работающую со стандартным вводом и выводом. Эта программа не имеет графического интерфейса, все входные данные вводятся пользователем с клавиатуры. Вывод программа производит на экран. С такой программой можно работать только в режиме командной строки, иначе вы попросту не увидите результат ее работы.

В любой операционной системе есть средства перенаправления ввода/вывода. То есть вы можете вывод одной программы перенаправить на ввод другой. Ну, или запустить программу и передать данные на ее стандартный ввод. Следовательно, веб-сервер может запустить нашу программу, передать ей информацию, затем прочитать вывод этой программы и передать его в браузер.

На каком языке написана программа, не имеет значения. Это может быть исполняемый файл (`exe`) или обычный сценарий (расширение `bat` в Windows). В Linux/UNIX тоже нет ограничений на используемый язык программирования. Для большей определенности представим, что наша программа называется `example.cgi`.

Вот теперь у вас уже почти вырисовалась картина работы интерфейса CGI. Пользователь запускает браузер и вводит следующий адрес (URL):

`http://localhost/cgi-bin/example.cgi?name=denis`

Этот адрес означает, что пользователь пытается запустить нашу программу и передать ей параметр `name` со значением `denis`.

Получив такой URL, наш веб-сервер пытается найти и запустить программу `example.cgi`, передав ей на стандартный ввод параметры, указанные пользователем. Программа запускается и обрабатывает полученные параметры. Затем она выводит результат обработки параметров на стандартный вывод. Веб-сервер перехватывает вывод программы и отправляет его в браузер пользователя. Пользователь видит результат выполнения программы!

Нашей программе кажется, что параметры передал ей не сервер, а пользователь. И она просто выводит данные, ничего не подозревая о том, что они будут перехвачены и отправлены еще куда-то.

Благодаря такой «прозрачности» CGI-приложение можно написать на любом языке программирования. Для написания приложений не нужны какие-либо дополнительные библиотеки. С одной стороны, это преимущество, а с другой — недостаток. Но если у языка программирования есть средства для разработки CGI-приложений, то их разработка становится намного проще. Язык программирования PHP обладает такими средствами. Вы только взгляните на листинг 5.1 — в нем представлено самое простое CGI-приложение, которое всего лишь выводит переданные ему параметры.

#### Листинг 5.1. Пример простого CGI-приложения

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *Query_String = getenv("QUERY_STRING");

    printf("Content-type: text/html\n\n");
    printf("<p>Переданные сценарию параметры: %s", Query_String);
    return 0;
}
```

Переменная окружения `QUERY_STRING` содержит строку параметров, т. е. (в нашем случае):

```
name=denis
```

Если параметров больше, то строка параметров будет выглядеть сложнее, например:

```
name=denis&age=25
```

Наше приложение несовершенно. Здесь еще предстоит сделать разбор параметров, т. е. разобрать строку параметров на пары «параметр–значение».

В PHP все это делать не нужно. Вам достаточно обратиться к параметру `name` вот так:

```
$_GET['name']
```

Существуют два метода передачи сценариям параметров: `GET` и `POST`. Далее мы рассмотрим оба метода.

## 5.2. Метод *GET*

Сначала мы рассмотрим метод `GET` как более простой. Предположим, что у нас есть сценарий `hello.php`, которому нужно передать параметр `name`:

```
http://localhost/hello.php?name=denis
```

Браузер выделяет из введенного URL протокол (`http`), имя сервера (`localhost`), имя сценария. Затем браузер устанавливает подключение с сервером по заданному протоколу и передает вот такой запрос:

```
GET hello.php HTTP/1.0\n
различные заголовки
\n\n
```

У метода `GET` есть одна особенность — все параметры передаются через строку адреса браузера, и пользователь их видит невооруженным глазом. А поэтому метод `GET` нельзя использовать для передачи секретной информации — например, паролей. Да и максимальная длина строки в методе `GET` ограничивается 255 символами, поэтому вряд ли вы будете задействовать его для передачи больших объемов информации — файлов или длинных текстов.

Получить доступ к параметру, переданному методом `GET`, можно так:

```
$_GET['имя']
```

Например:

```
// присваиваем переменной $name значение параметра name
$name = $_GET['name'];
```

## 5.3. Метод *POST*

При использовании метода `POST` передаваемые данные не отображаются в окне браузера. Запрос выглядит так:

```
POST hello.php HTTP/1.0\n
Content-Length: 10\n
\n
0123456789
```

Как видите, передаваемые сценарию данные отображаются уже после самого запроса. Заголовок запроса `Content-Length` указывает, сколько байтов нужно прочитать серверу.

Метод `POST` обычно используется для передачи больших объемов данных — например, файлов. А поскольку передаваемые данные не отображаются в браузере, его можно использовать для передачи паролей.

Получить доступ к параметрам, переданным методом `POST`, можно через массив `$_POST`:

```
$_POST['имя']
```

Например:

```
// присваиваем переменной $name значение параметра name  
$name = $_POST['name'];
```

О передаче файлов методом `POST` мы еще поговорим в *главе 29*. Также рекомендую внимательно прочитать *главы 7 и 41* — в *главе 7* рассказано об использовании форм для передачи параметров (к слову, `POST`-параметры можно передать только с помощью формы или специальных программ), а в *главе 41* — о том, как работать с запросами и ответами в *Laravel*!

### **Массив `$_REQUEST`**

Кроме массивов `$_GET` и `$_POST` в PHP есть массив `$_REQUEST`, объединяющий в себе массивы `$_GET`, `$_POST` и `$_COOKIE`. Если вы не знаете, откуда «пришла» переменная (была передана методом `GET` или `POST` или получена через `Cookies`), тогда используйте этот массив: `$_REQUEST['имя']`. Подробно о массиве `$_REQUEST` мы поговорим в *главе 25*.

## ГЛАВА 6



# Протокол HTTP и интерфейс CGI

## 6.1. Специальные переменные окружения CGI

При создании серьезных PHP-сценариев не обойтись без специальных переменных окружения CGI. Например, переменная `REMOTE_ADDR` содержит IP-адрес удаленного пользователя. Получить значение переменной окружения можно с помощью функции `getenv()`:

```
echo getenv("REMOTE_ADDR");
```

Переменные окружения также доступны через суперглобальный массив `$_SERVER`:

```
echo $_SERVER['REMOTE_ADDR'];
```

Самые полезные переменные окружения представлены в табл. 6.1.

**Таблица 6.1.** Полезные переменные окружения

Переменная	Значение
<code>HTTP_REFERER</code>	Адрес сайта, с которого пользователь пришел на ваш сайт
<code>HTTP_USER_AGENT</code>	Браузер пользователя
<code>REMOTE_HOST</code>	Доменное имя удаленного компьютера
<code>REMOTE_PORT</code>	Удаленный порт (по этому порту браузер будет ждать ответа от сервера)
<code>QUERY_STRING</code>	Строка параметров, переданная сценарию (используется редко)
<code>SCRIPT_NAME</code>	Путь к сценарию
<code>SERVER_ADDR</code>	IP-адрес сервера
<code>SERVER_NAME</code>	Доменное имя сервера
<code>SERVER_PORT</code>	Порт сервера, обычно 80
<code>SERVER_PROTOCOL</code>	Версия HTTP-протокола

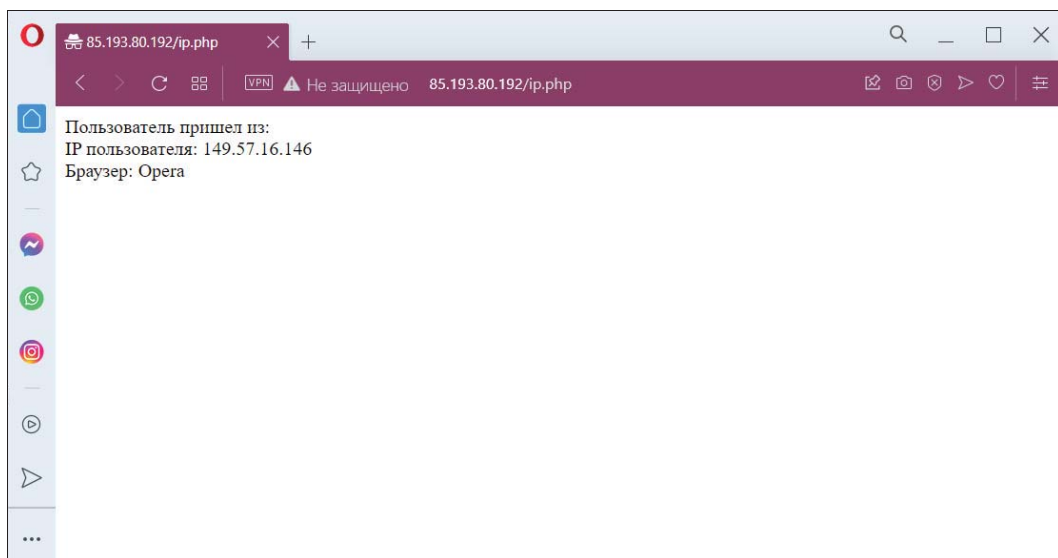


Рис. 6.1. Определение IP-адреса и браузера пользователя

Рассмотрим пример использования этих переменных окружения (рис. 6.1):

```
<?php

echo "Пользователь пришел из: ".getenv("HTTP_REFERER")."<BR>";

echo "IP пользователя: ".getenv("REMOTE_ADDR")."<BR>";

$browser=getenv("HTTP_USER_AGENT");
if(strpos($browser,"OPR")!=false)
    echo "Браузер: Opera<BR>";
else
    echo "Другой браузер ( $browser )<BR>";

?>
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *ip.php*), представленный в каталоге *Glava\_6* сопровождающего книгу электронного архива (см. приложение 4).

## 6.2. Заголовки протокола HTTP

Познакомимся с заголовками протокола HTTP, которые вам рано или поздно придется использовать в своих сценариях. Например, заголовок *Location* служит для перенаправления пользователя на другую страницу, а заголовки *GET/POST* — для получения файлов по протоколу HTTP (об этом мы поговорим в главе 16).

Вывести заголовки можно с помощью функции `Header()`, например:

```
// перенаправляем браузер пользователя на сайт https://example.com
Header("Location: https://example.com");
```

В табл. 6.2 представлены основные заголовки протокола HTTP.

Таблица 6.2. Заголовки протокола HTTP

Заголовок	Значение
Accept	Отправляется браузером серверу для информирования последнего о поддерживаемых MIME-типах, например <code>Accept: text/plain</code> . В данном случае поддерживается только обычный текст. Этот заголовок используется редко, и если вы не собираетесь разрабатывать собственный браузер, он вам не понадобится
Content-type	Информирует браузер о формате выводимых данных. Часто используемые форматы: <ul style="list-style-type: none"><li>• <code>text/plain</code> — обычный текст;</li><li>• <code>text/html</code> — текст в формате HTML;</li><li>• <code>image/jpeg</code> — изображение в формате JPEG;</li><li>• <code>image/png</code> — изображение в формате PNG;</li><li>• <code>image/gif</code> — изображение в формате GIF</li></ul>
Content-Length	Если используется передача данных серверу методом POST, то этот заголовок содержит длину передаваемых данных
Cookie	Содержит все Cookies, о них мы поговорим в главе 7
GET	Используется для получения файла методом GET, вот синтаксис запроса: <code>GET файл_или_сценарий?параметры HTTP/1.0</code>
Location	Перенаправление на другой сайт
POST	Используется для получения файла методом POST. Синтаксис такой же, как у заголовка GET
Pragma	Используется для всевозможных целей, чаще всего для запрета кеширования страницы: <code>Pragma: no-cache</code>
Referer	Содержит адрес сайта, откуда пользователь пришел на ваш сайт
User-Agent	Содержит информацию об используемом браузере, например: <code>Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]</code>

### 6.3. Коды ответов протокола HTTP

Наверняка каждый пользователь Интернета сталкивался с ошибкой 404, которая генерируется, если запрашиваемая страница не найдена. Вообще-то, код 404 — не ошибка, а обычный ответ сервера, который сообщает вам, что запрошенной стра-

ницы не существует. Тем не менее, коды из диапазона 4xx называются *ошибками клиента*. Код 404 попадает в эту категорию — пользователь ошибся при вводе адреса страницы сайта. Код 402 означает, что доступ к сайту платный. Иногда все ответы 4xx интерпретируются как ошибка 404 — все зависит от настроек сервера.

Кроме ответов 4xx есть и другие коды — например, коды из диапазона 2xx являются *кодами успеха*. Вы никогда их не увидите, поскольку если от сервера пришел код 200, значит, страница существует и отправлена в браузер. Понятное дело, что браузер выводит в этом случае не код ответа, а сразу полученную страницу.

Коды 5xx — это *ошибки сервера*. Например, сервер выводит ошибку 500 в случае, когда вы пытаетесь запустить сценарий, при выполнении которого возникла ошибка, и сервер больше не может продолжать его нормальное выполнение.

Кстати, коды 1xx называются *информационными* — они передают различную информацию, которая так или иначе относится к запросу, а коды 3xx — это коды *перенаправления*.



## ГЛАВА 7

# Передача параметров посредством HTML-формы

### 7.1. Создание простейшей формы и ее обработка в сценарии

В чем заключается цель любой программы? Правильно — в обработке данных. Программа получает исходные данные, затем обрабатывает их и передает результат. Как программа передаст результат: выведет в браузер пользователя, отправит по электронной почте или поместит в базу данных — нас сейчас не интересует, равно как и способы обработки данных (что программа может с ними делать). Но чтобы получить результат, нужно как минимум иметь входные данные. Вот они нас сейчас и интересуют.

Одним из часто используемых способов получения данных является ввод данных пользователем. HTML-форма (далее просто форма) предназначена для передачи РНР-сценарию введенных пользователем параметров. Форма описывается с помощью тега `FORM`:

```
<FORM action="сценарий" method="метод">  
<элементы формы>  
</FORM>
```

Мы указали два обязательных параметра формы: `action` и `method`. Вообще-то, и эти параметры не обязательны, но без них форма теряет свой смысл. Описание этих двух, а также остальных параметров формы представлено в табл. 7.1.

#### ПРИМЕЧАНИЕ

HTML-теги можно писать как прописными, так и строчными буквами — разницы никакой нет. Что и иллюстрируют примеры в этой главе.

*Таблица 7.1. Параметры формы*

Параметр	Описание
<code>action</code>	Задаёт сценарий, которому наша форма передает параметры
<code>method</code>	Определяет метод передачи параметров (GET или POST)

Таблица 7.1 (окончание)

Параметр	Описание
enctype	Тип кодирования данных формы. Обычно этот параметр определять не нужно. Но если вы собираетесь передавать файлы, то для этого параметра нужно установить значение <code>multipart/form-data</code>
name	Задаёт имя формы, через которое можно обращаться к её элементам в сценариях JavaScript/VBScript. При работе на PHP определять этот параметр не нужно
target	Позволяет определить окно, в котором будет загружаться обработчик формы (сценарий, указанный в параметре <code>action</code> ): <ul style="list-style-type: none"><li>• <code>_blank</code> — открывает страницу в новом окне браузера;</li><li>• <code>_self</code> — открывает страницу в текущем окне;</li><li>• <code>_parent</code> — открывает страницу во фрейме-родителе;</li><li>• <code>_top</code> — игнорирует все фреймы и открывает страницу в новом окне браузера. Если фреймов нет, то этот параметр работает как <code>_self</code></li></ul>

Предположим, что на сервере `localhost` имеется сценарий `hello.php`, которому нужно передать параметр `username`. В строке браузера это можно сделать так:

```
http://localhost/hello.php?username=denis
```

Теперь напишем HTML-форму `hello.html`, передающую сценарию параметр `username`:

```
<form action="http://localhost/hello.php" method="get">
Введите ваше имя <input type="text" name="username">
<input type="submit" value="OK" name="send">
</form>
```

В окне браузера эта форма будет выглядеть так, как показано на рис. 7.1.

Как только пользователь введёт своё имя и нажмёт кнопку **ОК**, форма методом `GET` передаст параметр `username` сценарию `hello.php`.

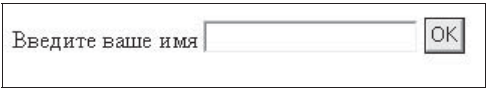


Рис. 7.1. Форма в окне браузера

Обратимся к сценарию `hello.php`. Нам нужно получить параметр `username`. Это можно сделать одним из способов:

```
echo $_GET['username'];
echo $_REQUEST['username'];
```

Первый вариант нужно использовать, если вы точно знаете, что параметр передан методом `GET`. А вот если вы забыли, каким методом передаете параметры, или же просто хотите сделать более универсальный сценарий, чтобы он работал с обоими методами передачи параметров, примените второй вариант.

В старых версиях PHP можно было использовать директиву `register_globals` (см. главу 8). Если она включена, то параметр `username` будет автоматически зарегистрирован как глобальная переменная `$username`:

```
echo $username;
```

Теперь напишем сценарий, который проверяет, передан ему параметр или нет. Если параметр не передан, то сценарий выводит форму. А вот если форма уже была отобрана и пользователь передал параметр, тогда мы его обработаем (просто выведем обратно в браузер). Проверить, существует ли параметр `username`, можно так:

```
if (isset($_GET['username'])) ...
```

Но лучше проверять наличие не параметра `username`, а параметра `send` — это наша кнопка. Так мы точно будем знать, что пользователь передал параметр через форму, а не вручную (листинг 7.1).

#### Листинг 7.1. Пример обработки параметра

```
<?php

if (!isset($_GET['send'])) {
    // выводим форму из hello.html
    echo file_get_contents('hello.html');
    // и прерываем выполнение сценария
    die();
}

// обрабатываем параметры
echo $_GET['username'];

?>
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *7-1.php*), представленный в каталоге *Glava\_7* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание, что для получения содержимого файла мы применяем следующую функцию, возвращающую содержимое файла:

```
file_get_contents ('hello.html');
```

Можно было бы вывести форму оператором `echo`, а не загружать из файла, но старайтесь, чтобы ваш PHP-код содержал минимум HTML-кода. Во-первых, так PHP-код будет компактнее, и его станет проще редактировать. Во-вторых, если ваш сценарий будет поддерживаться кем-то еще — например, дизайнером, которому со временем придется изменить HTML-код, лучше разделять код HTML и PHP.

## 7.2. Создание пользовательского интерфейса с помощью формы

Совсем недавно мы познакомились с двумя элементами формы: текстовым полем ввода и кнопкой отправки параметров (она называется **Submit**). Это далеко не все элементы формы. Вы можете использовать текстовые области (`textarea`), списки, зависимые и независимые переключатели и многое другое.

У каждого элемента формы должен быть параметр `name`, иначе вы не сможете к нему обратиться в PHP-сценарии:

```
<input type=text name=my_text>
```

Желательно все значения атрибутов HTML указывать в кавычках, например:

```
<input type="text" name="my_text">
```

Однако это создает определенные проблемы при выводе формы оператором `echo` — кавычки нужно экранировать:

```
echo "<input type=\"text\" name=\"my_text\">";
```

Одинарные кавычки использовать нельзя, потому что довольно часто нам бывает нужно задать значение по умолчанию, которое хранится в какой-нибудь переменной, а в одинарных кавычках вместо значения переменной вы увидите ее имя:

```
$val = "777";
```

```
// элемент формы будет содержать значение "777" (без кавычек)
echo "<input type=\"text\" name=\"my_text\" value=\"\$val\">";
```

```
// элемент формы будет содержать значение "$val" (без кавычек)
echo '<input type="text" name="my_text" value="$val">';
```

Впрочем, если значения HTML-параметров не содержат пробелов, от использования кавычек можно отказаться. Но значения, выводимые из сценария в форму, нужно всегда заключать в кавычки, поскольку они могут содержать пробелы:

```
$val = "два слова";
```

```
// элемент формы будет содержать значение "два слова" (без кавычек)
echo "<input type=text name=my_text value=\"\$val\">";
```

```
// элемент формы будет содержать значение "два" (без кавычек)
echo "<input type=text name=my_text value=$val>";
```

Вот теперь можно приступить к рассмотрению элементов формы.

### 7.2.1. Ввод текста. Теги *INPUT* и *TEXTAREA*

Тег `INPUT` является одним из самых часто используемых. Параметры тега `INPUT` представлены в табл. 7.2.

Таблица 7.2. Параметры тега `INPUT`

Параметр	Описание
type	Задаёт тип текстового поля: <ul style="list-style-type: none"><li>• <code>text</code> — обычный текст;</li><li>• <code>hidden</code> — скрытое поле, содержит значение, но не отображается на форме, и пользователь не может его изменить;</li><li>• <code>password</code> — поле для ввода пароля</li></ul>
value	Задаёт значение по умолчанию
size	Размер поля (в знако-местах)
maxlen	Максимальная длина строки, которую можно ввести в поле
name	Имя, по которому можно обращаться к элементу ввода в сценарии

Рассмотрим небольшой пример:

```
<input type=password size=8 maxlen=16>
```

Этот тег создаёт поле для ввода пароля (введённый текст будет заменяться звёздочками) длиной восемь знако-мест и максимальной длиной значения — 16 символов. То есть наше поле — небольшое, оно отображает только 8 символов (8 звёздочек, точнее), хотя в него можно ввести до 16 символов.

Теперь обратимся к тегу `TEXTAREA`. В отличие от тега `INPUT`, который позволяет ввести всего одну строку, тег `TEXTAREA` даёт возможность вводить многострочный текст:

```
<textarea параметры name=txt>
текст по умолчанию
</textarea>
```

Обратите внимание: если для тега `INPUT` не нужно было указывать закрывающий тег `/INPUT`, то для тега `TEXTAREA` закрывающий тег (`/TEXTAREA`) обязателен, иначе браузер не поймет, где заканчивается форма, и будет считать текстом поля `TEXTAREA` код формы.

Параметров у тега `TEXTAREA`, если не считать стандартного параметра `name`, всего три: `cols`, `rows` и `wrap`. Первые два — это соответственно ширина и высота текстового поля в знако-местах. Последний параметр задаёт тип переноса текста:

- ☐ `virtual` — самый оптимальный вариант переноса текста с автоматической полоской прокрутки и автоматическим переносом;
- ☐ `physical` — зависит от браузера, и предсказать, что появится в том или ином браузере, невозможно;
- ☐ `none` — вообще без переноса, текст вводится как есть. Это не очень удобно, но иногда полезно.

## 7.2.2. Зависимые и независимые переключатели

*Независимые* переключатели потому и называются независимыми, что состояние (включен/выключен) одного из переключателей формы не зависит от всех остальных.

Значения *зависимых* переключателей зависят друг от друга — включенным может быть только один из них. На рис. 7.2 представлена форма с зависимыми и независимыми переключателями.

Независимые переключатели	Зависимые переключатели
<input checked="" type="checkbox"/> Да, я хочу получать рассылку	Пол: <input checked="" type="radio"/> Мужчина <input type="radio"/> Женщина
<input checked="" type="checkbox"/> Да, я хочу получать сообщения от других пользователей	

Рис. 7.2. Зависимые (справа) и независимые (слева) переключатели

Для создания переключателей служит уже знакомый нам тег `INPUT`. Тип переключателя регулируется параметром `type`:

- ☐ `checkbox` — независимый переключатель;
- ☐ `radio` — зависимый переключатель.

Кроме параметра `type` вы можете задать следующие параметры:

- ☐ `name` — имя переключателя;
- ☐ `value` — значение переключателя;
- ☐ `checked` — переключатель активен.

Рассмотрим пример создания независимых переключателей:

```
<input type=checkbox value=yes name=subscr checked> Да, я хочу получать рассылку
<input type=checkbox value=yes name=email checked> Да, я хочу получать сообщения от других пользователей
```

Если пользователь включил переключатель, то сценарию будет передан параметр:

`имя_переключателя=значение`

Например, если пользователь хочет получать рассылку сайта, но не хочет получать сообщения от других пользователей, то сценарию будет передан параметр `subscr` со значением `yes` (значение устанавливается параметром `value`).

```
if (isset($_REQUEST['subscr']))
echo "Пользователь хочет получать рассылку сайта";
```

```
if (isset($_REQUEST['email']))
echo "Пользователь хочет получать сообщения пользователей";
```

Зависимые переключатели, подобные показанным на рис. 7.2, создаются так:

```
<input type=radio value=male name=sex checked> Мужчина
<input type=radio value=female name=sex> Женщина
```

Работают такие переключатели несколько иначе. Обратите внимание, что у обоих переключателей одно и то же имя: `sex`. Первый переключатель (со значением `male`) включен по умолчанию. Когда пользователь нажмет кнопку **ОК**, форма передаст сценарию параметр `sex`, но его значение будет зависеть от выбранного переключателя:

```
// проверяем, был ли передан параметр
if (isset($_REQUEST['sex'])) {
    // параметр передан, проверяем значение
    if ($_REQUEST['sex'] === "male") echo "Мужчина";
    else echo "Женщина";
}
```

### 7.2.3. Списки выбора

Тег `SELECT` позволяет организовать списки выбора. Параметр `size` задает размер списка в строках, параметр `multiple` разрешает выбор нескольких значений списка. Установить ширину поля со списком можно с помощью свойства стиля `width`.

Рассмотрим два списка (листинг 7.2).

#### Листинг 7.2. Два списка выбора

```
<p>Выберите год:
<select name=year size=1>
<option value=2023>2023</option>
<option value=2022>2022</option>
<option value=2020>2020</option>
<option value=2019>2019</option>
<option value=2018>2018</option>
<option value=2017>2017</option>
<option value=2016>2016</option>
</select>

<p>Какая операционная система у вас установлена?
<p><select name=os[] size=3 multiple>
<option value=10>Windows 10</option>
<option value=11>Windows 11</option>
<option value=linux>Linux</option>
</select>
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `7-2.html`), представленный в каталоге *Glava\_7* сопровождающего книгу электронного архива (см. приложение 4).

Первый список (рис. 7.3) не представляет собой ничего интересного — это обычный выпадающий список. Форма передает нашему сценарию параметр `year` со зна-

чением, совпадающим с одним из значений `value` из тега `OPTION`. Например, если пользователь выберет **2023**, то будет передан параметр:

```
year=2023
```

Обработать список можно так:

```
if (isset($_REQUEST['year']))  
echo $_REQUEST['year'];
```

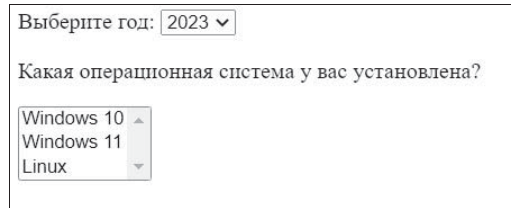


Рис. 7.3. Два списка выбора

У нас значения параметров довольно простые и не содержат пробелов, поэтому мы могли бы вообще упростить объявление списка до следующего вида:

```
<select name=year size=1>  
<option>2023</option>  
<option>2022</option>  
<option>2021</option>  
<option>2020</option>  
<option>2019</option>  
<option>2018</option>  
<option>2017</option>  
</select>
```

В этом случае в качестве значения `year` будет передан текст между тегами `OPTION` и `/OPTION`.

А теперь представим, что нам нужно сформировать список, возвращающий номер месяца, но пользователю не очень удобно работать с числами, ему проще выбрать название месяца. Поэтому параметр `value` тега `OPTION` нам очень пригодится:

```
<select name=month size=1>  
<option value=1>Январь</option>  
<option value=2>Февраль</option>  
<option value=3>Март</option>  
...
```

Во втором списке (см. рис. 7.3), как вы уже заметили, значения параметра `value` тоже отличаются от текста, заключенного в тег `OPTION`. Второй список разрешает множественный выбор благодаря параметру `multiple`. Обратите внимание, что после имени второго списка идут две квадратные скобки. Это означает, что мы передаем массив параметров, т. е. несколько значений, которые может выбрать пользователь.

Вот пример обработки такого списка:

```
$os = $_REQUEST['os'];

// проверяем, является ли $os массивом
if (is_array($os)) {
    foreach ($os as $key => $value) {
        echo "$value<br>";
    }
}
```

Если пользователь выбрал Windows 10 и Linux, то сценарий выведет:

```
10
linux
```

## 7.2.4. Форма для передачи файлов

С помощью тега `INPUT` можно создать поле загрузки файлов на сервер, но об этом мы поговорим в *главе 29*.

## 7.2.5. Кнопки

Установив все параметры, пользователь нажмет кнопку отправки параметров — уже знакомую нам кнопку **Submit**. Существуют два варианта этой кнопки: кнопка отправки формы и кнопка сброса (устанавливает параметры по умолчанию):

```
<input type=submit name=send value=OK>
<input type=reset value=Сброс>
```

Иногда, чтобы реализовать несколько действий, нам нужны несколько кнопок — например, кнопки **Добавить** и **Удалить**. А как определить, какую кнопку нажал пользователь? Для этого нужно не забыть определить параметр `name` кнопок:

```
<input type=submit name=add value=Добавить>
<input type=submit name=delete value=Удалить>
```

В сценарии кнопки можно обработать так:

```
// определяем, какую кнопку нажал пользователь
if (isset($_REQUEST['add'])) {
    // пользователь нажал кнопку "Добавить"
    ...
}
elseif (isset($_REQUEST['delete'])) {
    // пользователь нажал кнопку "Удалить"
    ...
}
else die('Пользователь не нажимал кнопки формы, а пытается запустить сценарий напрямую');
```

Наш сценарий не только сообщает, какую кнопку нажал пользователь, но и определяет, пытается ли пользователь запустить сценарий напрямую (возможно, он пытается взломать сайт?). Конечно, при использовании метода GET такую проверку легко обойти, добавив к строке запроса параметр `add=Добавить`. Но в случае метода POST проделать такое будет сложнее. Правда, тогда нужно будет искать параметр не в массиве `$_REQUEST`, а в массиве `$_GET`.

## 7.3. Проверка параметров формы

Вы никогда не должны доверять параметрам, полученным из формы. Виной тому человеческий фактор: кто-то может ввести неправильные данные по ошибке (например, ошибиться при вводе номера кредитной карты), а кто-то вводит неправильные данные умышленно — с целью взлома сайта. О защите сайта мы поговорим в *главе 43*, а сейчас рассмотрим основные принципы проверки параметров формы:

1. Обязательно проверяйте существование параметров формы с помощью функции `isset()` перед обращением к ним, чтобы не попасть в ситуацию запуска пользователем сценария без параметров.
2. Если проверять значения всех параметров вам не хочется (их может быть столь много, что такая проверка отрицательно скажется на читабельности сценария), поможет инициализация переменных, т. е. установка параметров по умолчанию. Этот способ представляется более уместным — даже если форма не передаст параметр, у вас будут параметры по умолчанию, и вы сможете продолжить работу с этими параметрами вместо вывода сообщения об ошибке.

Рассмотрим второй способ. Предположим, что мы ожидаем два параметра: год рождения (`$year`) и имя пользователя (`$username`). Установим значения по умолчанию, а затем получим значения этих параметров:

```
$year = 2023;  
$username='user';  
$year = $_REQUEST['year'];  
$username = $_REQUEST['username'];
```

Существует еще одна стратегия проверки — проверка типа значения. Если вы ожидаете строку, то нужно убедиться, что переданный параметр — строка, а не что-то иное (число, например). Если вы, наоборот, ждете число, тогда нужно убедиться, что пользователь ввел число, а не строку.

Для проверки типов вы можете использовать следующие функции:

- `is_numeric()` — возвращает `true`, если переданный ей аргумент — число;
- `is_string()` — возвращает `true`, если переданный ей аргумент — строка;
- `is_array()` — возвращает `true`, если переданный ей аргумент — массив.

Но проверить то, что значение является строкой, — мало. Атака по типу SQL-инъекции как раз и заключается в том, что злоумышленник отправляет на сервер строку, содержащую вредоносный SQL-код. Есть два способа защиты: или ис-

пользовать регулярные выражения, если вы знаете, на что должна быть похожа строка, — например, можно задать шаблон для номера телефона, или же фильтровать содержимое функцией `filter_input()`.

PHP содержит богатый набор фильтров: валидационные, санитизирующие (удаляющие запрещенные символы) и др. Подробно об этом мы поговорим в *главе 43*, когда будем рассматривать защиту сайта.

Иногда нужно проверить, соответствует ли введенная строка определенному формату — например: пользователь ввел e-mail или произвольную строку? Далее мы займемся проверкой корректности ввода e-mail и номера банковской карты.

### 7.3.1. Проверка корректности e-mail

Напишем функцию `is_email()`, которая будет проверять, является ли введенный пользователем текст электронным адресом. Если формат e-mail правильный, то функция возвращает `true`. В противном случае — `false`. Обратите внимание, что функция не проверяет существование домена пользователя, и если пользователь введет адрес **user@domain.com**, то функция сообщит, что адрес правильный, поскольку он соответствует правилам написания электронных адресов. Однако функция не гарантирует, что домен **domain.com** существует (листинг 7.3).

#### Листинг 7.3. Проверка корректности e-mail

```
function is_email($email) {  
    if (!preg_match('/^[A-Za-z0-9!#$%&\'*+~\-=/?^_`{|}~]+@[A-Za-z0-9-]+  
    (\.[A-Za-z0-9-]+)+[A-Za-z]$/', $email)) {  
        return false;  
    } else {  
        return true;  
    }  
}
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *7-3.php*), представленный в каталоге *Glava\_7* сопровождающего книгу электронного архива (см. приложение 4).

Для проверки формата адреса функция использует *регулярные выражения*. Подробно о них мы поговорим в *главе 33*, а пока вам нужно лишь знать, как эту функцию применять:

- ❑ добавьте описание функции в начало PHP-сценария или вынесите его в отдельный PHP-файл и подключите с помощью инструкции `require` (см. *главу 22*);
- ❑ вызовите функцию так:

```
$email = $_REQUEST['email'];  
  
if (is_email($email)) echo "E-mail правильный";  
else echo "Введите правильный e-mail!";
```

## 7.3.2. Проверка правильности номера банковской карты

Банковские карты (дебетовые и кредитные) все больше входят в обиход, и даже у нас теперь не боятся оплачивать с их помощью покупки в интернет-магазинах. Обработкой транзакции, т. е. снятием определенной суммы со счета клиента, занимается отдельная платежная система (с которой вы должны заключить договор). Выбор системы зависит от многих факторов, но сейчас речь не об этом. До выполнения запроса на транзакцию вы должны хотя бы убедиться, что клиент ввел номер банковской карты, а не свой домашний адрес. Выполнить проверку можно также с помощью регулярных выражений, зная формат номера банковской карты, — вам он понадобится для создания регулярного выражения.

### Листинг 7.4. Функция проверки корректности номера банковской карты

```
function is_cc_num($cc_num) {

    $card_type = "";

    // массив регулярных выражений, который используется для
    // определения типа карты
    $cards = array(
        '/^4\d{12}(\d\d\d){0,1}$/' => 'visa',
        '/^5[12345]\d{14}$/' => 'mastercard'
    );
    // определение типа карты (visa или mastercard)
    foreach ($cards as $reg_ex => $type) {
        if (preg_match($reg_ex, $cc_num)) {
            $card_type = $type;
            break;
        }
    }

    // мы не можем определить тип карты, возвращаем false
    if (!$card_type) {
        return false;
    }

    // проверка контрольной суммы карты
    $code = strrev($cc_num);
    $crc = 0;

    for ($i = 0; $i < strlen($code); $i++) {
        $current_num = intval($code[$i]);
        if ($i & 1) {
            $current_num *= 2;
        }
    }
}
```

```
$src += $current_num % 10;
if ($current_num > 9) {
    $src += 1;
}

if ($src % 10 == 0) {
    return $card_type;
} else {
    return false;
}
}
```

### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *7-4.php*), представленный в каталоге *Glava\_7* сопровождающего книгу электронного архива (см. *приложение 4*).

В начале функции мы задаем массив регулярных выражений, позволяющих определить тип банковской карты: VISA или MasterCard (если вы знаете форматы номеров карточек других систем, то можете расширить эту функцию после прочтения *главы 33*). Затем мы, используя этот массив, устанавливаем тип карты. Если тип карты установить не удалось, мы возвращаем *false*.

Далее мы проверяем контрольную сумму карты. Эта процедура стандартна, поэтому в большинстве случаев вы сможете использовать ее для проверки корректности номера карт и других систем. Если контрольная сумма не делится на 10, значит, мы возвращаем *false*. Если карта прошла все проверки, значит, мы возвращаем ее тип: *visa* или *mastercard*.

Использовать функцию очень просто:

```
echo is_cc_num("4111 1111 1111 1111");
```

### 7.3.3. Удаление лишних пробелов

Чего только не вводят пользователи в поля формы! Довольно часто они почему-то вводят пробелы до и после строки. Лишние пробельные символы можно удалить функцией *trim()* — например, так:

```
$username = trim($username);
```

Об остальных строковых функциях мы поговорим в *главе 13*.

## 7.4. Директива `@csrf` шаблонизатора Blade

Подробно шаблонизатор Blade рассматривается в *главе 23*. Здесь же мы рассмотрим лишь одну из его директив — `@csrf`, которая встраивает в форму специальный токен, предотвращающий CSRF-атаку на ваше веб-приложение. Эту директиву нужно поместить в защищаемую форму:

```
<form action="/api/test" method="post">  
@csrf  
...  
</form>
```

Если вы забудете указать директиву `@csrf`, Laravel не будет обрабатывать запрос из формы (см. главу 41).

Поскольку в этой главе рассматриваются формы, о директиве `@csrf` нельзя было не упомянуть, — новички слишком часто забывают о ней и потом тратят время, чтобы разобраться, что пошло не так.



## ГЛАВА 8

# Не забываем о поисковой оптимизации

### 8.1. «Дружественные» интернет-адреса

При работе над сайтом не следует забывать о его *поисковой оптимизации* — лучше озаботиться этим в процессе создания сайта, чем заниматься оптимизацией уже после запуска его в работу. Но как ни крути, а разработчик никогда не станет хорошим оптимизатором — нужно следить за последними новшествами в этой области знаний, как говорится, «быть в тренде», а на это у простого разработчика, как правило, нет времени. Однако есть некоторые моменты, которые при разработке сайта нужно учитывать обязательно. Один из таких моментов: SEF (Search Engine Friendly) URL — «дружественные» к поисковым машинам интернет-адреса.

Поисковые машины не любят, когда в интернет-адреса (URL) передаются какие-либо параметры. Посмотрим на один из адресов такого сайта:

**<https://example.com/book/98>**.

Он является дружественным к поисковым машинам. Но он мог бы выглядеть и так:

**<https://example.com/index.php?page=book&id=98>**.

Здесь видно, что сценарию `index.php` передаются два параметра: первый выбирает тип страницы (шаблона) для отображения контента. Понятно, что будет выводиться книга, поэтому название шаблона `book`. Второй параметр — это `id` книги.

Так вот, поисковым машинам такие адреса очень не нравятся. Раньше некоторые поисковые машины даже отказывались их индексировать, а некоторые и сейчас «переваривают» 1–2 параметра, а страницы с большим числом параметров в индекс не принимают. Нужно отметить, правда, что «Яндекс» и Google в настоящее время более или менее нормально относятся к динамическим страницам (именно так в терминологии SEO<sup>1</sup> называют страницы с параметрами), но правила хорошего тона поисковой оптимизации говорят, что нужно использовать «дружественные» к поисковым машинам интернет-адреса. В 2023 году созданный вами сайт, где адреса имеют множество параметров, будет смотреться в их глазах очень странно.

---

<sup>1</sup> SEO — поисковая оптимизация (от *англ.* Search Engine Optimization).

### 8.1.1. Организация SEF URL с помощью файла *.htaccess*

Забегая вперед, скажу, что организовать такие адреса можно путем редактирования основного (находящегося в корневом каталоге сайта) файла *.htaccess*. Причем здесь программирование?

А притом, что:

- в файле *.htaccess* нужно будет указывать регулярные выражения. Вы видели когда-нибудь веб-мастера или дизайнера, который бы разбирался в регулярных выражениях? А в этой книге им посвящена целая глава (см. главу 33);
- одного редактирования файла *.htaccess* мало. Нужно, чтобы ваши сценарии, генерирующие ссылки (например, сценарии, отвечающие за вывод меню, боковой панели и т. д.), генерировали «дружественные» к поисковым машинам интернет-адреса (SEF URL). То есть чтобы они выводили ссылки вида `/book/<ID>`, а не `index.php?p=book&id=<ID>`.

С выводом ссылок, думаю, вам все понятно — просто измените операторы, формирующие ссылки. А вот с файлом *.htaccess* нужно разбираться.

Чтобы перезапись URL была выполнимой задачей, на вашем веб-сервере (или на сервере хостинг-провайдера) должен быть установлен модуль перезаписи URL. Для Apache это `mod_rewrite`, для Microsoft IIS — `ISAPI_Rewrite`.

Возможности этих модулей настолько огромны, что мы не можем рассмотреть их в этой книге. Кроме того, синтаксис `mod_rewrite` отличается от синтаксиса `ISAPI_Rewrite`. Все написанное далее относится к `mod_rewrite`, поскольку сервер Apache в Интернете встречается гораздо чаще, чем Microsoft IIS. Впрочем, некоторые хостеры используют и другие серверы. В этом случае обратитесь в службу технической поддержки хостера, они должны вам помочь — хотя бы предоставить ссылку на документацию, где описано, как реализовать SEF URL на их хостинге.

Прежде всего нужно включить механизм перезаписи URL. Это делается путем добавления в файл *.htaccess* следующей строки:

```
RewriteEngine on
```

Далее надо добавить в этот файл правила перезаписи URL. У вас может быть несколько таких правил. Правила формируются директивой `RewriteRule`. Перейдем сразу к практике и рассмотрим следующую строчку:

```
RewriteRule ^book/([^/]+)$ /index.php?page=book&id=$1
```

Здесь видно, что URL вида `book/<ID>` будет перенаправляться на `/index.php?page=book&id=<ID>`. Кстати, это не простое перенаправление, а именно перезапись URL, — в адресной строке останется именно SEF URL.

Теперь разберемся в том, как сформирована наша строчка. Директиве `RewriteRule` передаются два регулярных выражения: первое задает нужный вам URL, второе — старый URL (тот, который с параметрами).

Рассмотрим составляющие наших регулярных выражений:

- ❑ знак `^` обозначает начало URL (после имени домена);
- ❑ знак `$` обозначает конец URL;
- ❑ знак `+` обозначает одно или несколько вхождений символов — т. е. мы ожидаем, что в идентификаторе будет указано несколько символов;
- ❑ используя символ `$1`, можно обратиться к тому, что сохранено в памяти, в нашем случае — в первом наборе скобок `[]`. Если у нас будет два набора скобок, то обратиться ко второму можно с помощью `$2` и т. д.

В нашем примере допускается использование любых идентификаторов — например, адрес `/book/php_mysql_ed6` будет перенаправлен в `/index.php?page=book&id=php_mysql_ed6`. Если вам нужны только цифровые идентификаторы, следует указать это в скобках.

Рассмотрим еще один пример:

```
RewriteRule ^tip/([0-9]+)/?$ /tip.php?id=$1 [L]
```

Цифры `0-9` в скобках означают, что мы ожидаем в качестве идентификатора продукта только число. Символ `[L]` в конце строки означает, что в случае совпадения с правилом нужно остановиться. В противном случае будут выполнены также все оставшиеся правила, если они есть. Без `[L]` регулярные выражения будут «жадными». У таких «жадных» выражений есть не очень приятная особенность. Рассмотрим это на примере:

```
RewriteRule ^(.*)/?index\.html$ /$1/ [L,R=301]
```

Здесь предписано перенаправление с `http://www.site.com/dir/index.html` на `http://www.site.com/dir/`. Но это явно не то, чего мы хотели. А так получилось, поскольку фрагмент `.*` захватит символ косой черты еще до того, как фрагмент `/?` его увидит.

Исправить ситуацию достаточно просто: нужно использовать символы `^` или `.*?` (вместо `.*`). Например, поставьте `^(.*?)/?` вместо `^(.*)/?` или `[/]+[/]` вместо `./`.

Более корректное правило будет выглядеть так:

```
RewriteRule ^(.*)/?index\.html$ /$1/ [L,R=301]
```

Подобных примеров можно привести довольно много, но их рассмотрение уже выходит за рамки этой книги. Интересующимся могу порекомендовать посетить следующую страницу:

**<http://web-optimizator.com/301-redirekt-htaccess/>**

Здесь вы также найдете правила, устраняющие произвольное зеркало сайта, — когда к сайту можно обратиться по адресам как с префиксом `www`, так и без него. Нужно выбрать один какой-то формат имени и использовать его. Выбор окончательно закрепляется через файл `.htaccess`.

Вот правила для сайта, пример с которым приведен в начале главы:

```
RewriteEngine On
RewriteCond %{HTTP_HOST} ^example.com
RewriteRule (.*?) http://www.example.com/$1 [R=301,L]
```

Первая директива вам знакома. Вторая определяет условие перезаписи URL, а именно, когда к узлу обратились без `www`. Затем следует директива `RewriteRule`, которая перенаправляет на заданный адрес — с `www`.

### 8.1.2. Использование фреймворков

При использовании `Laravel` или любого другого фреймворка ваши URL изначально будут SEF. Все маршруты (т. е. URL, которые будут задействованы приложением) указываются в файле `web.php` в следующем виде:

```
Route::get('/', 'App\Http\Controllers\Controller@Main');
Route::get('/about', 'App\Http\Controllers\Controller@About');
```

Первый параметр метода `get()` — это URL, на который должно реагировать веб-приложение, а второй — название метода, который будет обрабатывать этот URL. Подробно о маршрутах мы поговорим в *главе 38*.

## 8.2. Идентификаторы сеансов

Лет десять тому назад для передачи идентификаторов сеанса использовался один трюк, а именно — обход `cookies` при работе с сессиями. В этом случае идентификатор сеанса `SID` прикрепляется и передается вместе с URL:

```
<a href=admin.php?<?=SID?>>Ссылка</a>;
```

На самом деле такой сценарий весьма нежелателен. Скорее всего, поисковые машины не станут индексировать страницы с идентификаторами сеансов, поскольку идентификатор сеанса для поисковой машины — это обычный параметр (как в предыдущем разделе).

При этом, поскольку идентификатор меняется при каждом посещении сайта, адреса страниц также будут постоянно меняться. И даже если поисковик и проиндексирует страницу с `SID`, то, когда он посетит сайт в следующий раз, он ее снова проиндексирует, но под другим адресом — `SID` ведь изменился. Следовательно, может возникнуть проблема дублированного контента. Даже не знаю, что поисковые машины не любят больше: дублированный контент или же идентификаторы сеансов.

Дважды подумайте, нужно ли обходить `cookies` вот таким образом!

## 8.3. Производительность сценария

Задача PHP-сценария — вывести в браузер HTML-код. Существуют различные требования к самому HTML-коду — например, CSS и JavaScript-сценарии, которые упомянуты в HTML-коде, должны быть минимизированы, JavaScript-код должен

указываться в конце HTML-страницы, а не в самом начале (чтобы не тормозить отображение страницы), и т. д. Все эти правила — задача дизайнера, и пусть именно он ломает над ними голову. Мы же, как программисты, должны вывести HTML-код максимально быстро, чтобы устранить задержку на стороне сервера. Далее мы рассмотрим несколько способов ускорения ваших PHP-сценариев.

### 8.3.1. Использование шаблонизатора. Сокращение количества инструкций `echo`

Ускорить вывод HTML-кода можно несколькими способами. Первый из них — использование *шаблонизатора* вроде того же Blade или Smarty, который описан в *главе 23*. Преимущество готового решения в том, что в нем уже все оптимизировано, причем над оптимизацией вывода трудилась целая команда разработчиков. К тому же у шаблонизатора есть поддержка кеширования страниц, а оно дает огромное ускорение.

Если вы считаете, что использование таких решений, как Smarty, не оправдано на вашем сайте, то можете создать собственный шаблонизатор (см. *главу 22*). Да, у него не будет поддержки кеширования, но весь HTML-код станет выводиться только один раз.

Самую большую задержку вносит оператор `echo/print`. В случае с использованием шаблонизатора, даже самого простого и самодельного, мы сначала формируем HTML-код, а потом выводим его в браузер. Можно обойтись и без шаблонизатора, а просто объединить все операторы `echo` в один. Например, весь такой код:

```
$header = "Это заголовок сайта";
echo $header;
$content = "Это содержимое страницы";
echo $content;
$footer = "А здесь живут счетчики и копирайты";
echo $footer;
```

можно заменить следующим:

```
$html = $header . $content . $footer;
echo $html;
```

Здесь мы объединили контент в одну переменную и вывели ее в браузер одним оператором. В итоге у вас будет одна операция ввода/вывода, а не три.

Конечно, лучше использовать шаблонизатор, поскольку вы получите не только сокращение количества инструкций `echo`, но еще и возможность кеширования.

### 8.3.2. Включение OPcache

OPcache, или Zend OPcache — это расширение, позволяющее повысить производительность PHP путем кеширования скомпилированного байт-кода (см. также *разд. 4.3*). Современный PHP — это уже не чистый интерпретатор, которым он был в начальных версиях. Сейчас ваш сценарий сначала компилируется в промежуточный байт-код, а затем происходит выполнение этого байт-кода.

Даже самое простое приложение состоит из нескольких скриптов, к которым постоянно обращаются пользователи. Например, у вас может быть скрипт `index.php`, представляющий главную страницу сайта, а также `article.php`, выводящий определенную статью. Если `OPcache` выключено, то PHP при каждом обращении к этим сценариям генерирует байт-код заново. Если же включить `OPcache`, то код будет сгенерирован один раз, а дальше будет использоваться повторно. В качестве преимущества вы получите повышение производительности, а в качестве недостатка — если вы часто вносите изменения в свои сценарии, то вам придется после каждого изменения перезагружать веб-сервер (или сервис `php-fpm`, если у вас `nginx`), чтобы изменения вступили в силу (очистился кеш). Как вариант, на время разработки вы можете выключить `OPcache`, а затем, когда все будет готово, включить его снова. О том, как включить `OPcache`, было рассказано в *разд. 4.3*.

### 8.3.3. Включение HTTP 2.0

Включение HTTP 2.0 (HTTP/2) существенно повысит производительность вашего приложения особенно в том, что касается времени TTFB (Time To First Byte) — проверено на практике. У автора этой книги был сервер следующей конфигурации: `Nginx & Php-fpm`, PHP 7.2 — на котором выполнялось довольно сложное приложение. Именно включение HTTP 2.0 позволило существенно сократить TTFB.

В `nginx` для включения HTTP 2.0 нужно добавить инструкцию `http2` в блок `server`, а именно — в директиву `listen`:

```
server {  
    listen      443 ssl http2;
```

В `Apache` все сложнее — надо установить модуль `mod_http2` и произвести кое-какую настройку. Подробно об этом рассказывается в документе: <https://httpd.apache.org/docs/2.4/howto/http2.html>.

### 8.3.4. Обновление версии PHP

Существует аксиома: каждая следующая версия PHP быстрее, чем предыдущая. Пусть ненамного, но быстрее. Однако, если вы делаете апгрейд, скажем, с версии 7.2 на 8.2, то можете получить существенный прирост в производительности.

При чем здесь поисковая оптимизация? Дело в том, что поисковые машины накладывают серьезные ограничения на скорость вывода страницы. Считается хорошим результатом, если ваша страница загружается полностью за 2–3 секунды. Мобильная же версия страницы должна загружаться за 1 секунду. Поэтому способность версии 8.2 более быстро загружать страницы может сыграть вам на руку.

Проверить скорость загрузки вашего сайта можно инструментом Google PageSpeed Insights: <https://pagespeed.web.dev/>.

Этот инструмент хорош еще и тем, что не только сообщает скорость загрузки, но и предоставляет рекомендации по улучшению результатов именно для вашего сайта.



# **I. РАЗДЕЛ 3**

## **Массивы и списки**

<b>Глава 9.</b>	Основные операции над массивами и списками
<b>Глава 10.</b>	Функции сортировки массивов
<b>Глава 11.</b>	Особые операции над массивами



## ГЛАВА 9

# Основные операции над массивами и списками

### 9.1. Массив и список. Цикл *foreach*

Ранее в этой книге мы уже сталкивались с *массивами*, поскольку без них нельзя было реализовать тот или иной фрагмент кода. И в дальнейшем с массивами вы будете встречаться очень часто.

Массив — это упорядоченный набор данных. У каждого элемента массива есть свой индекс, или ключ (*index, key*), который однозначно идентифицирует элемент массива, — таким образом, индекс уникален для каждого элемента.

Давайте создадим массив:

```
$os[0] = "UNIX";  
$os[1] = "Linux";  
$os[2] = "Windows 10";  
$os[3] = "Windows 11";  
$os[4] = "macOS";
```

Массив `$os` содержит пять элементов, нумерация которых начинается с нуля. Поскольку после имени переменной мы указываем квадратные скобки, РНР понимает, что мы имеем дело с массивом, а не с обычной переменной.

Вывести все элементы массива можно так:

```
for ($i=0; $i < count($os); $i++) echo "$os[$i] <br>";
```

В этом цикле нет ничего необычного, кроме функции `count()`. Эта функция возвращает количество элементов массива. Для функции `count()` также определен псевдоним `sizeof()`. Какую функцию использовать: `count()` или `sizeof()` — разницы нет, используйте ту, название которой вам больше нравится.

РНР обладает очень удобной функцией — при добавлении в массив нового элемента совсем не обязательно указывать его индекс, РНР присвоит ему следующий свободный индекс. Приведенный ранее массив можно было бы создать так:

```
$os[] = "UNIX";  
$os[] = "Linux";  
$os[] = "Windows 10";
```

```
$os[] = "Windows 11";  
$os[] = "macOS";
```

Нумерация элементов в нем также будет начата с нуля.

По типам массивы в PHP различаются на ассоциативные массивы и списки:

- *список* — это обычный массив, где в качестве индексов элементов выступают числа. В нашем случае массив `$os` представляет собой список;
- *ассоциативный массив* — это массив, в котором в качестве индекса элементов массива выступают произвольные строки. Вот пример создания ассоциативного массива:

```
$DB["host"] = "localhost";  
$DB["user"] = "root";  
$DB["password"] = "";  
$DB["db_name"] = "test";
```

Ассоциативный массив `$DB` содержит параметры доступа к базе данных: узел, имя пользователя, пароль и имя базы данных. Обратиться к любому из элементов массива можно по его индексу:

```
echo $DB["host"];
```

Специально для перебора ассоциативных массивов используется оператор цикла `foreach`. Синтаксис его таков:

```
foreach ($массив as $ключ => $значение) { тело; }
```

Например:

```
foreach ($DB as $key => $value)  
    echo "$key => $value <br>";
```

Впрочем, чисто технически можно было бы реализовать перебор ассоциативного массива и циклом `for`, например:

```
for (reset($DB); ($k = key($DB)); next($DB))  
    echo "$k => " . current($DB) . "<br>";
```

Цикл `for` также удобно применять для обратного перебора ассоциативного массива:

```
for (End($DB); ($k = key($DB)); Prev($DB))  
    echo "$k = {$DB[$k]} <br>";
```

В последних двух примерах мы использовали функции `reset()`, `end()`, `next()`, `prev()`, `current()` и `key()`:

- функция `reset()` устанавливает внутренний указатель массива на первый элемент массива, функция `end()` — на последний;
- функция `next()` устанавливает указатель на следующий элемент массива, а функция `prev()` — на предыдущий;
- функция `key()` получает текущий ключ массива, а функция `current()` возвращает текущий элемент массива, т. е. значение текущего элемента. Вместо функции

`current()` можно использовать функцию `pos()`, которая является псевдонимом `current()`.

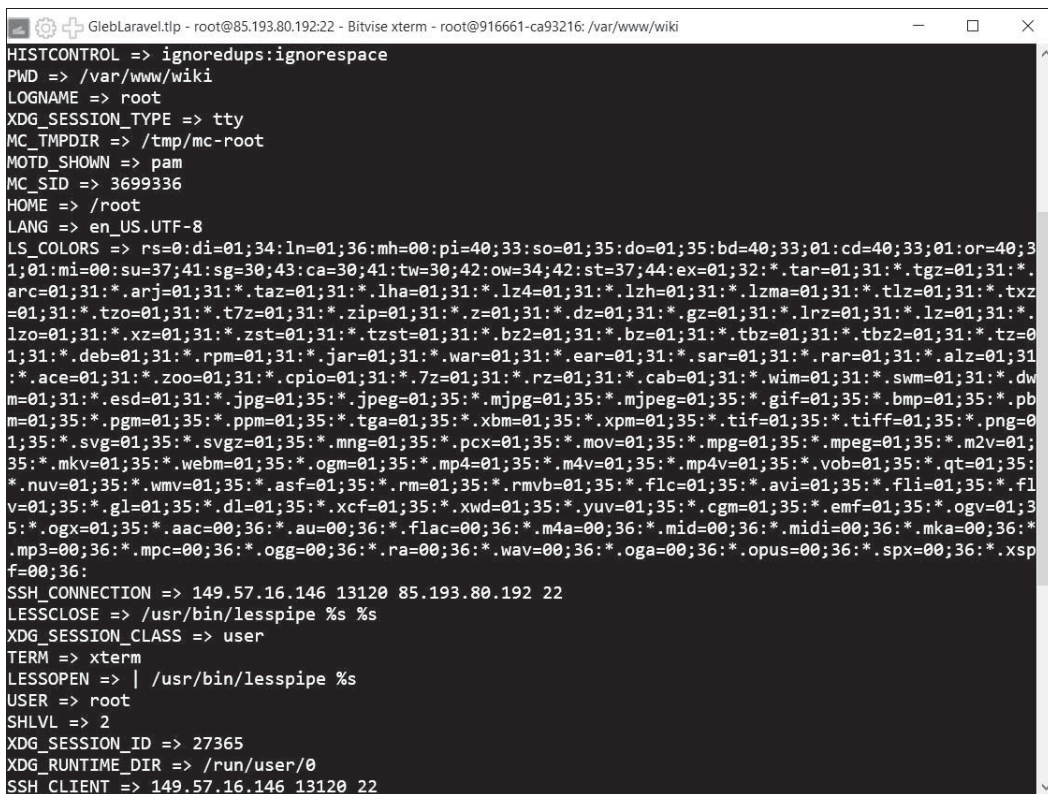
Для прямого (не обратного) перебора массива цикл `foreach` использовать гораздо удобнее, чем цикл `for`. Вот пример перебора глобального массива `$_SERVER`, содержащего все глобальные переменные нашего сервера:

```
<?php

foreach($_SERVER as $key => $value)
    echo "$key => $value\n";

?>
```

Вывод этого сценария приведен на рис. 9.1. Особой ценности этот вывод не имеет, однако на примере такого массива удобно продемонстрировать работу цикла `foreach`.



```
GlebLaravel.tlp - root@85.193.80.192:22 - Bitwise xterm - root@916661-ca93216: /var/www/wiki
HISTCONTROL => ignoredups:ignorespace
PWD => /var/www/wiki
LOGNAME => root
XDG_SESSION_TYPE => tty
MC_TMPDIR => /tmp/mc-root
MOTD_SHOWN => pam
MC_SID => 3699336
HOME => /root
LANG => en_US.UTF-8
LS_COLORS => rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;33;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tztst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.taz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dw
m=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pb
m=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=0
1;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;
35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:
*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.fl
v=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;3
5:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:
.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xsp
f=00;36:
SSH_CONNECTION => 149.57.16.146 13120 85.193.80.192 22
LESSCLOSE => /usr/bin/lesspipe %s %s
XDG_SESSION_CLASS => user
TERM => xterm
LESSOPEN => | /usr/bin/lesspipe %s
USER => root
SHLVL => 2
XDG_SESSION_ID => 27365
XDG_RUNTIME_DIR => /run/user/0
SSH_CLIENT => 149.57.16.146 13120 22
```

Рис. 9.1. Вывод массива `$_SERVER`

Начиная с РНР 7, в цикле `foreach` произведены важные изменения. Во-первых, итерация по массиву при помощи функции `foreach()` больше не сдвигает внутренний указатель массива, который можно получать и изменять при помощи функций `current()`, `next()`, `reset()` и им подобных. Например, в РНР 7/8 такой код:

```
$array = [0, 1, 2];
foreach ($array as &$val) {
    var_dump(current($array));
}
```

выведет три раза `int(0)`. Раньше он бы вывел `int(1)`, `int(2)`, `int(3)`, т. е. позицию указателя.

Во-вторых, `foreach` по значению теперь всегда работает с копией массива. Поэтому любое изменение массива в процессе итерации бессмысленно. Например, в РНР 7 такой код:

```
$array = [0, 1, 2];
$ref =& $array;          // Если нужно старое поведение
foreach ($array as $val) {
    var_dump($val);
    unset($array[1]);
}
```

выведет все три элемента (0, 1 и 2), а ранее код вывел бы элементы 0 и 2, поскольку элемент 1 был бы удален во время первой итерации.

В-третьих, если вам все еще нужно изменять массив в процессе итерации, вы можете использовать ссылки (оператор `&`). Например, такой код:

```
$array = [0];
foreach ($array as &$val) {
    var_dump($val);
    $array[1] = 1;
}
```

выведет теперь `int(0)` `int(1)`. Если бы мы не использовали `&`, в выводе был бы только `int(0)`.

## 9.2. Функции *list()* и *array()*

Вернемся к нашему массиву `$os` и рассмотрим функцию `list()`:

```
list($os0, $os1, $os2, $os3, $os4) = $os;
```

Переменным `$os0`–`$os5` будут присвоены значения элементов массива `$os[0]`–`$os[5]` соответственно.

Функция `array()` служит для создания массивов — как списков, так и ассоциативных:

```
// пример создания списка
$os = array("UNIX", "Linux", "Windows 10", "Windows 11", "macOS");

// ассоциативный массив
$DB = array("host" => "localhost", "user" => "root",
            "password" => "", "db_name" => "test");
```

Чуть ранее мы создавали массивы `$os` и `$DB` другим способом. Представленные сейчас способы аналогичны рассмотренным ранее. Вам нужно выбрать тот способ, который кажется вам наиболее удобным.

Элементы массива тоже могут быть массивами, т. е. вы можете создавать многомерные массивы. Вот пример массива, содержащего параметры для двух MySQL-соединений:

```
$DB = array (
    "0" => array("host" => "localhost", "user" => "root",
                "password" => "", "db_name" => "test" ),
    "1" => array("host" => "server.ru", "user" => "root",
                "password" => "", "db_name" => "test" )
);
```

Получить доступ к параметрам первого соединения можно так:

```
echo $DB["0"]["host"];
echo $DB["0"]["user"];
```

В PHP 5.5 появилась возможность использовать функцию `list()` внутри цикла `foreach` — например:

```
$arr1 = [
    [1, 2],
    [3, 4],
    [5, 6]
];

foreach ($arr1 as list($x, $y)) {
    echo "X = $x; Y = $y <BR>";
}
```

Сценарий выведет следующее:

```
X = 1; Y = 2
X = 3; Y = 4
X = 5; Y = 6
```

Начиная с PHP 7, в функцию `list()` внесли много изменений. Первое изменение коснулось порядка добавления элементов. Теперь `list()` не добавляет элементы в обратном порядке. Например:

```
list($array[], $array[], $array[]) = [1, 2, 3];
var_dump($array);
```

Элементы в массиве будут размещены в порядке добавления, т. е. [1, 2, 3], а не [3, 2, 1], как было раньше.

Затем больше нельзя присваивать пустые списки. Поэтому все следующие операторы в PHP 7/8 недопустимы:

```
list() = $a;
list(,) = $a;
list($x, list(), $y) = $a;
```

Также `list()` больше не поддерживает распаковку строк. Рассмотрим код:

```
$string = "xy";  
list($x, $y) = $string;
```

В результате `$x` и `$y` будут равны `null`. Раньше этот код в переменную `$x` поместил бы значение "x", а в переменную `$y` — значение "y".

Однако `list()` теперь работает с объектами посредством `ArrayAccess`, например:

```
list($a, $b) = (object) new ArrayObject([0, 1]);
```

Здесь `$a` будет равна 0, а `$b` — 1. Ранее такой код присвоил бы обоим переменным значение `null`.

## 9.3. Удаление массива

Вам вряд ли понадобится удалять массив, разве что он содержит очень много информации, которую вы уже быстренько (помните об ограничении на время выполнения сценария?) обработали, и хотите продолжить работу сценария, но так, чтобы не превысить ограничение на размер используемой памяти.

Удалить массив можно так же, как и обычную переменную, — с помощью функции `unset()`:

```
unset($DB);
```

## 9.4. Слияние массивов

Слияние (объединение) массивов подразумевает создание одного массива, состоящего из элементов объединяемых массивов. Слияние осуществляется с помощью оператора `+`.

Предположим, что у нас есть два массива:

```
$SERVER = array("host" => "localhost", "user" => "root", "password" => "");  
$DB = array("dbname" => "test", "table" => "example");
```

Выполним их слияние:

```
$DBINFO = $SERVER + $DB;
```

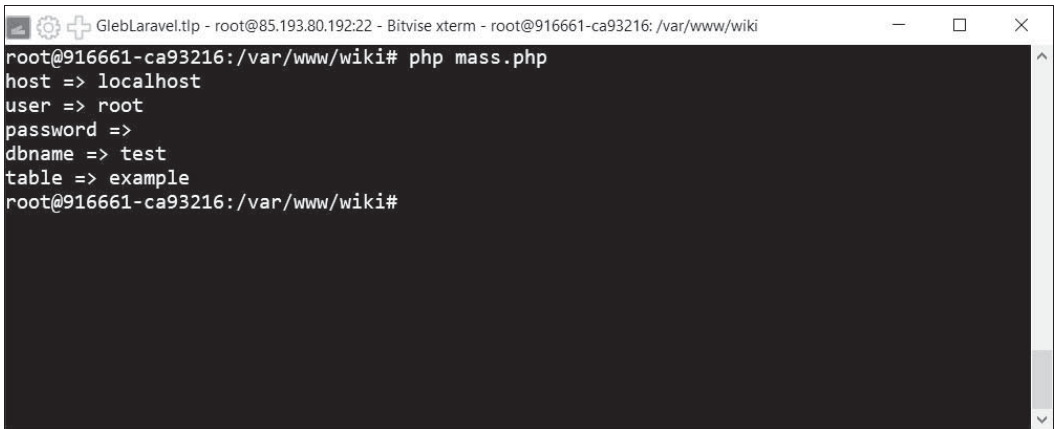
Получим следующий массив:

```
"host" => "localhost", "user" => "root", "password" => "",  
"dbname" => "test", "table" => "example"
```

Напомним, что вывести массив можно так:

```
foreach($DBINFO as $key => $value)  
    echo "$key => $value\n";
```

Этот вывод показан на рис. 9.2.



```
GlebLaravel.tlp - root@85.193.80.192:22 - Bitvise xterm - root@916661-ca93216: /var/www/wiki
root@916661-ca93216:/var/www/wiki# php mass.php
host => localhost
user => root
password =>
dbname => test
table => example
root@916661-ca93216:/var/www/wiki#
```

Рис. 9.2. Вывод массива `$DBINFO`

А теперь попробуем объединить два списка, например:

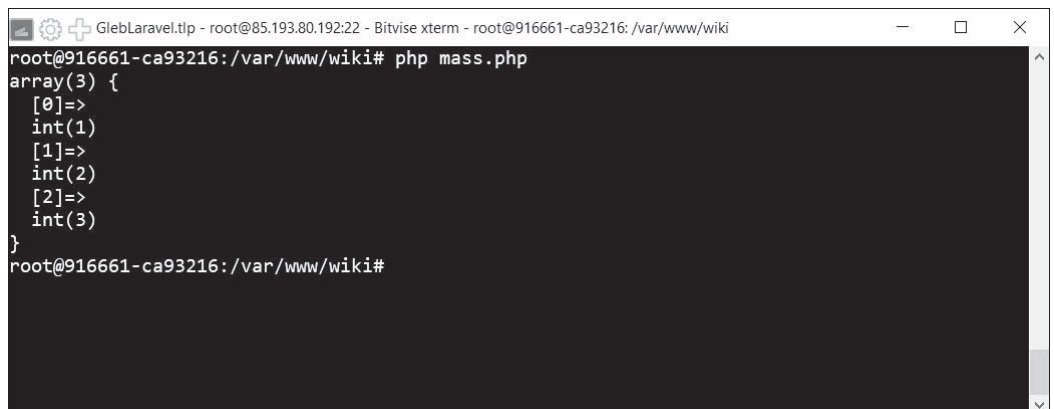
```
$A1 = array(1, 2, 3);
$A2 = array(4, 5, 6);
$A3 = $A1 + $A2;
```

При отладке, когда не нужен «красивый» вывод, гораздо проще использовать функцию `var_dump()`, чем цикл `foreach`, и перебирать массив с ее помощью. Функция `var_dump()` сделает за нас всю работу и предоставит более информативный вывод (рис. 9.3):

```
var_dump($A3);
```

Однако результат вывода нас удивит — полученный массив `$A3` вместо значений 1, 2, 3, 4, 5, 6 будет содержать только значения 1, 2, 3.

Почему? Это особенность оператора `+`. Если при слиянии массивов будут найдены элементы с одинаковыми индексами, то оператор `+` возьмет их из первого массива. Вот так-то...



```
GlebLaravel.tlp - root@85.193.80.192:22 - Bitvise xterm - root@916661-ca93216: /var/www/wiki
root@916661-ca93216:/var/www/wiki# php mass.php
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  int(3)
}
root@916661-ca93216:/var/www/wiki#
```

Рис. 9.3. Вывод массива `$A3`

Устранить этот недостаток можно с помощью функции `array_merge()`, которая сравнивает не ключи, а пары `ключ=>значение`. Если для ассоциативных массивов с успехом можно использовать оператор `+`, то для слияния списков удобно применять `array_merge()`:

```
$A1 = array(1, 2, 3);  
$A2 = array(4, 5, 6);  
$A3 = array_merge($A1, $A2);  
var_dump($A3);           // см. рис. 9.4
```

В этом случае мы получим то, что нам нужно, а именно — массив `$A3`, содержащий все значения массивов `$A1` и `$A2` (рис. 9.4).

Для слияния многомерных массивов служит функция `array_merge_recursive()`:

```
array array_merge_recursive ( array $array1 [, array $... ] )
```

A screenshot of a terminal window titled 'GlebLaravel.tlp - root@85.193.80.192:22 - Bitvise xterm - root@916661-ca93216: /var/www/wiki'. The terminal shows the command 'php mass.php' being executed. The output is a PHP array dump: 'array(6) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=> int(4) [4]=> int(5) [5]=> int(6) }'. The prompt 'root@916661-ca93216: /var/www/wiki#' is visible at the bottom.

```
root@916661-ca93216:/var/www/wiki# php mass.php  
array(6) {  
  [0]=>  
    int(1)  
  [1]=>  
    int(2)  
  [2]=>  
    int(3)  
  [3]=>  
    int(4)  
  [4]=>  
    int(5)  
  [5]=>  
    int(6)  
}  
root@916661-ca93216:/var/www/wiki#
```

Рис. 9.4. Результат слияния массивов функцией `array_merge()`

## 9.5. Функция *print\_r()*

Иногда нужно вывести массив в отладочных целях — просто, чтобы посмотреть его элементы. Мы уже познакомились с функцией `var_dump()`, а теперь рассмотрим функцию `print_r()`, вывод которой еще лучше подходит для отображения содержимого массивов. Какой из этих двух функций воспользоваться, решать вам: функция `var_dump()` выводит больше полезной информации — например, тип значения, а вывод функции `print_r()` более понятен для человека и не содержит ничего лишнего.

Для глубокого анализа лучше, конечно, взять `var_dump()`, а для того, чтобы просто заглянуть внутрь массива, удобнее применить `print_r()`. Помните, что обе эти функции используются исключительно из соображений отладки, и их не следует применять в производственной версии вашего кода.

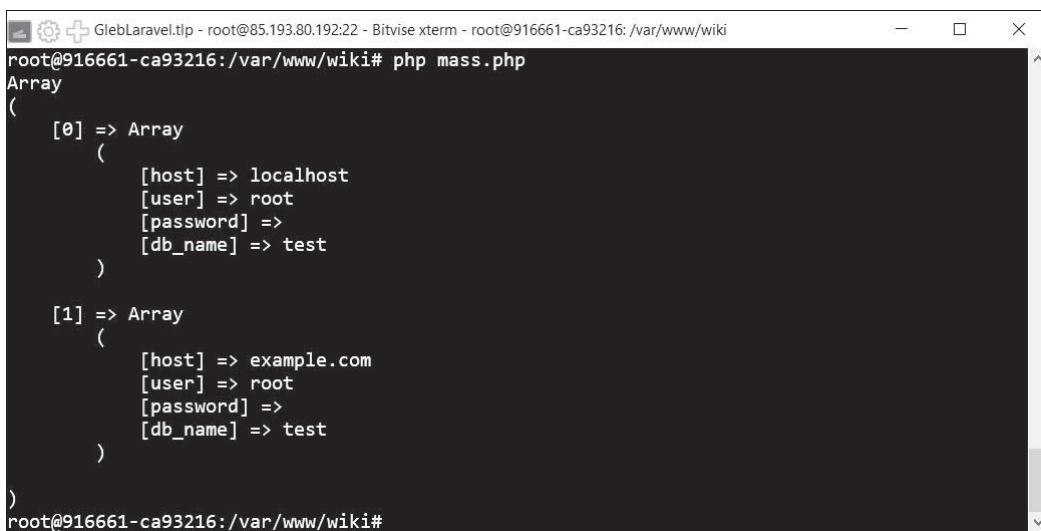
Итак, чтобы не возиться с `foreach()` и `echo`, намного проще вызвать функцию `print_r()`:

```
$DB = array (
    "0" => array( "host" => "localhost", "user" => "root",
        "password" => "", "db_name" => "test" ),
    "1" => array( "host" => "example.com", "user" => "root",
        "password" => "", "db_name" => "test" )
);

print_r($DB);
```

Вывод функции `print_r()` представлен на рис. 9.5.

Вообще, функция `print_r()` — весьма разносторонняя и позволяет «распечатать» информацию о переменной любого типа: от обычной текстовой переменной до MySQL-результата или объекта.



```
GlebLaravel.tlp - root@85.193.80.192:22 - Bitwise xterm - root@916661-ca93216:/var/www/wiki
root@916661-ca93216:/var/www/wiki# php mass.php
Array
(
    [0] => Array
        (
            [host] => localhost
            [user] => root
            [password] =>
            [db_name] => test
        )
    [1] => Array
        (
            [host] => example.com
            [user] => root
            [password] =>
            [db_name] => test
        )
)
root@916661-ca93216:/var/www/wiki#
```

Рис. 9.5. Вывод массива с помощью функции `print_r()`

## 9.6. Разыменовывание массива

Возможность разыменовывания массива (array dereferencing) появилась еще в версии PHP 5.4. Представим, что у нас есть функция, возвращающая массив:

```
function a() {
    return array(0, 1, 1, 0, 1, 0, 0, 1);
}
```

Как можно видеть, функция возвращает массив из восьми элементов. Раньше нужно было получить результат этой функции (массив), а потом уже обращаться к элементам полученного массива:

```
$arr1 = a();  
echo $arr1[5];
```

Начиная с версии 5.4, вы можете сделать это так:

```
echo a()[5];
```

Другими словами, есть возможность обращаться к элементам массива, возвращаемого функцией, без предварительного присваивания массива временной переменной.

В любом случае функция возвращает массив, а не один лишь элемент, поэтому память все равно выделяется. Если вы работаете с очень большими массивами информации и заботитесь об использовании памяти, просто не забывайте очищать временные переменные функцией `unset()`:

```
$arr1 = a();  
echo $arr1[5];  
unset($arr1);
```



## ГЛАВА 10

# Функции сортировки массивов

## 10.1. Сортировка массивов

Сортировка — это одна из самых востребованных операций над массивом. Получив в массив данные (например, из файла), вам нужно их отсортировать и вывести в браузер пользователю.

В PHP есть много функций сортировки массива. В этой главе мы изучим лишь самые востребованные, а также рассмотрим создание собственной функции сортировки.

## 10.2. Функция `sort()` — сортировка списка

Функция `sort()` служит для сортировки обычных массивов, т. е. списков. Прототип этой функции выглядит так:

```
bool sort ( array &array [, int sort_flags] )
```

Рассмотрим пример использования этой функции:

```
$A = array (3, 1, 9, 5, 0, 4, 5); // исходный список
sort($A);                       // сортируем
print_r($A);                    // выводим
```

Результат сортировки представлен на рис. 10.1. У функции `sort()` есть аналог — функция `rsort()`, выполняющая сортировку в обратном порядке. Набор параметров у нее такой же, как у функции `sort()`.

Вот несколько фактов, которые вам необходимо знать об этой функции:

- ☐ функция возвращает `true`, если сортировка успешно завершена. Обычно так оно и происходит, поэтому нет необходимости проверять результат функции;
- ☐ функция назначает новые ключи для элементов массива, поэтому ее нельзя использовать для сортировки ассоциативных массивов, иначе вы рискуете потерять данные;
- ☐ функция выполняет сортировку по возрастанию (или в алфавитном порядке, если элементы массива — строки);

- ❑ вы можете указать следующие флаги сортировки (второй параметр функции):
- `SORT_REGULAR` — обычное сравнение элементов без изменения их типов;
  - `SORT_NUMERIC` — сравнивать элементы как числа;
  - `SORT_STRING` — сравнивать элементы как строки;
  - `SORT_LOCALE_STRING` — сравнивать элементы как строки, учитывая текущую «локаль» (языковые параметры системы).



```
root@916661-ca93216:/var/www/wiki# php mass.php
Array
(
    [0] => 0
    [1] => 1
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 5
    [6] => 9
)
```

Рис. 10.1. Результат сортировки массива

## 10.3. Функция `asort()` — сортировка массива по значениям

В PHP вам чаще придется работать с ассоциативными массивами. Для их сортировки предназначена функция `asort()`:

```
bool asort ( array &array [, int sort_flags] )
```

Функция `asort()` выполняет сортировку по значениям. Представим, что у нас есть массив `$fruits`. Обратите внимание, что ни ключи, ни значения этого массива не расположены в алфавитном порядке:

```
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");
```

Выполним его сортировку:

```
asort($fruits);
```

А теперь выведем содержимое массива функцией `print_r()`:

```
Array
(
    [c] => apple
```

```
[b] => banana
[a] => orange
)
```

Как видите, функция `asort()` выполнила сортировку по значениям массива — они расположены в алфавитном порядке.

Кроме функции `asort()` вы можете использовать функцию `arsort()`, выполняющую сортировку по значениям в обратном порядке. Результат применения этой функции к нашему массиву будет таким:

```
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
)
```

Флаги сортировки у функций `asort()` и `arsort()` такие же, как у функции `sort()`.

## 10.4. Функция *ksort()* — сортировка по ключам

Функция `ksort()` работает подобно функции `asort()`, но выполняет сортировку не по значениям, а по ключам. Сортировку по ключам в обратном порядке обеспечивает функция `krsort()`.

Сортировку по ключам иллюстрирует следующий фрагмент кода:

```
// исходный массив
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");
// сортировка по ключам
ksort($fruits);
print_r($fruits);
// сортировка по ключам в обратном порядке
krsort($fruits);
print_r($fruits);
// вывод будет следующим:
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
)
Array
(
    [c] => apple
    [b] => banana
    [a] => orange
)
```

## 10.5. Функции `array_reverse()` и `shuffle()`

Функция `array_reverse()` возвращает массив с элементами в обратном порядке, например:

```
$fruits = array("b" => "banana", "a" => "orange", "c" => "apple");  
$reverse_fruits = array_reverse($fruits);
```

Массив `$reverse_fruits` будет таким:

```
Array  
(  
    [c] => apple  
    [a] => orange  
    [b] => banana  
)
```

Функция `shuffle()` перемешивает элементы массива в произвольном порядке. Каким окажется вывод `shuffle()`, сказать сложно — в этом и прелесть случайной сортировки. Однако помните, что эта функция полностью изменяет массив, поэтому ее нужно использовать только для списков.

## 10.6. Собственная функция сортировки

До этого мы сортировали массивы, состоящие из строк и чисел. Но в массив вы можете поместить все что угодно — например, результаты MySQL-запросов. Как в этом случае выполнить сортировку?

Здесь вам помогут «пользовательские» версии уже рассмотренных функций:

- `usort()` — сортировка списка;
- `uasort()` — сортировка массива по значениям;
- `uksort()` — сортировка массива по ключам.

Этим функциям нужно передать два параметра: массив, который нужно отсортировать, и *пользовательскую функцию сортировки*.

Функция сортировки должна принять два аргумента (тип аргументов такой же, как у элементов массива) и «сказать», какой из двух аргументов больше. Точнее, функция должна передать одно из следующих значений:

- `-1`, если первый элемент меньше второго;
- `0`, если элементы равны;
- `1`, если первый элемент больше второго.

Рассмотрим небольшой пример:

```
<?php  
function cmp($a, $b)  
{  
    if ($a == $b) {
```

```

        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$a = array(4, 2, 5, 7, 1);

usort($a, "cmp");

print_r($a);

?>

```

Наша функция сравнения — `cmp()`. Она не делает ничего сверхъестественного — просто сравнивает числа: аргументы `$a` и `$b`. Если `$a` равно `$b`, то функция возвращает 0, если `$a` меньше `$b`, то возвращается `-1` или `1` — в противном случае.

Обратите внимание на конструкцию:

```
return ($a < $b) ? -1 : 1;
```

Работает она так. Вычисляется истинность выражения в скобках. Если выражение истинно (в нашем случае `$a < $b`), то возвращается первое значение после вопросительного знака (`-1`), в противном случае возвращается второе значение (`1`).

## 10.7. Натуральная сортировка

Представим, что у нас есть массив, содержащий имена файлов:

```
$files= array("ch13.doc", "ch10.doc", "ch3.doc", "ch1.doc");
```

Попробуем отсортировать его функцией `asort()`:

```
asort($files);
print_r($files);
```

Результат будет таким:

```
Array
(
    [3] => ch1.doc
    [1] => ch10.doc
    [0] => ch13.doc
    [2] => ch3.doc
)
```

С точки зрения компьютера ошибок нет — все элементы отсортированы в лексикографическом порядке. Но нам бы хотелось, чтобы файл главы 3 следовал сразу после файла главы 1. Чтобы реализовать задуманное, нам нужно использовать алгоритм натуральной сортировки:

```
bool natsort ( array &$array )
```

Если отсортировать наш массив функцией `natsort()`, результат будет таким:

```
Array
(
    [3] => ch1.doc
    [2] => ch3.doc
    [1] => ch10.doc
    [0] => ch13.doc
)
```

Функция `natsort()` чувствительная к регистру символов, но в РНР есть аналогичная функция, которая игнорирует регистр символов:

```
bool natcasesort ( array &$array )
```

Сравним результаты `natsort()` и `natcasesort()`:

```
$files = $files2 = array("ch13.doc", "Ch10.doc", "ch3.doc", "ch1.doc");
natsort($files);
print_r($files);

natcasesort($files2);
print_r($files2);
```

А теперь посмотрим на вывод:

```
Array
(
    [1] => Ch10.doc
    [3] => ch1.doc
    [2] => ch3.doc
    [0] => ch13.doc
)
Array
(
    [3] => ch1.doc
    [2] => ch3.doc
    [1] => Ch10.doc
    [0] => ch13.doc
)
```

Второй результат — сортировка функцией `natcasesort()` — больше соответствует представлениям о «человеческой» сортировке, не говоря уже о сортировке с помощью `asort()`.



## ГЛАВА 11

# Особые операции над массивами

### 11.1. Добавление и удаление элементов массива

Добавить элемент в конец массива можно с помощью оператора []:

```
$массив[] = новый_элемент;
```

Кроме оператора [] можно использовать функцию `array_push()`:

```
int array_push ( array &array, mixed var [, mixed ...] )
```

После добавления элемента в конец массива функция возвращает новое количество элементов массива. Заметьте, что с помощью `array_push()` можно добавить не один элемент, а несколько, — например:

```
array_push(массив, элемент1, элемент2, элемент3 ...)
```

Для удаления последнего элемента массива служит функция `array_pop()`:

```
mixed array_pop ( array &array )
```

Допустим, у нас есть массив фруктов:

```
$fruits = array("orange", "banana", "apple", "linux");
```

Удалим «лишний» элемент, который не имеет никакого отношения к фруктам:

```
$element = array_pop($fruits);
```

Теперь посмотрим, что после этого будут содержать массив `$fruits` и переменная `$element`:

```
print_r($fruits);  
print_r($element);
```

Вот что выведут функции `print_r()`:

```
Array  
(  
    [0] => orange
```

```
[1] => banana
[2] => apple
)
linux
```

Как видите, из массива `$fruits` удален последний элемент, а переменная `$element` как раз и содержит этот элемент.

Теперь попробуем добавить элементы в начало списка. Для этого нужно использовать функцию `array_unshift()`:

```
int array_unshift ( array &array, mixed var [, mixed ...] )
```

Эта функция, как и `array_push()`, может добавить в массив сразу несколько элементов, например:

```
$fruits = array("orange", "banana", "apple");
array_unshift($fruits, "lemon", "tangerine");
```

Теперь наш массив будет выглядеть так:

```
Array
(
    [0] => lemon
    [1] => tangerine
    [2] => orange
    [3] => banana
    [4] => apple
)
```

Удалить первый элемент массива можно с помощью функции `array_shift()`:

```
array_shift($fruits);
```

После этого массив будет выглядеть так:

```
Array
(
    [0] => tangerine
    [1] => orange
    [2] => banana
    [3] => apple
)
```

## 11.2. Упаковка переменных в массив и их извлечение

Предположим, что у нас есть три переменные:

```
$host = "localhost";
$user = "root";
$password = "secret";
```

Упакуем их в массив `$DB` с помощью функции `compact()`:

```
$DB = compact("host", "user", "password");
```

#### **ПРИМЕЧАНИЕ**

Заметьте, что функции `compact` нужно передать имена переменных, а не их значения! То есть нужно вызвать функцию как `compact("host", "user", "password")`, а не `compact($host, $user, $password)`.

Функция `compact()` создаст следующий массив:

```
Array
(
    [host] => localhost
    [user] => root
    [password] => secret
)
```

Для распаковки ключей массива в переменные служит функция `extract()`:

```
int extract ( array var_array [, int extract_type [, string prefix]] )
```

Например:

```
extract($DB);
```

При распаковке массива будут созданы переменные с именами, соответствующими названиям ключей массива, т. е. `$host`, `$user` и `$password`. Значения этих переменных будут такими же, как у соответствующих им ключей массива.

Все вроде просто, если не принимать во внимание один факт. Что делать, если переменные `$host`, `$user` и `$password` уже существуют?

Поведение функции регулируется вторым параметром:

- ☐ `EXTR_OVERWRITE` — если распаковываемая переменная уже существует, она будет перезаписана;
- ☐ `EXTR_SKIP` — если переменная с таким именем существует, будет сохранено ее прежнее значение;
- ☐ `EXTR_PREFIX_SAME` — если переменная существует, к ее имени будет добавлен префикс, заданный третьим параметром функции `extract()`;
- ☐ `EXTR_PREFIX_ALL` — префикс, заданный третьим параметром, будет добавлен ко всем извлекаемым переменным.

В главе 7 мы познакомились с функцией `serialize()`, которая может сериализовать массив, т. е. свести его к строке. После чего его можно будет сохранить в Cookies или в базе данных. Предположим, что нам нужно сохранить в Cookies несколько переменных. Чтобы не сохранять их все по отдельности, можно запаковать их в массив, затем его сериализовать и сохранить полученную строку как одну переменную.

Как только эти переменные нам понадобятся, мы получим из Cookies сохраненную строку, затем превратим ее в массив функцией `unserialize()` и распакуем массив в переменные.

## 11.3. Получение части массива

Функция `array_slice()` возвращает часть массива `array` длиной `length`, начиная с индекса `offset`:

```
array array_slice ( array array, int offset [, int length [, bool
preserve_keys]] )
```

Если параметр `length` не задан, то возвращаются все элементы массива, начиная с `offset`:

```
$input = array(1, 2, 3, 4, 5);

$output = array_slice($input, 0, 3);    // новый массив = 1, 2, 3
$output = array_slice($input, 2);       // новый массив = 3, 4, 5
$output = array_slice($input, -2, 1);    // новый массив = 4
```

Вы можете задать отрицательное смещение (`offset`) — отсчет будет начинаться с конца массива, а не с начала.

## 11.4. Функции автоматического заполнения массива

Функция `array_fill()` позволяет произвести заливку массива указанным количеством значений:

```
array array_fill ( int $start_index , int $num , mixed $value )
```

Первый параметр — это начальный индекс массива, второй параметр — количество значений, третий параметр — само значение. Заполним массив `$A` (начиная с индекса 0) значениями 5, всего 10 значений:

```
$A = array_fill(0, 10, 5);
```

Функция `array_fill_keys()` заполняет не только значениями, но и ключами. Ей нужно передать два параметра: массив с ключами, элементам которых будет присвоено значение — второй параметр функции:

```
array array_fill_keys ( array $keys , mixed $value )
```

Рассмотрим небольшой пример:

```
// определяем ключи:
$keys = array("a", "b", "c");
// заполняем массив, указываем ключи и значение 777:
$a = array_fill_keys($keys, 777);
```

Результат будет следующим:

```
Array
(
    [a] => 777
    [b] => 777
    [c] => 777
)
```

Функция `range()` позволяет создать массив, состоящий из диапазона чисел от `$low` до `$high`, шаг ряда чисел задается необязательным третьим параметром, который по умолчанию равен 1:

```
array range ( mixed $low , mixed $high [, number $step = 1 ] )
```

Пример:

```
$R = range(0, 100);
```

## 11.5. Сравнение массивов

Функция `array_diff()` сравнивает несколько массивов:

```
array array_diff ( array $array1 , array $array2 [, array $ ... ] )
```

Результат работы функции — разница между массивами. Рассмотрим небольшой пример:

```
$a1 = array("a" => "green", "red", "blue", "red");
$a2 = array("b" => "green", "yellow", "red");
$diff = array_diff($a1, $a2);
```

```
print_r($diff);
```

Функция `array_diff()` возвращает массив, состоящий из значений массива, заданного первым параметром, которые отсутствуют в любом другом массиве, перечисленном в последующих аргументах.

Функция `array_diff()` не учитывает ключи (индексы), поэтому ее целесообразно использовать для сравнения списков и обычных, неассоциативных массивов. В частности, результат предыдущего сравнения будет такой:

```
Array
(
    [1] => blue
)
```

Для сравнения ассоциативных массивов нужно использовать следующую функцию:

```
array array_diff_assoc ( array $array1 , array $array2 [, array $... ] )
```

Функция `array_diff_assoc()` производит дополнительную проверку ключей. Если сравнить массивы `$a1` и `$a2` из предыдущего примера, то результат будет таким:

```
Array
(
    [b] => brown
    [c] => blue
    [0] => red
)
```

Как видно по результату работы функции, она возвращает массив, состоящий из значений массива, заданного первым параметром, которые отсутствуют в любом другом массиве, который перечислен в последующих аргументах.

Для сравнения только ключей массивов используется функция `array_diff_key()`:

```
array array_diff_key ( array array1, array array2 [, array ...] )
```

Функция возвращает массив, содержащий все значения из массива, указанного в первом параметре, имеющие ключи, отсутствующие в последующих параметрах.

## 11.6. Полезные операции над массивом

### 11.6.1. Вычисление суммы и произведения всех элементов массива

Функция `array_sum()` возвращает сумму всех элементов массива. Понятно, что все элементы массива должны быть числами:

```
number array_sum ( array $array )
```

Пример:

```
$a = range(1,100);
echo array_sum($a);
```

Для вычисления произведения всех элементов массива служит функция `array_product()`:

```
echo array_product($a);
```

### 11.6.2. Проверка существования элемента в массиве

Функция `in_array()` возвращает `true`, если элемент `$needle` присутствует в массиве `$haystack`:

```
bool in_array ( mixed $needle , array $haystack [, bool $strict ] )
```

Пример:

```
$a = array("Linux", "Mac OS", "Windows");
if (in_array("BSD", $a)) echo "Элемент BSD находится в массиве";
```

В результате выполнения PHP-кода ничего не будет выведено, поскольку элемента `BSD` в массиве `$a` нет.

Функция `in_array()` существенно сокращает ваш PHP-код — вам больше не нужно возиться с операторами `foreach()` и `for()` для перебора всех элементов массива и сравнения их с эталоном.

Функция `in_array()` производит поиск по значениям, а функция `array_key_exists()` — по ключам, что позволяет проверить существование элемента с указанным ключом:

```
$a= array('name' => 'denis', 'password' => '123');
if (array_key_exists('name', $a)) {
    echo "Элемент 'name' существует в массиве";
}
```

### 11.6.3. Получение случайного элемента из массива

Функция `array_rand()` получает случайный элемент из массива `$input`:

```
mixed array_rand ( array $input [, int $num_req = 1 ] )
```

Второй параметр задает количество случайных элементов. Функция `array_rand()` возвращает ключ случайного элемента. Если вы указали `$num_req` больше 0, то будет возвращено несколько ключей. При попытке указать больше ключей, чем есть вообще элементов в массиве, вы получите ошибку уровня `E_WARNING`. Небольшой пример:

```
// генерируем массив
$a = range(0,100);
// получаем два случайных элемента
$rkeys = array_rand($a, 2);
// выводим эти элементы
echo $input[$rkeys[0]] . "<br>";
echo $input[$rkeys[1]] . "<br>";
```

Ранее перед вызовом функции `array_rand()` нужно было инициализировать генератор случайных чисел, но, начиная с версии PHP 4.2, это больше делать не нужно.

### 11.6.4. Удаление дубликатов из массива

Функция `array_unique()` удаляет повторяющиеся элементы из массива:

```
array array_unique ( array $array [, int $sort_flags = SORT_STRING ] )
```

Функция не обращает внимания на ключи массива и не изменяет переданный ей массив, а работает с его копией:

```
$a = array(1, 2, 3, 3, 2, 1);
$unique_a = array_unique($a);
```

В массиве `$unique_a`, как несложно было догадаться, окажутся значения 1, 2, 3.

### 11.6.5. Получение значений и ключей массива

Функция `array_values()` возвращает все значения массива. Эту функцию удобно использовать для преобразования ассоциативного массива в обычный список. Рассмотрим пример:

```
$array = array("name" => "guest", "password" => "empty");  
print_r(array_values($array));
```

Результат будет следующим:

```
Array  
(  
    [0] => guest  
    [1] => empty  
)
```

Как видите, все ключи ассоциативного массива были удалены, вместо них значениям массива были присвоены числовые индексы.

Кроме функции `array_values()` есть похожая функция — `array_keys()`. Как вы уже догадались, эта функция возвращает все ключи массива в виде списка. Вот результат ее работы:

```
Array  
(  
    [0] => name  
    [1] => password  
)
```

### 11.6.6. Замена местами значений и ключей

Функция `array_flip()` позволяет поменять местами значения и ключи. Проще всего рассмотреть ее работу на примере:

```
$a = array("a"=>1, "b"=>2);  
print_r(array_flip($a));
```

Результат будет таким:

```
Array  
(  
    [1] => a  
    [2] => b  
)
```

### 11.6.7. Подсчет значений в массиве

Представим, что у нас есть массив, значения которого могут повторяться. Функция `array_count_values()` позволяет подсчитать количество повторов значений в массиве:

```
$array = array("denis", "hello", 1, 1, "hello");  
print_r(array_count_values($array));
```

Результат будет следующим:

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

### 11.6.8. Замена в массиве

Функция `array_replace()` заменяет значения элементов первого массива значениями из последующих массивов с такими же ключами:

```
array array_replace ( array &$amp;array , array &$amp;array1 [, array &$amp;array2 [,  
array &$amp;... ] ] )
```

Рассмотрим пример работы этой функции:

```
$a = array("audi", "bmw", "mercedes");  
$r = array(0 => "zaz");
```

```
$a = array_replace($a, $r);  
print_r($a);
```

Нумерация элементов списка, как мы уже знаем, начинается с 0, — следовательно, элементу "audi" был присвоен индекс 0. Второй массив, `$r`, состоит всего из одного элемента с ключом 0 и значением "zaz". При вызове функции `array_replace` в массиве `$a` был найден только один элемент с ключом 0. Его значение было заменено значением единственного элемента массива `$r`:

```
Array  
(  
    [0] => zaz  
    [1] => bmw  
    [2] => mercedes  
)
```

### 11.6.9. Поиск в массиве

Функция `array_search()` производит поиск указанного значения в массиве и в случае успеха возвращает его индекс (ключ):

```
mixed array_search ( mixed $needle , array $haystack [, bool $strict ] )
```

Если указанное значение не найдено, функция возвращает `false`. Учитывая, что булево значение `false` соответствует небулевому значению 0, результат работы функции лучше анализировать оператором `===`.

Пример использования функции:

```
$a = array("audi", "bmw", "mercedes");
$search_result = array_search("zaz", $a);

if ($search_result === false)
echo "Запрашиваемый элемент в массиве не найден";
else echo "Ключ запрашиваемого элемента: $search_result";
```

## 11.6.10. Прогулка по массиву

Функция `array_walk()` позволяет применить пользовательскую функцию к каждому элементу массива. Использование этой функции удобнее, чем вызов вашей функции в цикле `for` или `foreach`. Прототип функции:

```
bool array_walk ( array &$amp;array , callback $funcname [, mixed $userdata ] )
```

Нашей функции будет передано два параметра: значение элемента и ключ элемента. Вы можете указать третий параметр, `$userdata`, который будет передан функции в качестве третьего параметра, если это вам нужно.

Напишем функцию, выводящую переданный ей элемент:

```
function my_print($item, $key) {
echo "$key = $item <br>";
}
```

«Прогуляемся» по массиву:

```
$a = array("audi", "bmw", "mercedes");
array_walk($a, 'my_print');
```

Вывод будет таким:

```
0 = audi
1 = bmw
2 = mercedes
```

Из главы 9 мы узнали, что PHP поддерживает многомерные массивы, т. е. такие, в которых значением элемента массива также является массив. Для «прогулки» по многомерному массиву используется рекурсивная версия функции `array_walk()`:

```
bool array_walk_recursive ( array &$amp;input , callback $funcname [, mixed $userdata ] )
```

Результатом работы функции является либо `true`, либо `false`.

Иногда полезно собирать результаты работы пользовательской функции. Например, мы напишем функцию вычисления квадрата числа, «схорим» ей массив с числами, функция пройдет по каждому элементу и вернет его квадрат. Теперь нам нужно «собрать» результаты для каждого вызова функции. Я догадываюсь, что вы знаете, как реализовать это средствами PHP, но есть способ проще — функция `array_map()`. Лучше продемонстрировать ее работу на примере.

Первым делом напишем функцию `asqr()`:

```
function asqr($n)
{
    return($n * $n);
}
```

В ней нет ничего сложного. Далее вызовем функцию `array_map()` для массива `$a`, результат запишем в массив `$result`:

```
$a = array(1, 2, 3);
$result = array_map("asqr", $a);
print_r($result);
```

Массив `$result` будет следующим:

```
Array
(
    [0] => 1
    [1] => 4
    [2] => 9
)
```

Каждый элемент нового массива является квадратом соответствующего ему элемента старого массива.



# **I. РАЗДЕЛ 4**

## **Функции в PHP**

<b>Глава 12.</b>	Полезные стандартные функции. Работа с датой
<b>Глава 13.</b>	Функции для работы со строками
<b>Глава 14.</b>	Работаем с файлами и каталогами
<b>Глава 15.</b>	Вывод графических изображений средствами PHP
<b>Глава 16.</b>	Работа с сетевыми сокетами в PHP. Сетевые функции
<b>Глава 17.</b>	Собственные функции



## ГЛАВА 12

# Полезные стандартные функции. Работа с датой

### 12.1. Генератор случайных чисел

В веб-программировании довольно часто используется генератор случайных чисел. Хотите вывести случайную картинку из галереи или фразу дня? А может, вам нужно выбрать случайный баннер из рекламной кампании? Все это можно сделать, получив случайный номер, соответствующий отображаемому объекту, а затем отобразив сам объект. Получить случайный номер можно с помощью функции

```
mt_rand():
```

```
int mt_rand ([int min, int max])
```

Значения `min` и `max` задают диапазон чисел, из которого будет случайно выбрано число. Следующий код генерирует пять случайных чисел:

```
for ($i=0; $i < 5; $i++)  
    echo mt_rand(0, 1000) . " ";
```

Напишем сценарий, выводящий фразу дня (message of the day — сокращенно `motd`). Создайте два файла: `motd.php` и `motd.txt`. Первый файл — это наш сценарий, код которого представлен в листинге 12.1, а второй — обычный текстовый файл, в котором записано несколько фраз, по одной в каждой строке.

#### Листинг 12.1. Сценарий `motd.php`

```
<?php
```

```
// читаем файл motd.txt в массив  
$motd = file('motd.txt');  
// количество элементов массива  
$max = count($motd);
```

```
// получаем случайное число  
$i = mt_rand(0, $max);
```

```
// выводим случайную фразу
echo $motd[$i];

?>
```

### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *12-1.php*), представленный в каталоге *Glava\_12* сопровождающего книгу электронного архива (см. приложение 4).

## 12.2. Дата и время

### 12.2.1. Кратко о timestamp

Время в UNIX (следовательно, и в PHP) представляется в формате timestamp. Timestamp — это количество секунд, прошедших с полуночи 1 января 1970 года. Такой формат не очень понятен пользователям, зато весьма удобен для компьютера и самого программиста. Вы можете очень легко сравнивать даты, переводить дату на определенное число дней, месяцев или лет — вам нужно всего лишь добавить (или вычесть) определенное количество секунд.

Чтобы вам в дальнейшем было проще оперировать с датами, можно использовать данные табл. 12.1.

**Таблица 12.1.** Некоторые единицы времени в секундах

Время	Секунды
одна минута	60
один час	3600
один день	86 400
одна неделя	604 800

Получить текущее время (точнее, текущий timestamp) можно функцией `time()`:

```
echo time();
```

Если ваш сценарий выполняется больше 1 секунды, желательно сохранить значение timestamp в переменной, иначе при следующем вызове `time()` (через 1 секунду) это значение уже изменится:

```
$current_time = time();
```

### 12.2.2. Функции *strtotime()* и *checkdate()*

Если вы хотите получить timestamp даты из прошлого или будущего (а не текущей даты), вам надо использовать функцию `strtotime()`. В качестве значения этой функции нужно передать строку, описывающую дату. В табл. 12.2 представлены примеры этой функции.

Функция `strtotime()` довольно удобна — вам не надо высчитывать timestamp вручную, а нужно только указать дату или ее смещение. Конечно, если вы не знаете английского, то вам будет полезно запомнить названия дней недели:

- ❑ Monday — понедельник;
- ❑ Tuesday — вторник;
- ❑ Wednesday — среда;
- ❑ Thursday — четверг;
- ❑ Friday — пятница;
- ❑ Saturday — суббота;
- ❑ Sunday — воскресенье.

Если дата указана неправильно, функция вернет `-1`. Первое, что приходит в голову, — использовать эту функцию для проверки даты, введенной пользователем. И если функция вернет `-1`, значит, дата неправильная. Но это очень плохая идея — если пользователь введет `yesterday` или `now`, то для функции это будет корректная дата. Так что для проверки даты лучше воспользоваться функцией `checkdate()`:

```
bool checkdate (int month, int day, int year)
```

Дата считается корректной (функция возвращает `true`), если год находится в диапазоне от 1901 до 32 767, месяц — от 1 до 12, день — в зависимости от выбранного месяца (функция также учитывает високосные годы).

Таблица 12.2. Примеры функции `strtotime()`

Функция <code>strtotime()</code>	Значение <code>timestamp</code>
<code>strtotime("December 31, 2008")</code>	31 декабря 2008 г. (полночь, поскольку время не указано)
<code>strtotime("2008-12-31")</code>	31 декабря 2008 г. (полночь, поскольку время не указано)
<code>strtotime("Friday 12:12")</code>	Пятница, 12:12
<code>strtotime("+1 day")</code>	Следующий день (к текущему <code>timestamp</code> будет добавлено 86 400 секунд)
<code>strtotime("+1 week")</code>	Неделя спустя с текущего момента (к текущему <code>timestamp</code> будет добавлено 604 800 секунд)
<code>strtotime("-3 months")</code>	Три месяца назад (с текущего момента)
<code>strtotime("next Monday")</code>	Следующий понедельник
<code>strtotime("last Monday")</code>	Прошлый понедельник
<code>now</code>	Текущая дата

12.2.3. Вывод даты

Теперь рассмотрим функцию `date()`, выводящую дату, отформатированную в соответствии с заданной строкой формата:

```
string date (string format [, int timestamp])
```

Первый параметр — это формат даты, второй — timestamp. Если второй параметр не задан, выводится текущее время. Например:

```
$d = 1216790861;
echo date("d.m.Y", $d);           // выведет 23.07.2008
```

В строке формата вы можете использовать любые символы, кроме символов — модификаторов формата (в нашем случае: d, m, Y). Если вам нужно вывести символ, который является модификатором формата, его надо экранировать с помощью слеша — например: \d\a\y:

```
echo date("Сегодня d.m.Y");
echo date("\d\a\y d ");
```

Модификаторы формата функции date() представлены в табл. 12.3.

Таблица 12.3. Модификаторы формата функции date()

Модификатор	Описание	Диапазоны/примеры возвращаемых значений
g	Час, 12-часовой формат, без лидирующего нуля	1–12
G	Час, 24-часовой формат, без лидирующего нуля	0–23
h	Час, 12-часовой формат, с лидирующим нулем	01–12
H	Час, 24-часовой формат, с лидирующим нулем	00–23
a	До/после полудня (нижний регистр)	am или pm
A	До/после полудня (верхний регистр)	AM или PM
i	Минуты с лидирующим нулем	00–59
s	Секунды с лидирующим нулем	00–59
d	День месяца, две цифры с лидирующим нулем	01
D	Текстовое представление дня (трехбуквенная аббревиатура)	Mon
j	День месяца (1 или 2 цифры)	1
l	Название дня недели (полное)	Monday
w	Номер дня недели	0 (Вс) до 6 (Сб)
z	Номер дня года	0–365
W	Номер недели в году	42
F	Полное название месяца	December
M	Короткое название месяца (трехбуквенное)	Dec
m	Номер месяца с лидирующим нулем	01
n	Номер месяца без лидирующего нуля	1
t	Количество дней в текущем (или заданном, если указан timestamp) месяце	от 28 до 31

Таблица 12.3 (окончание)

Модификатор	Описание	Диапазоны/примеры возвращаемых значений
L	Високосный ли год	1, если год високосный, 0 — в противном случае
Y	Год, четыре разряда	1999 или 2008
y	Год, два разряда	99 или 08

12.2.4. Использование *type="date"*

Очень часто начинающие программисты, разрабатывающие также и формы ввода, допускают одну и ту же ошибку: они начинают «изобретать велосипед», а именно — поле выбора даты. В крупных компаниях, где разработкой фронтенда занимаются отдельные специалисты, такая ошибка исключена, поскольку специалисты по фронтенду знают, что они делают. Начинающие же PHP-программисты пытаются выдумывать свои различные способы ввода даты. Кто-то просто использует текстовое поле, кто-то — прикручивает какой-то JavaScript-сценарий, отображающий календарь, кто-то — пишет такой сценарий самостоятельно.

К сожалению, многие забывают о стандартном типе "date" для текстового поля input:

```
<input type="date" name="dt">
```

Этот тип поддерживается всеми современными браузерами и создает поле выбора даты, возвращающее строку формата ГГГГ-ММ-ДД, — как раз в таком формате, который принимает СУБД MySQL. Очень удобно — не нужно никаких преобразований строки, содержащей дату, перед ее помещением в MySQL (рис. 12.1).

Использование *type="date"* не освобождает вас от проверки строки на правильность даты. Ведь злоумышленники могут напрямую передать сценарию в качестве значения поля формы все что угодно — даже SQL-код. Поэтому сначала нужно выполнить проверку строки на ее корректность, а затем уже вставлять в базу данных.

Дополнительная информация доступна по адресу: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/date>.

12.2.5. Функция *checkdate()*:  
проверка даты на корректность

Представим, что у нас есть дата, введенная пользователем. Нужно проверить ее на корректность. Это может быть просто некорректная дата, например: 2018-02-31 (в феврале не может быть 31 день), а может быть вообще не дата — например: hello.

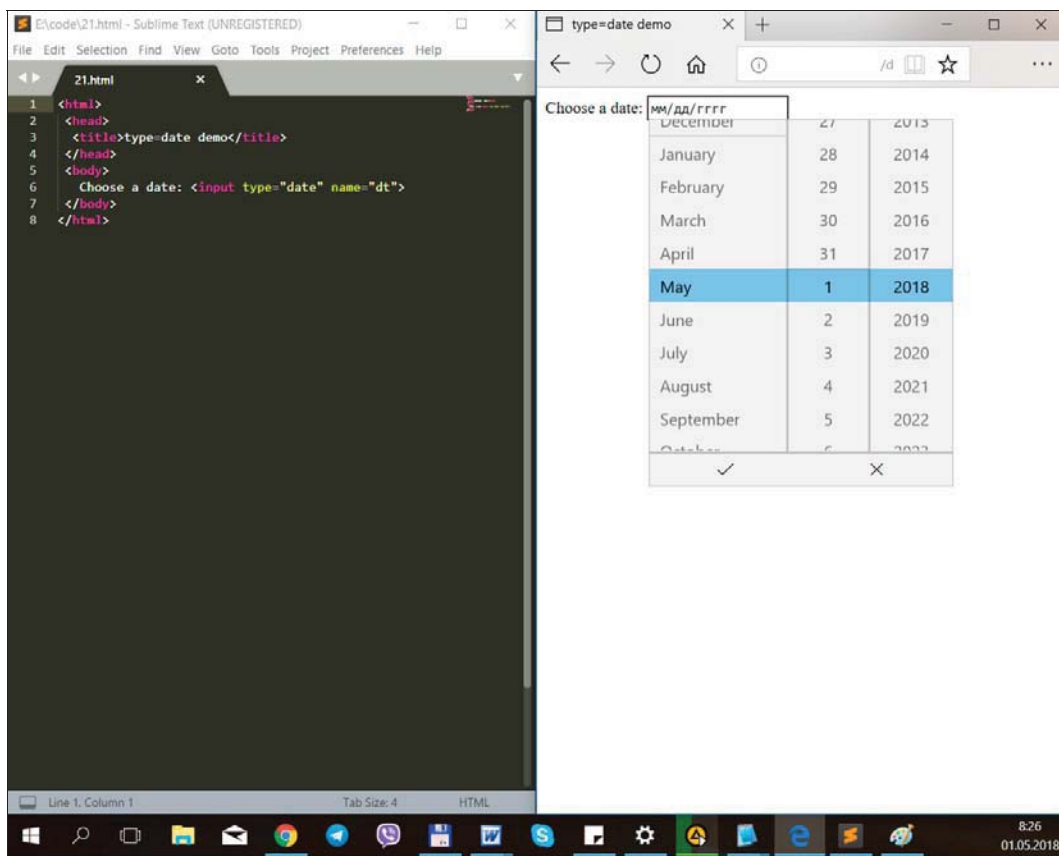


Рис. 12.1. Поле выбора даты в браузере Microsoft Edge

Первое, что приходит в голову, — проверить, соответствует ли строка формату ГГГГ-ММ-ДД. Это можно легко сделать, например, с помощью регулярных выражений. Но поставленную задачу это не решит, т. к. приведенная в качестве примера дата 2018-02-31 вполне соответствует формату, но ее нельзя назвать правильной.

Да и что делать, если формат даты поменяется и у вас возникнет необходимость работы с датой в другом формате?

Но пока не будем усложнять, а представим, что-таки нам нужен только формат ГГГГ-ММ-ДД. Проверить введенную дату на соответствие этому формату, как уже отмечалось, можно с помощью регулярных выражений — это хорошо. Далее можно разбить строку на подстроки и получить три строки с годом, месяцем и днем соответственно. Следующий шаг — передать функции `checkdate()` полученные переменные:

```
checkdate($month, $day, $year)
```

Эта функция проверит корректность даты по григорианскому календарю. Таким образом, проверка даты у нас получается трехэтапная:

- ❑ проверка на соответствие формату;
- ❑ разделение даты на компоненты;
- ❑ проверка корректности функцией `checkdate()`.

На мой взгляд, решение довольно сложное и малоэффективное — если нужно будет изменить формат даты (например, чтобы месяц был прописью), этот вариант уже не сработает.

То есть нам нужен универсальный вариант, который бы работал с датой и временем самых разных форматов. Для этого мы воспользуемся конструкцией `DateTime::createFromFormat()`. Методу `createFromFormat()` здесь передается формат и дата, которая должна соответствовать заданному формату.

Напишем функцию `validateDate()`:

```
function validateDate($date, $format = 'Y-m-d H:i:s')
{
    $d = DateTime::createFromFormat($format, $date);
    return $d && $d->format($format) == $date;
}
```

Проверим, как работает функция:

```
var_dump(validateDate('2023-02-28 12:12:12')); # true
var_dump(validateDate('2023-02-30 12:12:12')); # false
var_dump(validateDate('2023-02-28', 'Y-m-d')); # true
var_dump(validateDate('28/02/2023', 'd/m/Y')); # true
var_dump(validateDate('30/02/2023', 'd/m/Y')); # false
var_dump(validateDate('14:50', 'H:i')); # true
var_dump(validateDate('14:77', 'H:i')); # false
var_dump(validateDate(14, 'H')); # true
var_dump(validateDate('14', 'H')); # true
```

Наконец-то мы получили универсальное решение, способное работать с датой и временем в любом формате.

## 12.2.6. Класс *DateTime*: удобная работа с датой и временем

С одной стороны, мы еще не подошли к рассмотрению такой важной темы, как объектно-ориентированное программирование (ООП). С другой — класс `DateTime` является встроенным в PHP, и ему место среди стандартных функций. Поэтому пока просто используйте приведенные фрагменты кода как есть, а в *главе 24* мы рассмотрим ООП в PHP.

### Создание даты по строке

Класс `DateTime` представляет собой объектно-ориентированный интерфейс для работы с датой и временем. Поскольку это ООП, то нужно создать *объект* типа `DateTime`. Делается это так:

```
$dt = new DateTime();
```

Если вы не передаете в конструктор аргумент, то будет создан экземпляр класса, представляющий текущую дату и время. Если нужно создать экземпляр с определенной датой/временем, тогда в конструктор нужно передать строку, представляющую эту дату/время. О формате строк, представляющих дату и время, вы можете прочитать в документации: <https://www.php.net/manual/en/datetime.formats.php>.

Пример:

```
$dt = new DateTime('2022-03-27 23:12');
```

При желании можно указать и формат, и строку в этом формате:

```
$datetime = DateTime::createFromFormat('M j, Y H:i:s', 'Mar 5, 2023 22:10:11');
```

При создании объекта `DateTime` можно указывать часовой пояс. Делается это так:

```
$timezone = new DateTimeZone('Europe/London');  
$datetime = new DateTime('2023-08-20', $timezone);
```

## Вычисление количества дней между двумя датами

Иногда нужно вычислить количество дней между двумя датами — например, между текущей датой и датой определенного события. Для этого можно использовать класс `DateTime`. Пример:

```
$date1 = new DateTime("2023-07-06");  
$date2 = new DateTime("2023-07-09");  
$diff = $date2->diff($date1)->format("%a");
```

Можно использовать и временные метки. Для начала надо преобразовать даты во временные метки, а затем уже подсчитать количество дней между ними (разделив разницу между метками на 86 400):

```
$start = strtotime('2023-01-25');  
$end = strtotime('2023-02-20');  
$days_between = ceil(abs($end - $start) / 86400);
```

Если между датами разница в несколько лет, то приведенный пример даст только количество дней — например, 1240 или 1560. Обычному пользователю это ничего не скажет, поскольку ему придется в уме делить полученное число на 365, а затем вычислять месяцы по остатку. Это не очень удобно. Было бы хорошо, если бы наш сценарий сам показывал количество лет, месяцев, дней между датами:

```
$datetime1 = new DateTime("2022-07-20");  
$datetime2 = new DateTime("2023-07-22");  
$difference = $datetime1->diff($datetime2);  
echo 'Разница: '.$difference->y.' years, '  
      .$difference->m.' months, '  
      .$difference->d.' days';  
print_r($difference);
```

Вывод сценария будет таким:

```
Difference: 1 years, 0 months, 2 days  
DateInterval Object
```

```
(
    [y] => 1
    [m] => 0
    [d] => 2
    [h] => 0
    [i] => 0
    [s] => 0
    [weekday] => 0
    [weekday_behavior] => 0
    [first_last_day_of] => 0
    [invert] => 0
    [days] => 367
    [special_type] => 0
    [special_amount] => 0
    [have_weekday_relative] => 0
    [have_special_relative] => 0
)
```

## Работа с временными интервалами

Класс `DateInterval` предназначен для манипуляций экземплярами класса `DateTime`. Экземпляр `DateInterval` представляет собой фиксированный отрезок времени — например: "two weeks" или относительное время "yesterday". Также этот класс представляет методы `add()` и `sub()` для манипуляции значениями `DateTime`.

Рассмотрим небольшой пример:

```
// Задаем начальное время
$datetime = new DateTime('2017-02-02 12:00:00');
// Двухнедельный интервал
$interval = new DateInterval('P2W');
// Добавляем 2 недели к исходной дате и получаем результат
$datetime->add($interval);
echo $datetime->format('Y-m-d H:i:s');
```

Интервал может содержать следующие буквы:

- ☐ Y — годы;
- ☐ M — месяцы;
- ☐ D — дни;
- ☐ W — недели;
- ☐ H — часы;
- ☐ M — минуты;
- ☐ S — секунды.

Например, `P2W` означает 2 недели, а `P2DT3H2M` — 2 дня, три часа и 2 минуты. Значение времени отделяется буквой `T` (если оно включено в интервал).

Иногда нужно узнать количество дней в месяце. Например, чтобы произвести запрос вроде такого:

```
$q = "select * from fuel where (dt >= \"\$date_from\" and dt <= \"\$date_to\")";
```

В этом случае на помощь придет функция `cal_days_in_month()`:

```
$daysInMonth = cal_days_in_month(CAL_GREGORIAN, $month, $year);
```

Ей нужно передать три параметра: тип календаря (в нашем случае — `CAL_GREGORIAN`), месяц и год. Функция вернет количество дней в месяце в заданном году (функция учитывает високосные годы).

## 12.2.7. Настройка PHP для корректной работы с датами

Для корректной работы API необходимо в файле `php.ini` установить ваш часовой пояс:

```
date.timezone = 'Europe/London';
```

Если этого не сделать, PHP покажет предупреждение. Также можно установить часовой пояс из сценария, когда нет возможности редактировать `php.ini`:

```
date_default_timezone_set('Europe/London');
```

Список часовых поясов доступен по адресу:

<http://www.php.net/manual/en/timezones.php>.

## 12.3. Математические функции

Стихия PHP — обработка строк и работа с базами данных, но никак не математические вычисления. Я не утверждаю, что PHP не пригоден для математики, наоборот, у него есть все необходимые (стандартные) математические функции (табл. 12.4), чтобы вам не пришлось изобретать велосипед заново. Однако будете ли вы ими пользоваться? Решать вам!

**Таблица 12.4.** Некоторые математические функции PHP

Функция	Описание
<code>acos(float \$x)</code>	Арккосинус $x$
<code>asin(float \$x)</code>	Возвращает арксинус $x$
<code>atan(float \$x)</code>	Арктангенс аргумента $x$
<code>cos(float \$x)</code>	Возвращает косинус $x$
<code>exp(float \$x)</code>	Возвращает экспоненту (2,71828182845) в степени $x$
<code>log(float \$x)</code>	Возвращает натуральный логарифм $x$
<code>pi()</code>	Возвращает число $\pi$
<code>pow(float \$x, \$a)</code>	Возвращает $x$ в степени $a$
<code>sin(float \$x)</code>	Возвращает синус аргумента $x$
<code>sqrt(float \$x)</code>	Возвращает квадратный корень $x$ . Напомню, что $x$ должен быть положительным
<code>tan(float \$x)</code>	Тангенс $x$

Все функции, представленные в табл. 12.4, относятся к расширению Math. Посмотреть список всех функций этого расширения можно по ссылке: <http://www.php.ru/manual/ref.math.html> (страница руководства на русском языке).

Но это не единственное математическое расширение PHP. Есть еще статистические функции: <http://www.php.ru/manual/ref.stats.html>.

Также есть расширение GMP, позволяющее производить вычисления с повышенной точностью: <http://www.php.ru/manual/ref.gmp.html>.

Для вычислений с произвольной точностью используются функции расширения BC Math: <http://www.php.ru/manual/ref.bc.html>.

#### **ПРИМЕЧАНИЕ**

В PHP 7/8 целые числа в 64-битных сборках для Windows теперь представляются в 64-битном виде, а не в 32-битном, как раньше. Тогда использование 64-битных сборок было бессмысленным занятием, если требовалось работать с большими числами.



## ГЛАВА 13

# Функции для работы со строками

### 13.1. Основные строковые функции

В *главе 3* мы рассмотрели основные операции над строками. Сейчас мы поговорим о функциях обработки строк.

К базовым строковым функциям относят следующие функции:

- ☐ `strlen()` — возвращает длину строки;
- ☐ `strpos()` — позволяет найти подстроку в строке;
- ☐ `str_contains()` — позволяет проверить, есть ли подстрока в строке (появилась в PHP 8);
- ☐ `str_starts_with()` — проверяет, начинается ли наша строка с заданной подстроки (PHP 8);
- ☐ `str_ends_with()` — проверяет, заканчивается ли наша строка заданной подстрокой (PHP 8);
- ☐ `strstr()` — возвращает первое вхождение подстроки в строку;
- ☐ `strchr()` — является псевдонимом для функции `strstr()`;
- ☐ `substr()` — возвращает подстроку;
- ☐ `strcmp()` — сравнивает две строки;
- ☐ `strncmp()` — сравнивает первые `n` символов строки;
- ☐ `strncasecmp()` — сравнивает первые `n` символов без учета регистра символов;
- ☐ `strcasecmp()` — сравнивает две строки без учета регистра символов;
- ☐ `strnatcmp()` — сравнивает две строки, используя алгоритм натуральной сортировки;
- ☐ `strnatcasecmp()` — то же, что и `strnatcmp()`, но без учета регистра символов;
- ☐ `str_repeat()` — используется для повтора строки;
- ☐ `str_replace()` — производит замену в строке;

- `strspn()` — определяет наличие начальных символов в строке;
- `strcspn()` — определяет отсутствие начальных символов в строке.

Рассмотрим все эти функции по порядку.

- Первой функции нужно передать всего лишь один параметр — строку:

```
$str = "string";  
echo strlen($str); // выведет 6
```

- Функции `strpos()` надо передать три параметра: основную строку, искомую подстроку и смещение — номер символа основной строки, с которого следует начинать поиск подстроки:

```
int strpos (string $haystack, string $needle [, int offset])
```

Если подстрока не найдена, функция возвращает `false`, а если найдена, то первую позицию вхождения подстроки в строку. Пример использования функции:

```
$pos = strpos($my_string, $find);  
  
if ($pos === false) {  
    echo "Строка '$find' не найдена в строке '$my_string'";  
} else {  
    echo "Строка '$find' найдена в строке '$my_string' в позиции $pos";  
}
```

Функция `strpos()` чувствительна к регистру символов. Если регистр символов для вас не важен, лучше использовать функцию `stripos()`, которая является полным аналогом `strpos()`, за исключением того, что игнорирует регистр символов.

Практически всем практикующим программистам ранее приходилось использовать функцию `strpos()` не по назначению. Предназначена она для поиска позиции подстроки в строке, а применялась в основном для проверки, входит ли указанная подстрока в строку. В PHP 8 наконец-то появилась функция `str_contains()`, позволяющая проверить наличие подстроки `$needle` в строке `$haystack`:

```
str_contains ( string $haystack , string $needle ) : bool
```

Использовать эту функцию стало гораздо удобнее, чем `strpos()`. Применяя ее, помните, что если мы ищем пустую подстроку, то функция всегда будет возвращать `true`:

```
str_contains("Hello World!", "Hello"); // true  
str_contains("Hello World!", "Foo"); // false  
str_contains("abc", ""); // true  
str_contains("", ""); // true
```

Также нужно понимать, что эта функция тоже является регистрозависимой:

```
str_contains("Hello world!", "hello"); // false
```

Функции `str_starts_with()` и `str_ends_with()` аналогичны ей, но проверяют наличие строки `$needle` в начале и в конце (соответственно) строки `$haystack`:

```
str_starts_with ( string $haystack , string $needle ) : bool
str_ends_with ( string $haystack , string $needle ) : bool
```

- ❑ Функция `strrpos()` находит позицию последнего вхождения подстроки `$needle` в строку `$haystack`. Функция `stripos()` делает то же самое, но без учета регистра:

```
int strrpos(string $haystack, string $needle [, int $offset = 0 ])
int stripos(string $haystack, string $needle [, int $offset = 0 ])
```

- ❑ Функция `strstr()` предназначена для поиска первого вхождения подстроки `needle` в строку `$haystack`:

```
string strstr (string $haystack, mixed $needle [, bool $before_needle = false ])
```

Если третий параметр равен `true`, то функция возвращает часть строки `$haystack`, находящуюся до строки `$needle`. Третий параметр у функции `strstr()` появился в PHP версии 5.3.0. Рассмотрим небольшой пример:

```
$email = 'dm@example.com';
$domain = strstr($email, '@');
echo $domain;           // выведет @example.com

// только в PHP 5.3.0 или выше
$user = strstr($email, '@', true);
echo $user;             // выведет dm
```

Функция `strstr()` является регистрозависимой. Если вам нужно игнорировать регистр символов, тогда используйте функцию `stristr()`, параметры у нее такие же, как у функции `strstr()`. Обе функции возвращают `false`, если подстрока `$needle` не найдена.

Функции `strstr()` и `stristr()` удобно использовать для определения наличия в строке какого-нибудь образца, например:

```
// если в строке $v есть подстрока "UNION",
// прекратить работу сценария:
if (stristr($v, "UNION") !== false) die();
```

- ❑ Функция `substr()` возвращает подстроку указанной строки `string` длиной `length`, начиная с позиции `start`:

```
string substr (string $string, int $start [, int $length])
```

Если последний параметр (длина) не указан, то возвращается подстрока, начиная с позиции `start` и до конца строки. Пример:

```
$sub = substr("123456", 2); // возвращает "3456"
```

- ❑ Для сравнения строк служит функция `strcmp()`:

```
int strcmp (string $str1, string $str2)
```

Функции нужно передать две сравниваемые строки. Функция возвращает следующие значения:

- 0 — строки равны;
- 1 — первая строка «больше» (лексикографически), чем вторая;
- -1 — первая строка «меньше», чем вторая.

Функция `strcmp()` учитывает регистр символов. Если вам нужно сравнивать строки без учета регистра, используйте функцию `strcasecmp()`. Ее параметры и возвращаемые значения такие же, как у `strcmp()`.

- ❑ Функция `strncmp()` сравнивает первые `n` символов строки (учитывается регистр символов):

```
int strncmp (string $str1 , string $str2 , int $len)
```

Первые два параметра — как у обычной `strcmp()`, второй — те самые `n` символов. Возвращаемые значения аналогичны функции `strcmp()`.

- ❑ Аналог функции `strncmp()`, но без учета регистра символов — функция `strncasecmp()`.

В главе 10, рассматривая функции для сортировки массивов, мы познакомились с алгоритмом натуральной сортировки. Функции `strnatcmp()` и `strnatcasecmp()` сравнивают строки, используя этот алгоритм:

```
int strnatcmp (string $str1 , string $str2)
int strnatcasecmp (string $str1 , string $str2)
```

Указанные функции возвращают те же значения, что и функция `strcmp()`. Функция `strnatcasecmp()` игнорирует регистр символов.

- ❑ Функция `str_repeat()` возвращает `n` повторов указанной строки:

```
string str_repeat (string $input, int $n)
```

Например:

```
echo str_repeat("1", 3);           // выведет 111
```

- ❑ Осуществить замену в строке можно с помощью функции `str_replace()`:

```
mixed str_replace (mixed $search, mixed $replace, mixed $subject [, int &count])
```

Эта функция производит замену подстроки `search` строкой `replace` в строке `subject`. Последний параметр задает количество замен, которые должна произвести функция. Обратите внимание, что этот параметр доступен, начиная с PHP версии 5, поэтому если у вас до сих пор PHP 4, то самое время обновиться!

Заметьте, что функция `str_replace()` не изменяет исходную строку `subject`, а работает с ее копией, а потом возвращает результат.

Функция `str_ireplace()` аналогична функции `str_replace()`, но игнорирует регистр символов.

□ Теперь рассмотрим функции `strpos()` и `strcspn()`:

```
int strpos(string str1, string str2)
int strcspn(string str1, string str2)
```

Первая функция определяет наличие начальных символов в строке и возвращает длину начального фрагмента строки `str1`, который полностью состоит из строки `str2`. Например:

```
$str = "function";
$substr_count = strpos($str, "func");
echo ($substr_count);           // выведет 4
```

Функция `strcspn()` возвращает длину начального фрагмента строки `str1`, который состоит полностью не из символов, которые есть в `str2`.

## 13.2. Специальные функции замены

При обработке данных, полученных от пользователя, удобно использовать следующие функции:

- `nl2br()` — заменяет все символы новой строки (`\n`) HTML-тегом `<br>`;
- `htmlspecialchars()` — заменяет специальные символы (вроде «больше», «меньше») их HTML-эквивалентами.

Этим функциям нужно передать только строку, в которой необходимо произвести замену, например:

```
$string = "Первая строка\nВторая строка";
echo nl2br($string);
```


```
$string2 = "5 < 10 > 2";
$output = htmlspecialchars($string2);
```

Указанные функции не изменяют исходную строку, а работают с ее копией. Для обратного преобразования строки, обработанной функцией `htmlspecialchars()`, служит функция `htmlspecialchars_decode()`:

```
string htmlspecialchars_decode (string $string [, int $quote_style = ENT_COMPAT ])
```

Начиная с PHP 5.4, функцию `htmlspecialchars()` существенно улучшили. Старая версия была немного «неинтуитивной». Например, если вы передавали функции неправильно закодированную строку, то функция возвращала просто пустую строку. Самое интересное: она выдавала ошибку, но только в том случае, если отображение ошибок было отключено. Да, именно отключено. Вроде бы логично — несмотря на то, что отображение ошибок отключено, мы все равно выводим сообщение об ошибке. Но на практике это означало, что на компьютере разработчика вы не видели ошибку (т. к. на нем включается, как правило, отображение всех ошибок), но зато ошибки «вылезали» при переносе проекта на реальный сервер.

В PHP 5.4 неправильное поведение функции исправлено. К тому же добавлены два аргумента, позволяющих выбрать альтернативу возвращаемой строке (для большей информативности):

- ☐ `ENT_IGNORE` — отбрасывает всю некорректную последовательность. Однако в этом случае вы не увидите недопустимых случаев, кроме того, это создает угрозу безопасности;
- ☐ `ENT_SUBSTITUTE` — вместо удаления недопустимых символов они будут заменены на символ Unicode  (U + FFFD).

Пример:

```
htmlspecialchars($str, ENT_SUBSTITUTE);
```

Функция `htmlentities()` служит для преобразования всех символов в HTML-эквиваленты:

```
string htmlentities (string $string [, int $flags = ENT_COMPAT [, string $charset [, bool $double_encode = true ]]])
```

Первый параметр функции — это сама строка, второй параметр может принимать следующие значения:

- ☐ `ENT_COMPAT` — конвертирует двойные кавычки и игнорирует одинарные;
- ☐ `ENT_QUOTES` — конвертирует и двойные, и одинарные кавычки;
- ☐ `ENT_NOQUOTES` — игнорирует кавычки;
- ☐ `ENT_IGNORE` — опция предназначена для обеспечения обратной совместимости со старыми версиями PHP. Применять ее не следует, поскольку она негативно влияет на безопасность вашего приложения.

Третий параметр — кодировка символов. Используйте следующие значения (тип значения — `string`, поэтому значение нужно указывать в кавычках):

- ☐ `"UTF-8"` — для кодировки UTF-8;
- ☐ `"cp1251"`, `"windows-1251"` или просто `"1251"` — для кодировки Windows-1251;
- ☐ `"KOI8-R"` — для кодировки KOI8-R;
- ☐ `"cp866"` — для кириллической DOS-кодировки.

Когда четвертый параметр выключен (значение `false`), PHP не будет кодировать существующие в строке HTML-эквиваленты. По умолчанию разрешается двойное преобразование.

Функция `html_entity_decode()` служит для обратного преобразования строки, закодированной с помощью функции `htmlentities()`, в первоначальный вид:

```
string html_entity_decode (string $string [, int $quote_style = ENT_COMPAT [, string $charset ]])
```

Назначение параметров у этой функции такое же, что и у `htmlentities()`.

## 13.3. Функции преобразования строки

К этой группе относятся следующие функции:

- `strip_tags()` — удаляет из строки HTML-теги;
- `stripslashes()` — удаляет из строки слешы;
- `stripslashes()` — удаляет из строки слешы, добавленные функцией `addslashes()`;
- `strtolower()` — переводит символы строки в нижний регистр;
- `strtoupper()` — переводит символы строки в верхний регистр;
- `strrev()` — «переворачивает» строку;
- `trim()` — удаляет пробельные символы до и после строки;
- `ltrim()` — удаляет пробельные символы в начале строки;
- `lcfirst()` — делает первый символ строки строчным (переводит его в нижний регистр);
- `rtrim()` — удаляет пробельные символы в конце строки;
- `chop()` — псевдоним для `rtrim()`;
- `str_shuffle()` — перемешивает символы строки в случайном порядке;
- `str_rot13()` — выполняет над строкой преобразование ROT13;
- `urlencode()` — кодирует строку для ее передачи в URL;
- `urldecode()` — декодирует строку, закодированную функцией `urlencode()`;
- `ucfirst()` — делает прописным первый символ заданной строки (переводит его в верхний регистр);
- `ucwords()` — делает заглавным первый символ каждого слова в строке;
- `wordwrap()` — разбивает текст согласно указанным параметрам.

Конечно, это далеко не все строковые функции, имеющиеся в арсенале PHP, но зато наиболее полезные, которые можно использовать в самых разнообразных ситуациях. Рассмотрим их далее по порядку.

- Функция `strip_tags()` удаляет из текста HTML-теги, оставляя обычный текст. Если передать функции HTML-страницу, то она вернет обычный текст без форматирования. К тому же эта функция с успехом защищает ваш сайт от вредоносного кода, который может передаваться в HTML-тегах (например, в тегах `OBJECT` или `SCRIPT`). Злоумышленник запросто может передать такой код, например, в качестве текста сообщения в гостевой книге. Впрочем, если вы хотите оставить некоторые безобидные теги — например, `BR`, то можете указать их в качестве второго параметра функции:

```
$html = "<h1>Заголовок<h1><p>Текст<br>Следующая строка";  
  
// $txt1 = "Заголовок<p>Текст<br>Следующая строка"  
$txt1 = strip_tags($html, "<p><br>");
```

```
// $txt2 = "ЗаголовокТекстСледующая строка"  
$txt2 = strip_tags($html);
```

- ❑ Функция `stripslashes()` проста в использовании — ей нужно передать всего один параметр — строку, из которой требуется удалить слешы:

```
string stripslashes (string $str)
```

Есть функции и обратные этой: `addslashes()` и `addcslashes()`. Вторая экранирует строку слешами, а первая — слешами в стиле C:

```
string addcslashes (string $str, string $charlist)  
string addslashes (string $str)
```

Первый параметр здесь — экранируемая строка. Параметр `charlist` у `addcslashes` задает символы, которые нужно экранировать, — будьте осторожны при задании этого диапазона.

- ❑ Функции `strtolower()` и `strtoupper()` мы подробно рассматривать не будем — они слишком просты. Всё, что здесь нужно, — это передать функции строку, регистр которой надо изменить. Как и остальные строковые функции PHP, эти функции не изменяют исходную строку. Если вам нужно изменить исходную строку, можно вызвать функцию так:

```
$str = strtolower($str);
```

- ❑ Функция `strrev()` служит для «переворачивания» строки:

```
string strrev (string $string)
```

Пример:

```
echo strrev("123"); // выведет 321
```

- ❑ Довольно часто приходится удалять пробелы в данных, оставленных пользователем в форме. Функции `trim()`, `ltrim()` и `chop()` удаляют из строки пробелы и пробельные символы (типа табуляции). Однако эти функции не панацея. Они помогут избавиться от пробелов, которые находятся в начале и/или в конце строки. Но они не в силах удалить лишние пробелы, находящиеся в середине строки, — например, здесь:

```
"Иван    Иванов"
```

- ❑ Функции `str_shuffle()` и `str_rot13()` очень просты — им нужно передать всего один параметр — строку, которую требуется преобразовать. Результат — преобразованная строка, исходная строка не изменяется.
- ❑ Если вам нужно сгенерировать URL, содержащий специальные символы (или хотя бы просто пробелы) или символы русского языка, вам лучше закодировать строку функцией `urlencode()`. В этом случае все символы заменяются последовательностью `%код`, а пробелы — символом `+`.
- ❑ Функция `urldecode()` выполняет обратное преобразование строки:

```
$s = "Изучаем PHP";  
echo urlencode($s);
```

Вывод будет следующим:

```
%C8%E7%F3%F7%E0%E5%EC+PHP
```

- Функция `ucfirst()` делает прописным первый символ переданной ей строки, а функция `ucwords()` — первый символ каждого слова в строке:

```
string ucfirst (string $str)
string ucwords (string $str)
```

Пример использования:

```
echo ucfirst("hello world"); // выведет Hello world
echo ucwords("hello world"); // выведет Hello World
echo lcfirst("Hello");      // выведет hello
```

- Предположим, что у нас есть довольно длинный текст (переменная `$txt`), который для удобства вывода/чтения можно разбить на части с помощью функции `wordwrap()`:

```
// разбиваем текст по 30 символов
$txt = wordwrap($txt, 30, "<br>");
```

Заметьте, что функция `wordwrap()` распознает слова и не разбивает их. Если же вы хотите вставлять указанный разделитель через заданное количество символов, тогда установите последний необязательный параметр в 1:

```
$txt = wordwrap($txt, 30, "<br>", 1);
```

## 13.4. Функции преобразования кодировок

Для преобразования кодировок русского языка служит функция `convert_cyr_string()`:

```
string convert_cyr_string (string $str, string $from, string $to)
// преобразование текста из кодировки KOI8-R в Windows-1251:
echo convert_cyr_string ($string, "k", "w");
```

Первый параметр — это строка, кодировку которой нужно преобразовать. Второй параметр — исходная кодировка, третий — кодировка назначения. Поддерживаются следующие кодировки:

- `k` — KOI8-R;
- `w` — Windows-1251;
- `i` — iso8859-5;
- `a` — x-cp866;
- `m` — x-mac-cyrillic.

Функция `convert_uuencode()` кодирует строку в формате `uuencode`. Функция `convert_uudecode()` декодирует строку из формата `uuencode` в обычный вид:

```
string convert_uencode (string $data)
string convert_udecode (string $data)
```

## 13.5. Функции для работы с отдельными символами строки.

### Разыменовывание строки

Вы можете получить доступ к отдельным символам строки с помощью квадратных скобок, например:

```
$nums = "654321";
echo $nums[1]; // выведет 5
```

Заметьте, что нумерация символов в строке начинается с 0, поэтому в приведенном примере и будет выведена цифра 5.

Функции `chr()` и `ord()` возвращают символ по заданному коду и код по заданному символу соответственно. Например:

```
echo chr(65); // выведет A
echo ord('A'); // выведет 65
```

В PHP 5.5 появилось еще одно сомнительное новшество, позволяющее сделать код компактнее, — разыменовывание строки. В целом разыменовывание строки выглядит так:

```
echo 'Привет'[0]; // Выведет "П"
```

Согласитесь, проще было бы просто вывести 'п'. Впрочем, немного это новшество себя оправдывает в функциях. Представим, что есть функция, возвращающая строку:

```
function getSt() {
    return "Привет";
}
```

Нам нужно обратиться к первому символу этой строки (с индексом 0). Если раньше приходилось сначала присвоить результат какой-нибудь строке, а уже потом обращаться к ее символам, то теперь мы можем обратиться к результату напрямую:

```
// Раньше
$temp = getSt();
echo $temp[0];
// В PHP 5.5
echo getSt()[0];
```

Ситуация с разыменовыванием строки такая же, как и с разыменовыванием массива.

## 13.6. Функция `md5()` и другие функции шифрования/хеширования. API хеширования паролей

Функция `md5()` кодирует заданную строку с помощью алгоритма MD5. Особенность этого алгоритма заключается в том, что закодированную строку нельзя обратно раскодировать. Алгоритм MD5 часто используется для кодирования паролей.

Как же потом проверить, правильный ли пароль ввел пользователь, если зашифрованную строку нельзя расшифровать? Все очень просто — вам нужно закодировать с помощью функции `md5()` принятый от пользователя пароль, а затем сравнить две закодированные строки, например:

```
$pass = "5ebe2294ecd0e0f08eab7690d2a6ee69";

// получаем пароль от пользователя и кодируем его:
$password = $_POST['password'];
$password = md5($password);

if ($pass === $password) echo "Пароль правильный";
    else echo "Пароль неправильный";
```

- ❑ Функция `md5_file()` возвращает MD5-хеш указанного файла:

```
string md5_file (string $filename [, bool $raw_output = false ])
```

И вам не придется читать файл в строку для вычисления его MD5-хеша.

- ❑ Функции `sha1()` и `sha1_file()` аналогичны функциям `md5()` и `md5_file()`, только вместо алгоритма MD5 используют алгоритм SHA1 (US Secure Hash Algorithm 1).
- ❑ Функция `crc32()` вычисляет контрольную сумму CRC32 для указанной строки `$str`:

```
int crc32 (string $str)
```

- ❑ Функция `crypt()` возвращает хеш указанной строки. Функция более универсальна, чем `md5()`, поскольку позволяет использовать различные алгоритмы шифрования: DES, Blowfish, MD5, SHA1:

```
string crypt (string $str [, string $salt ])
```

Подробное описание функции `crypt()` вы найдете по адресу: <http://ua2.php.net/manual/en/function.crypt.php>.

В PHP 5.5 появился интерфейс (API) хеширования паролей (Password Hashing API). Предполагается, что применение этого интерфейса облегчит и стандартизирует хеширование паролей, и вместо множества разных функций программист сможет использовать теперь один API. На мой взгляд, функциональности у этого API пока маловато — по крайней мере, сейчас. Надеюсь, скоро все изменится к лучшему, а пока рассмотрим то, что есть. Дополнительную информацию об интерфейсе хеширования паролей можно получить по адресу: <http://php.net/password>.

Основная функция API — `password_verify()`. Ей нужно передать пароль и хеш, а она проверит, соответствует ли пароль заданному хешу. Определение функции следующее:

```
boolean password_verify ( string $password, string $hash )
```

Хеш можно создать функцией `password_hash()` из этого API или же функцией `crypt()`. Функция `password_verify()` возвращает `true`, если пароль и хеш соответствуют друг другу, иначе — `false`.

Пример использования функции `password_verify()`:

```
// Хеш пароля.  
$hash = ' $2y$10$hgWxmcltA0SYpCe4zm.H0ew3otg8HCezeTt1CIvuoblVX12JKFcce';  
  
if (password_verify('1234567', $hash)) {  
    echo 'Правильный пароль!';  
} else {  
    echo 'Пароль неправильный!';  
}
```

А теперь рассмотрим функцию `password_hash()`:

```
string password_hash ( string $password, integer $algo [, array $options ] )
```

Первый параметр — это пароль, хеш которого нужно сгенерировать, второй параметр — алгоритм шифрования. Третий параметр необязательный — он задает массив с опциями. В настоящее время поддерживаются две опции:

- ☐ `salt` — «соль», используемая при хешировании пароля;
- ☐ `cost` — алгоритмическая стоимость вычисления пароля.

Пока поддерживается только алгоритм Blowfish, и второй параметр может принимать одно из значений:

- ☐ `PASSWORD_BCRYPT` (integer) = 1
- ☐ `PASSWORD_DEFAULT` (integer) = `PASSWORD_BCRYPT`

Первая константа соответствует алгоритму Blowfish, а вторая задает алгоритм по умолчанию, который пока еще тоже является алгоритмом Blowfish. В следующих версиях PHP эта константа будет переопределена и появятся другие алгоритмы шифрования.

Как видите, возможностей у нового API пока мало, но, полагаю, уже в ближайшее время мы увидим его развитие.

## 13.7. Функции `explode()` и `implode()`: работа с подстроками

При работе с файлами вы часто будете сталкиваться со строками с разделителями. Например, формат CSV для разделения полей строки использует двоеточие:

```
Поле1:Поле2:Поле3
```

- Функция `explode()` читает строку с разделителями, разбивает ее на подстроки и помещает их в массив, каждый элемент которого — подстрока. Для примера рассмотрим следующий сценарий:

```
$str = "user:password:/home/user";
```

```
$info = explode(":", $str);
```

```
print_r($info);
```

Вывод будет следующим:

```
Array
(
    [0] => user
    [1] => password
    [2] => /home/user
)
```

У функции `explode()` есть еще и третий параметр, ограничивающий количество полей в просматриваемой строке. В этом случае функция прочитает только указанное нами количество полей.

- Функция `implode()` служит для объединения элементов массивов в строку:

```
string implode (string $glue , array $pieces)
```

Первый параметр — это разделитель, которым будут разделены элементы массива при их помещении в строку. В строку помещаются только значения элементов массива, но не ключи. Функцию `implode()` удобно использовать для чтения всего файла в одну строку:

```
$txt = implode('', file('article.txt'));
```

- Функция `join()` является псевдонимом для функции `implode()`.

## 13.8. Статистические функции

- Функция `count_chars()` возвращает количество символов в строке:

```
mixed count_chars (string $string [, int $mode = 0 ])
```

Возвращаемое значение зависит от второго параметра (`$mode`):

- 0 — массив элементов, ключами которого являются коды символов, значением каждого элемента является количество данного символа в строке;
- 1 — то же, что и 0, но в массив включаются символы с частотой больше 0;
- 2 — то же, что и 0, но в массиве представлены символы с частотой 1;
- 3 — строка, содержащая все уникальные символы;
- 4 — строка, содержащая все неиспользуемые символы.

Рассмотрим небольшой пример:

```
$str= "Hello!";

foreach (count_chars($str, 1) as $i => $val) {
    echo "Символ " . chr($i) . " встречается в строке $val раз(a) <br>";
}
```

Вывод сценария будет следующим:

```
Символ ! встречается в строке 1 раз(a)
Символ H встречается в строке 1 раз(a)
Символ e встречается в строке 1 раз(a)
Символ l встречается в строке 2 раз(a)
Символ o встречается в строке 1 раз(a)
```

□ Функция `str_word_count()` возвращает количество слов в строке:

```
$str = "Hello, world!";
echo str_word_count($str);
```

Функция `substr_count()` подсчитывает, сколько раз подстрока `$needle` встречается в подстроке `$haystack`:

```
int substr_count (string $haystack , string $needle [, int $offset = 0
[, int $length ]])
```

## 13.9. Функции вывода текста

Для вывода текста мы привыкли применять конструкцию `echo`. Если вам не нравится `echo` в том виде, как используем его мы, вы можете вызывать `echo` как функцию — со скобками, внутри которых будут параметры, которые и отобразит функция `echo()`:

```
void echo (string $arg1 [, string $... ])
```

Программистам, знакомым с C, понравится функция `print()`, которая хоть как-то напоминает их любимый язык программирования:

```
int print (string $arg)
```

По большому счету `print` тоже не является функцией в прямом смысле слова — это конструкция языка программирования (оператор), как и `echo`.

Рассмотрим несколько примеров вывода строки `Hello`:

```
echo "Hello";
echo("Hello");
print "Hello";
print("Hello");
```

Все эти варианты равноправны и выведут нашу строку без каких-либо нюансов.

Совсем другое дело — функция `sprintf()`, которая возвращает аргументы согласно переданной ей строке формата. Другими словами, `sprintf()` возвращает отформатированную строку. Эта функция похожа на аналогичную функцию в языке C:

```
string sprintf (string $format [, mixed $args [, mixed $...]])
```

#### ПРИМЕЧАНИЕ

Обратите внимание: функция `sprintf()` именно возвращает строку, а не выводит ее в браузер. Вы должны получить отформатированную строку (результат работы функции `sprintf()`) и вывести ее в браузер самостоятельно функцией `echo()` или `print()`.

Строка формата (первый аргумент, `$format`) задает формат для каждого аргумента, переданного после аргумента `$format`. Строка формата состоит из 0 или более модификаторов формата. Каждый модификатор формата предваряется символом `%`. В табл. 13.1 приводятся модификаторы формата для функции `sprintf()`.

**Таблица 13.1.** Модификаторы формата для `sprintf()`

Модификатор	Описание
<code>%</code>	Используется для вывода символа <code>'%'</code>
<code>b</code>	Аргумент считается целым числом, выводится в двоичном виде
<code>c</code>	Аргумент считается целым числом, выводится в виде символа с соответствующим ASCII-кодом
<code>d</code>	Аргумент считается числом, выводится в виде десятичного числа со знаком
<code>e</code>	Аргумент считается числом с плавающей запятой (float) и выводится в научной нотации (например, <code>1.3e+3</code> )
<code>u</code>	Целое беззнаковое число в десятичном виде
<code>f</code>	Аргумент считается числом с плавающей запятой (float) в десятичном виде с плавающей запятой
<code>o</code>	Целое число, выводится в восьмеричной системе
<code>x</code>	Число в шестнадцатеричном виде, все символы строчные
<code>X</code>	Число в шестнадцатеричном виде, все символы прописные

Пример:

```
$n = 100;
// выводим число 100 в десятичном, шестнадцатеричном и двоичном виде
echo sprintf("Dec: %d Hex: %X Bin: %b", $n, $n, $n);
```

Функцию `sprintf()` удобно использовать, если требуется присвоить результат ее работы переменной для последующей обработки. Если же нужно просто вывести отформатированную строку, следует воспользоваться функцией `printf()`:

```
int printf (string $format [, mixed $args [, mixed $...]])
```

Параметры у функции `printf()` такие же, как у `sprintf()`, разница заключается в том, что `printf()` сразу выводит строку в браузер. Дополнительную информацию о модификаторах форматов функций `printf()` и `sprintf()` можно получить по адресу: <http://ua2.php.net/manual/en/function.sprintf.php>.

Функция `fprintf()` выводит отформатированную строку в поток (stream), заданный дескриптором `$handle`:

```
int fprintf (resource $handle, string $format [, mixed $args [, mixed $...]])
```

Пример:

```
if (!$f = fopen('numbers.txt', 'w'))  
die();  
$n = 555;  
fprintf($f, "%X %b", $n, $n);
```

Кроме функции `printf()`, в РНР есть подобная ей функция — `vprintf()`, которая действует аналогично функции `printf()`, но вместо списка аргументов принимает массив аргументов:

```
int vprintf (string $format, array $args)
```

Массиво-ориентированный аналог есть и для функции `sprintf()` — такая функция называется `vsprintf()`:

```
string vsprintf (string $format, array $args)
```

Функция `vfprintf()` выводит отформатированную строку в поток (stream), заданный дескриптором `$handle`:

```
int vfprintf (resource $handle, string $format, array $args)
```

## 13.10. Установка локали

Многие строковые функции для своей правильной работы требуют точной установки *локали*. Для установки локали служит функция `setlocale()`:

```
string setlocale (int $category, string $locale [, string $...])
```

Первый параметр — это категория объектов, к которым должны быть применены настройки локали:

- ☐ `LC_ALL` — для всего указанного далее;
- ☐ `LC_COLLATE` — для сравнения строк;
- ☐ `LC_CTYPE` — для классификации и преобразования строк — например, для функций вроде `strtoupper()`;
- ☐ `LC_MONETARY` — для `localeconv()`;
- ☐ `LC_NUMERIC` — для десятичного разделителя;
- ☐ `LC_TIME` — для форматирования даты и времени функцией `strftime()`;

- `LC_MESSAGES` — для ответов системы (доступно, если PHP откомпилирован с опцией `libintl`).

Второй параметр — это сама локаль. Для русского языка применяется локаль `ru_RU`.

В наши дни кодировка UTF-8 практически стала стандартом, поэтому желательно настроить локаль на использование этой кодировки. Делается это так:

```
setlocale(LC_ALL, "ru_RU.UTF-8");
```

К сожалению, хотя этот вызов будет прекрасно работать на UNIX (чего в большинстве случаев вполне достаточно — ведь все серьезные проекты помещаются на UNIX-хостинг), в Windows такой трюк не работает.

## 13.11. Форматирование чисел и денежных величин

- Форматирование денежных величин обеспечивает функция `money_format()`. Для ее правильной работы необходима точная установка локали.

Получить подробную информацию об этой функции можно по адресу: <http://ua2.php.net/manual/en/function.money-format.php>.

- Функция `number_format()` служит для форматирования чисел:

```
string number_format (float $number [, int $decimals = 0 ])
```

Первый параметр — это число, второй — количество знаков после запятой для вещественных чисел:

```
$n = 1000000;  
echo number_format($n, 0);           // выведет 1,000,000
```

Полный прототип функции выглядит так:

```
string number_format (float $number, int $decimals = 0,  
string $dec_point = '.', string $thousands_sep = ',')
```

Третий параметр — это символ, разделяющий целую и дробную части числа (по умолчанию — точка), четвертый параметр — это разделитель тысяч (по умолчанию — запятая).

## 13.12. Преобразование систем счисления

Функция `bin2hex()` выполняет преобразование из двоичной системы в шестнадцатеричную, а функция `hex2bin()` — обратное преобразование из шестнадцатеричной системы в двоичную.

Прототипы функций следующие:

```
string bin2hex (string $str)  
string hex2bin (string $data)
```

## 13.13. Строки в PHP 7/8

В PHP 7/8 добавлена поддержка строк длиной более 231 байта, но только в 64-битных сборках PHP. Другими словами, не только ваша система должна быть 64-битной, но и сама сборка PHP.

Также был добавлен синтаксис `\u{xxxxxx}` для строк, позволяющий указывать внутри PHP-строк произвольные символы Unicode. Например:

```
"\u{1F49A}"
```

Дополнительную информацию об этом нововведении можно найти здесь:

❑ [https://wiki.php.net/rfc/remove\\_hex\\_support\\_in\\_numeric\\_strings](https://wiki.php.net/rfc/remove_hex_support_in_numeric_strings);

❑ [https://wiki.php.net/rfc/unicode\\_escape](https://wiki.php.net/rfc/unicode_escape).

Строки, которые содержат шестнадцатеричные цифры, теперь не рассматриваются как числа. Пример нового поведения:

```
var_dump("0x123" == "291"); // bool(false) (ранее true)
var_dump(is_numeric("0x123")); // bool(false) (ранее true)
var_dump("0xe" + "0x1"); // int(0) (ранее 16)
var_dump(substr("foo", "0x1")); // string(3) "foo" (ранее "oo")
```

Для проверки, содержит ли строка шестнадцатеричное число, и для ее конвертирования в целое число можно использовать функцию `filter_var()`:

```
$str = "0xffff";
$num = filter_var($str, FILTER_VALIDATE_INT, FILTER_FLAG_ALLOW_HEX);
if (false === $num) {
    throw new Exception("Invalid integer!");
}
var_dump($num); // int(65535)
```



7

## ГЛАВА 14

# Работаем с файлами и каталогами

### 14.1. Права доступа в UNIX

Прежде чем приступать к изучению функций для работы с файлами и каталогами, мы должны рассмотреть вопрос назначения прав доступа к файлам и каталогам. Если вы сейчас работаете в Windows, то можете пропустить этот раздел, но вы обязательно вернетесь к нему, когда станете переносить свои сценарии на сервер хостинг-провайдера (который обычно работает под управлением одного из вариантов UNIX) и столкнетесь с проблемой доступа к файлам.

В UNIX у каждого файла (или каталога, который по сути тоже является файлом) есть свой *владелец* — пользователь, создавший файл. И для каждого каталога и файла вы можете задать права доступа. Точнее, права доступа автоматически задаются при создании каталога (файла), а вы при необходимости можете их изменить. Чуть позже мы поговорим об этой необходимости с точки зрения РНР.

Существуют три права доступа: чтение (r), запись (w), выполнение (x). Для каталога право на выполнение означает право на просмотр содержимого каталога.

Вы можете установить разные права доступа для владельца (т. е. для себя), для группы владельца (т. е. для всех пользователей, входящих в одну с владельцем группу) и для прочих пользователей. Пользователь root может получить доступ к любому файлу (каталогу) вне зависимости от прав, которые вы установили.

При выводе содержимого каталога на UNIX-сервере права доступа к файлу выводятся примерно так:

```
-r--r----- 1 user group 300 Apr 11 11:11 video.txt
```

В этой записи последовательность символов `-r--r-----` обозначает права доступа. Первый дефис означает, что перед нами обычный файл. В случае каталога на его месте стояла бы буква `d`. Следующие три символа (`r--`) определяют права доступа владельца. Первый символ — это чтение, второй — запись, третий — выполнение. Как можно видеть, в нашем случае владельцу разрешено только чтение этого файла, запись и выполнение запрещены, поскольку в правах доступа режимы `w` и `x` не определены.

Следующие три символа (r--) задают права доступа для членов группы владельца. Права здесь такие же, как и у владельца, — можно читать файл, но его нельзя изменять или запускать.

Последние три символа (---) задают права доступа для прочих пользователей. Прочие пользователи не имеют права ни читать, ни изменять, ни выполнять этот файл. При попытке получить доступ к файлу они увидят сообщение: **Access denied**.

Права доступа задаются командой `chmod`. Существуют два способа указания прав доступа: символьный (когда указываются символы, задающие право доступа: r, w, x) и абсолютный. Так уж заведено, что в мире UNIX чаще пользуются абсолютным методом. Разберемся, в чем он заключается. Для этого рассмотрим следующий набор прав доступа:

`rw-r-----`

Он предоставляет владельцу право чтения и модификации файла (rw-), запускать файл владелец не может. Члены группы владельца могут только просматривать файл (r--), а все остальные пользователи не имеют вообще никакого доступа к файлу.

Возьмем отдельный набор прав — например, для владельца:

`rw-`

Чтение разрешено, значит, мысленно записываем 1. Запись разрешена, значит, запоминаем еще 1, а вот выполнение запрещено, поэтому запоминаем 0. Получается число 110. Если из двоичной системы число 110 перевести в восьмеричную, получится число 6. Для перевода можно воспользоваться табл. 14.1.

Таблица 14.1. Преобразование чисел из двоичной системы в восьмеричную

Двоичная система	Восьмеричная система
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Аналогично произведем разбор прав для членов группы владельца. Получится 100, т. е. 4 в восьмеричной системе. С третьим набором (---) все вообще просто: это 000, т. е. восьмеричный 0.

Записываем полученные числа в восьмеричной системе в порядке владелец — группа — остальные. Получится число 640 — это и есть права доступа в абсолютной записи. Для того чтобы установить эти права для файла, выполните команду:

`chmod 640 <имя_файла>`

Наиболее популярные права доступа:

- ❑ 644 — владельцу можно читать и изменять файл, остальным пользователям — только читать;
- ❑ 666 — читать и изменять файл можно всем пользователям;
- ❑ 777 — всем можно читать, изменять и выполнять файл. Напомню, что для каталога право выполнения — это право просмотра его оглавления.

Если вы хотите кому-нибудь файл «подарить», т. е. сделать какого-либо пользователя владельцем файла, то вам нужно выполнить команду `chown`:

```
chown пользователь файл
```

Учтите, что, возможно, после изменения владельца файла вы сами не сможете получить к нему доступ, ведь его владельцем будете уже не вы.

А теперь поговорим о нюансах доступа к файлам из RНР-сценариев. Предположим, что у вас есть сценарий гостевой книги `gb.php` и файл `gb.txt`, в котором хранятся ее записи. Вы загрузили оба файла на сервер по протоколу FTP. Следовательно, вы — владелец этих двух файлов. По умолчанию (в большинстве случаев) им будут назначены права доступа 644.

Теперь пользователь заходит на страницу гостевой книги и пытается добавить новую запись. RНР пытается открыть файл `gb.txt` в режиме записи, но не может этого сделать. Почему? Потому что интерпретатор RНР выполняется от имени пользователя `www-data` (от имени этого же пользователя работает и сам веб-сервер), а для всех остальных пользователей право доступа равно 4 (только чтение).

Что делать в этом случае? Вы можете присвоить файлу `gb.txt` (не `gb.php`!) право доступа 666, но это позволит получить к нему полный доступ не только пользователю `www-data`, но и любому другому пользователю. Или же можно этот файл пользователю `www-data` «подарить» (владелец задается в формате пользователь:группа):

```
chown www-data:www-data gb.txt
```

Перед таким «подарком» уточните у администратора сервера, от имени какого пользователя выполняется RНР, чтобы случайно не подарить файл другому пользователю. Недостатком этого метода является то, что после этого вы ничего не сможете сделать с файлом `gb.txt` (ни отредактировать, ни удалить), поскольку он будет теперь принадлежать другому пользователю.

Чтобы не было проблем с доступом к хранилищу данных, рекомендуется применять для хранения данных таблицы MySQL, но иногда (например, когда MySQL недоступен) приходится использовать и файлы.

## 14.2. Чтение файла

Существуют три способа чтения файла:

- ❑ открываем файл функцией `fopen()` и читаем его функцией `fread`;
- ❑ построчное чтение файла с помощью функции `file()`;
- ❑ чтение всего файла сразу функцией `file_get_contents()`.

### 14.2.1. Функции `fopen()` и `fread()`

Функции открытия файла `fopen()` нужно передать всего два параметра: имя файла и режим доступа к файлу:

```
fopen (string filename, string mode [, bool use_include_path  
[, resource zcontext]])
```

Остальные два параметра настолько редко используются, что мы даже не будем их рассматривать.

Имя файла можно указать разными способами (предположим, что мы хотим получить доступ к файлу `gb.txt`):

- ❑ `/home/user/htdocs/gb.txt` — полный путь к файлу (мы считаем, что файл находится в каталоге `htdocs` домашнего каталога пользователя `user`);
- ❑ `gb.txt` — относительный путь к файлу, файл находится в одном каталоге с PHP-сценарием;
- ❑ `../gb.txt` — файл `gb.txt` находится в родительском каталоге относительно каталога, в котором находится PHP-сценарий;
- ❑ `data/gb.txt` — файл находится в подкаталоге `data` каталога, в котором находится PHP-сценарий.

Второй параметр `fopen()` — это не права доступа, как вы могли бы подумать, а символы, задающие режим доступа к файлу (табл. 14.2).

Таблица 14.2. Режимы доступа к файлу

Режим	Описание
<code>r</code>	Только чтение. Если файл не существует, функция вернет <code>false</code>
<code>r+</code>	Режим чтения/записи. Указатель файла будет помещен на его начало, т. е. при записи в файл новые данные будут записаны поверх уже имеющихся. Например, если в файле была строка <code>Мама мыла раму</code> , то после записи в файл строки <code>Папа</code> получится строка <code>Папа мыла раму</code> . Чтобы не создавать непонятных ситуаций (что "папа" "мыла"?), помните об особенности этого режима — новые данные не вставляются в файл, а записываются поверх уже имеющихся (как в режиме вставки на клавиатуре). Если файл не существует, функция вернет <code>false</code>
<code>w</code>	Функция создаст новый пустой файл. Если файл с таким именем существует, он будет перезаписан. Файл открывается в режиме записи
<code>w+</code>	То же самое, что и <code>r+</code> , но перезаписывает файл, если он существует, или создает файл, если такого файла не существует. Файл открывается в режиме чтения/записи
<code>a</code>	Открывает файл в режиме записи. Запись производится в конец файла. Если файл не существует, функция вернет <code>false</code>
<code>a+</code>	Открывает файл в режиме чтения/записи. Указатель записи файла устанавливается на конец файла. Это наиболее универсальный режим: если файл существует, его содержимое не удаляется, если файл не существует, создается пустой файл
<code>b</code>	Открывает файл в двоичном (бинарном) режиме. Полезен для чтения нетекстовых файлов (когда вы знаете, что делаете)

Большинство ошибок при открытии файлов связаны с нарушением прав доступа. Предположим, что вы пытаетесь создать в своем домашнем каталоге `/home/den` файл `gb.txt` (открыть в режиме `w` или `a+`). Но интерпретатор PHP, выполняющийся от имени пользователя `apache`, не имеет доступа к вашему домашнему каталогу в режиме записи, поэтому вы не сможете создать файл. Вот пути решения этой проблемы:

- ❑ назначить домашнему каталогу права доступа `777` — тогда вы сможете без ограничений создавать/модифицировать/удалять файл, но это же смогут делать любые другие пользователи системы. Потом не удивляйтесь, что ваш сайт взломали. Одним словом, этот способ отпадает сразу;
- ❑ создать каталог `data` (в домашнем каталоге) и назначить ему права `777` — в этом каталоге вы сможете создавать и удалять файлы. Если ваши сценарии используют много файлов для хранения данных, это неплохой вариант;
- ❑ создать файл `gb.txt` вручную и присвоить ему права `666` — неплохой вариант, если файл всего один. Если файлов 100, лучше использовать предыдущий вариант.

Для открытия файла следует не просто вызвать функцию `fopen()`, а получить дескриптор файла `$handle`, который потом будет использован в других функциях (чтобы та же функция `fread()` знала, из какого файла нужно произвести чтение, если вы открыли несколько файлов). Вот примеры открытия файлов функцией `fopen()`:

```
$handle = fopen("/home/user/file.txt", "r");
$handle = fopen("/home/user/file.gif", "wb");
$handle = fopen("http://www.example.com/", "r");
$handle = fopen("ftp://user:password@example.com/somefile.txt", "w");
```

После открытия файла нужно проанализировать, открыл ли PHP файл. Если `$handle` установлен в `false`, то файл не открыт. Намного проще это сделать с помощью конструкции `or die`:

```
$handle = fopen("/home/den/file.txt", "w") or die("Не могу создать файл");
```

Как работает эта конструкция, думаю, понятно — если функция возвращает `false`, выводится сообщение об ошибке, и работа сценария завершается.

Итак, файл мы открыли. Теперь нужно что-то из него прочитать. Для этого служит функция `fread()`, которой надо передать два параметра: дескриптор файла и количество байтов, которое должно быть из файла прочитано:

```
// читаем из файла 100 байтов и помещаем в строку $f
$s = fread($handle, 100);
```

Если в файле будет всего 50 байтов, то функция `fread()` прочитает только эти 50 байтов. При чтении файла нужно контролировать, достигнут ли конец файла. Функция `feof()` возвращает `true`, если достигнут конец файла.

Рассмотрим пример посимвольного чтения файла в строку `$s`:

```
$handle = fopen("file.txt", "r");

$s = "";
```

```
while (!feof($handle))
{
    $s = $s . fread($handle, 1);
}

echo $s;
```

### ПРИМЕЧАНИЕ

Вместо `fread($handle, 1)` для чтения одного символа можно использовать функцию `fgets($handle)`.

По окончании обработки файл нужно закрыть с помощью функции `fclose()`:

```
fclose($handle);
```

Вообще-то, PHP при завершении сценария автоматически закрывает все открытые файлы, но закрытие файла программистом считается хорошим тоном.

## 14.2.2. Функция `file()`: построчное чтение файла

Функция `fread()` возвращает указанное пользователем количество байтов. А что делать, если нам нужно получить строку файла? Ведь мы не знаем, сколько символов в строке. В одной строке может быть один символ, а в другой — 50.

Для чтения строки из файла служат следующие функции:

- ❑ `fgets()` — читает строку из файла;
- ❑ `fgetss()` — читает строку из файла и удаляет HTML-теги.

Вот пример построчного чтения и вывода файла:

```
$handle = fopen("file.txt", "r");

while (!feof($handle))
{
    echo fgets($handle);
}

fclose($handle);
```

Обратите внимание, что для организации построчного чтения нам потребовались четыре функции. Намного проще использовать функцию `file()`, которая читает файл в массив. Каждый элемент массива — это прочитанная строка из файла. Вот как можно организовать построчное чтение файла:

```
$f = file('file.txt');
```

Все, что осталось, — это вывести файл. Вот пример построчного чтения и вывода HTML-страницы:

```
$lines = file('https://www.example.com/');

foreach ($lines as $line_num => $line) {
    echo "Строка #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br />\n";
}
```

Если вы будете использовать функции `fgets()` и `fgetss()`, вам нужно знать их прототипы:

```
string fgets (resource handle [, int length])
string fgetss (resource handle [, int length [, string allowable_tags]])
```

Первый параметр — это дескриптор файла, второй — максимальная длина строки в байтах, третий (для функции `fgetss()`) — разрешенные HTML-теги.

### 14.2.3. Чтение всего файла: функция `file_get_contents()`

Предположим, что мы прочитали файл функцией `file()`. Затем мы его можем обработать построчно — например, вывести. Но использовать цикл для вывода строки не совсем удобно. Несколько удобнее объединить массив в строку и вывести уже ее, например:

```
$s = join('', file('file.txt'));
echo $s;
```

Но и эта конструкция слишком громоздкая. Намного проще использовать функцию `file_get_contents()`, которая возвращает сразу все содержимое файла:

```
echo file_get_contents('file.txt');
```

## 14.3. Запись файла

Запись информации в файл можно осуществить с помощью следующих функций:

- ☐ `file_put_contents()` — записывает строку в файл;
- ☐ `fwrite()` (или `fputs()`) — служит как для записи строки, так и для записи бинарных данных.

Вот прототипы этих функций:

```
int file_put_contents (string filename, mixed data [, int flags [, resource
                                                                context]])
int fwrite (resource handle, string string [, int length])
```

Функцию `file_put_contents()` использовать весьма просто — вам не потребуется получать дескриптор файла, проверять этот дескриптор, вызывать функцию `fwrite()`. Достаточно только указать имя файла, данные и флаги. Самый полезный флаг `FILE_APPEND` — его наличие обеспечивает запись данных в конец файла.

Функции `fwrite()` нужно передать дескриптор файла, данные и необязательный параметр `length` — максимальную длину записываемых данных (если строка `string` длиннее, чем `length`, то будет записано `length` байтов).

Если вы хотите убедиться, что функцией `fwrite()` данные записаны в файл, а не остались где-то в буфере записи, используйте функцию `fflush()`:

```
fflush($handle);
```

## 14.4. Создание временных файлов

Иногда для хранения промежуточных данных нам нужен временный файл. Создавать его с помощью `fopen()` не слишком удобно. Во-первых, нам тогда придется контролировать имя файла — чтобы ненароком не перезаписать существующий файл. Во-вторых, временный файл после обработки нужно будет удалить. Намного проще использовать функцию `tmpfile()`, создающую временный файл в каталоге `/tmp`.

Вот пример работы функции `tmpfile()`:

```
$temp = tmpfile();  
fwrite($temp, "данные");  
fseek($temp, 0);           // переходим на начало файла  
echo fread($temp, 1024);  
fclose($temp);             // при закрытии временный файл удаляется
```

Заметьте, что временный файл открывается в режиме чтения/записи.

## 14.5. Работа с CSV-файлами

Предположим, что у нас есть небольшая электронная таблица, содержащая данные о продажах за год. Сохранив эту таблицу в формате CSV, мы получим файл следующего содержания:

```
Январь;19000  
Февраль;25000  
Март;64400  
Апрель;56000  
Май;29000  
Июнь;35006  
Июль;19800  
Август;17000  
Сентябрь;45000  
Октябрь;59800  
Ноябрь;48000  
Декабрь;75000
```

Несмотря на то что формат называется CSV (Comma-Separated Values, разделенные запятой значения), Excel почему-то разделяет поля точкой с запятой, а не просто запятой. PHP-функция `fgetcsv()` по умолчанию использует в качестве разделителя

запятую, поэтому, если мы работаем с CSV-файлами, полученными из Excel, нужно задать разделитель:

```
fgetcsv($handle, 1000, ";")
```

Первый параметр — это дескриптор файла, полученный с помощью `fopen()`, второй — максимальная длина строки в байтах, третий — разделитель. Если разделитель не указан, то в качестве разделителя используется запятая. Если не указана длина (или длина равна 0), то функция будет читать данные до конца строки.

Рассмотрим пример обработки CSV-файла:

```
$row = 1; // номер строки
$handle = fopen("book.csv", "r");

while (($data = fgetcsv($handle, 1000, ";")) !== FALSE) {
    // количество полей в строке
    $num = count($data);

    echo "<p> $num полей в строке $row: <br /></p>\n";

    $row++;

    // выводим поля
    for ($c=0; $c < $num; $c++) {
        echo $data[$c] . "<br />\n";
    }
}
fclose($handle);
```

Для записи строки в CSV-файл служит функция `fputcsv()`:

```
int fputcsv (resource handle [, array fields [, string delimiter [, string
enclosure]])
```

Первый параметр — дескриптор файла, второй — массив полей, которые нужно поместить в файл, третий параметр — разделитель. Последний параметр задает ограничитель полей — по умолчанию это двойная кавычка (").

Каждый элемент массива, передаваемого функции, — это строка, содержащая поля, разделенные разделителем, например:

```
$list = array (
    'январь;10700',
    'февраль;1789',
    'март; 10012'
);
```

Вот пример записи массива `$list` в файл `file.csv`:

```
$fp = fopen('file.csv', 'w');

foreach ($list as $line) {
    fputcsv($fp, split(';', $line));
}

fclose($fp);
```

Функция `str_getcsv()` выполняет разбор CSV-строки в массив:

```
array str_getcsv ( string $input [, string $delimiter = ",",  
[, string $enclosure = '"' [, string $escape = "\\\" ]]] )
```

Рассмотрим ее параметры:

- ☐ `input` — обрабатываемая строка;
- ☐ `delimiter` — разделитель поля (по умолчанию: `,`);
- ☐ `enclosure` — устанавливает символ ограничителя поля. Обычно значение поля записывается в кавычки, это же значение используется по умолчанию;
- ☐ `escape` — задает экранирующий символ. По умолчанию — это символ обратного слеша.

Функция возвращает индексированный массив, содержащий полученные из строки поля. Рассмотрим небольшой пример из реального проекта:

```
$dsa = str_getcsv($v, ";");           // data string array  
$id = $dsa[0];                       // ID  
$dt = $dsa[1];                       // дата  
$st = str_replace('T', ' ', $dsa[2]); // start-time  
$tt = str_replace('T', ' ', $dsa[3]); // stop time  
$dist = $dsa[4];                     // расстояние в метрах
```

Функция здесь возвращает массив, который мы помещаем в переменную `$dsa`. Далее мы обращаемся к полям CSV-файла в порядке их следования (нумерация начинается с 0). Первое поле имеет имя `$dsa[0]`, второе — `$dsa[1]` и т. д.

## 14.6. Специальные функции для работы с файлами

PHP умеет не только читать и записывать файлы, но и производить операции с файлами на уровне файловой системы: вы можете копировать, переименовывать и удалять файлы. Также в составе PHP есть функции, позволяющие изменять права доступа к файлам и каталогам. Далее мы рассмотрим PHP-функции для работы с файлами, разбитые на группы.

### 14.6.1. Функции для работы с именами файлов

К этой группе относятся следующие три функции:

- ☐ `basename()` — возвращает имя файла из указанного пути;
- ☐ `realpath()` — возвращает абсолютный путь к файлу;
- ☐ `file_exists()` — позволяет проверить, существует ли файл.

Функции в работе довольно простые, поэтому мы ограничимся примерами их использования:

```
$path = "/home/user/htdocs/index.php";
$file = basename($path); // $file содержит "index.php"
$file = basename($path, ".php"); // $file содержит "index"

// выведет /home/user/htdocs/index.php
echo realpath("index.php");

if (file_exists($path)) {
    echo "Файл существует";
} else {
    echo "Файл не существует";
}
```

## 14.6.2. Работа с правами доступа

Функции работы с правами доступа позволяют установить права доступа, определить права доступа, определить, можно ли выполнять определенные операции с файлом (например, запись или выполнение), и другие операции, связанные с доступом к файлу:

- ☐ `chgrp()` — изменяет группу владельцев файла;
- ☐ `chmod()` — изменяет права доступа к файлу или каталогу;
- ☐ `chown()` — изменяет владельца файла;
- ☐ `filegroup()` — возвращает идентификатор группы файла;
- ☐ `fileowner()` — возвращает идентификатор владельца файла;
- ☐ `fileperms()` — предоставляет информацию о правах на файл или каталог;
- ☐ `is_dir()` — определяет, является ли файл каталогом;
- ☐ `is_executable()` — определяет, является ли файл исполняемым;
- ☐ `is_file()` — определяет, является ли файл обычным файлом;
- ☐ `is_link()` — определяет, является ли файл символической ссылкой;
- ☐ `is_readable()` — определяет, доступен ли файл для чтения;
- ☐ `is_uploaded_file()` — определяет, был ли файл загружен с помощью HTTP POST;
- ☐ `is_writable()` — определяет, доступен ли файл для записи.

Рассмотрим некоторые из этих функций чуть подробнее.

- ☐ Функциям `chgrp()`, `chmod()` и `chown()` нужно передать имя файла и соответственно идентификатор группы, значение прав доступа, идентификатор пользователя (нового владельца файла), например:

```
chgrp("file.txt", "user");
chmod("file.txt", 500);
chown("file.txt", "user");
chmod("file.txt", 666);
```

**ПРИМЕЧАНИЕ**

Идентификаторы пользователя и группы можно задавать как строкой, так и числом. Права доступа для `chmod()` нужно задавать только числом.

- ❑ Функциям `filegroup()`, `fileowner()` и `fileperms()` нужно передать всего один параметр — имя файла. Функции возвращают число: идентификатор группы, владельца, значение прав доступа соответственно. Пример:

```
echo substr(sprintf('%o', fileperms('/etc/passwd')), -4);
```

- ❑ Функциям `is_*`() нужно тоже передать имя файла/каталога, а в результате получить `true` или `false` — в зависимости от того, истинно ли проверяемое функциями условие или нет:

```
if (is_writable('file.txt'))
echo "Файл доступен для записи";
else
echo "Файл не доступен для записи";
```

### 14.6.3. Копирование, переименование и удаление файлов

- ❑ Для копирования файла служит функция `copy()`:

```
bool copy (string source, string dest)
```

Первый параметр — это имя исходного файла, второй параметр — имя файла назначения (файл будет перезаписан в случае существования). Если копирование прошло удачно, функция возвращает `true`, в противном случае — `false`.

Небольшой пример копирования файла:

```
$file = 'file.txt';
$newfile = 'file.txt.bak';

if (!copy($file, $newfile)) {
echo "Не удалось скопировать $file...\n";
}
```

- ❑ Для переименования или перемещения файла служит функция `rename()`:

```
bool rename (string oldname, string newname)
```

Как и функция `copy()`, эта функция возвращает `true` в случае успешного завершения операции. Пример переименования/перемещения файла:

```
// переименовываем /tmp/tmp.txt в /home/den/my_file.txt
rename("/tmp/tmp.txt", "/home/den/my_file.txt");
```

- ❑ Для перемещения загруженного файла в другой каталог служит функция `move_uploaded_file()`:

```
bool move_uploaded_file (string filename, string destination)
```

Эта функция похожа на `rename()`, но перед перемещением проверяет, был ли файл загружен по HTTP.

- ❑ Функция `unlink()` удаляет файл, которому нужно передать имя удаляемого файла:  
`unlink('file.txt');`

При выполнении операций с файлами помните о правах доступа:

- ❑ чтобы PHP мог скопировать файл в указанный вами каталог, пользователь, от имени которого запущен PHP, должен иметь право записи в этот каталог;
- ❑ чтобы прочитать файл (например, при копировании), у PHP должно быть право на чтение файла.

### 14.6.4. Время доступа к файлу

Для получения и установки времени последнего доступа к файлу используются следующие функции:

- ❑ `filetime()` — возвращает время последнего доступа к файлу;
- ❑ `filemtime()` — возвращает время последнего изменения файла;
- ❑ `touch()` — устанавливает время доступа и модификации файла.

Первым двум функциям нужно передать всего один параметр — имя файла. Функции возвращают время в формате UNIX timestamp (число секунд, прошедшее с полуночи 1 января 1970 г.).

Функции `touch()` можно передать три параметра:

```
bool touch (string filename [, int time [, int atime]])
```

Первый параметр — имя файла, второй — время доступа и модификации файла. Если вам нужно задать разное время для операций доступа (открытия) и модификации (изменения), то укажите третий параметр — время доступа. Тогда второй параметр будет использоваться для установки только времени модификации файла. Время нужно задавать в формате timestamp (см. главу 12).

Если второй (понятно, и третий) параметр не задан, то будет использовано текущее время.

### 14.6.5. Другие полезные функции

Функция `filesize()` позволяет получить размер файла в байтах. Ей нужно передать только имя файла. А с помощью функции `filetype()` вы сможете определить тип файла. Функции также нужно передать имя файла.

Вот возможные значения результата:

- ❑ `fifo` — очередь, First In First Out;
- ❑ `char` — символьное устройство UNIX (например, консоль или клавиатура);

- ❑ `dir` — каталог;
- ❑ `block` — блочное устройство UNIX (например, жесткий диск);
- ❑ `link` — ссылка;
- ❑ `file` — обычный файл;
- ❑ `unknown` — файл неизвестного типа.

Например:

```
echo filetype('/etc/passwd');      // выведет file
echo filetype('/etc/');            // выведет dir
```

Нужно заметить, что в Windows будут доступны лишь два типа: `file` и `dir`. Остальные типы доступны только в UNIX.

## 14.7. Совместный доступ к файлу

Предположим, что мы разрабатываем сценарий гостевой книги, который хранит записи в файле `gb.txt`. Вроде бы все просто: сначала сценарий открывает файл только для чтения и выводит записи гостевой книги, затем, если пользователь пожелает оставить сообщение, файл открывается для записи, и новое сообщение записывается в файл.

Но что делать, если два пользователя одновременно пытаются добавить сообщение в гостевую книгу, или один сценарий открыл файл для записи, а второй пытается открыть для чтения?

Результат в обоих случаях непредсказуем. Поэтому, чтобы быть уверенным, что одновременно отправленные данные записаны в нужном порядке (а не вперемешку) и что сценарий точно сможет прочитать файл, нужно использовать *механизм блокировок*. Этот механизм доступен только в UNIX, т. е. он будет работать на сервере хостинг-провайдера, но не на вашем домашнем компьютере под управлением Windows.

Для установки блокировки служит функция `flock()`:

```
bool flock (resource handle, int operation [, int &wouldblock])
```

Первый параметр — это дескриптор файла, второй — тип блокировки. Если третий параметр установлен в `true`, то нужно не только запереть файл, но и блокировать выполнение сценария, пока файл не станет доступным.

Рассмотрим значения второго параметра:

- ❑ `LOCK_EX` — исключительная блокировка, ее нужно применять при записи в файл, при этом все остальные сценарии не смогут получить доступ к файлу, пока вы не снимете блокировку (или пока не завершится ваш сценарий);
- ❑ `LOCK_SH` — разделяемая блокировка, ее можно использовать при чтении данных из файла;
- ❑ `LOCK_NB` — если другой процесс заблокировал файл, то вы можете или ждать, пока файл разблокируется, или же получить сообщение об ошибке и завершить

выполнение сценария. Значение `LOCK_NB`, добавленное, например, к исключительной блокировке, запрещает ожидание;

- ❑ `LOCK_UN` — разблокирование файла:

```
$handle = fopen("gb.txt", "a");

// исключительная блокировка
if (flock($handle, LOCK_EX+LOCK_NB)) {
    fwrite($handle, "Данные");
    flock($handle, LOCK_UN);           // отпираем файл
} else {
    echo "Не могу выполнить блокировку файла!";
}

fclose($fp);                         // закрываем файл
```

## 14.8. Функции для работы с каталогами

- ❑ Для создания каталога служит функция `mkdir()`:

```
bool mkdir (string pathname [, int mode [, bool recursive
[, resource context]])
```

Первый параметр — это имя создаваемого каталога, а второй — права доступа к каталогу. Понятное дело, что у вас должно быть право на изменение каталога, в котором вы собираетесь создать новый подкаталог. Третий параметр нужно установить в `true`, если вы хотите создать дерево каталогов.

Примеры использования функции `mkdir()`:

```
mkdir("images", 777);
mkdir("gallery/upload", 777, true);
```

- ❑ Функция `rmdir()` удаляет пустой каталог:

```
rmdir("images");
```

- ❑ Функция `dirname()` возвращает имя каталога из пути, например:

```
$path = "/etc/passwd";
echo dirname($path);           // выведет /etc
```

- ❑ Для получения списка файлов каталога служит функция `scandir()`:

```
array scandir (string $directory [, integer $sorting_order])
```

Первый параметр задает имя каталога, второй — порядок сортировки (1 — обратная сортировка). Пример использования этой функции:

```
$dir    = '/home/den';
$files_1 = scandir($dir);
$files_2 = scandir($dir, 1);
```

```
print_r($files_1);  
print_r($files_2);
```

- Функция `scandir()` появилась только в PHP 5, в более «древних» версиях PHP для вывода содержимого каталога служили функции `opendir()` и `readdir()`:

```
$dir = "/home/den";  
  
$dh = opendir($dir);  
while (false !== ($filename = readdir($dh))) {  
    $files[] = $filename;  
}  
  
// сортировка массива files в алфавитном порядке  
sort($files);  
print_r($files);  
  
// обратная сортировка массива files  
rsort($files);  
print_r($files);
```

В настоящее время поддержка PHP 4 вообще не важна, т. к. с этой версии ушли даже самые «древние» хостинги. Используйте только функцию `scandir()`, иначе ваш код будет похож на динозавра, особенно когда уже во всю используется версия PHP 7 и происходит плавный переход на PHP 8. Вариант кода с функциями `opendir()` и `readdir()` приведен в книге сугубо в образовательных целях.

#### **ПРИМЕЧАНИЕ**

Много различных примеров использования функции `scandir()` вы найдете по адресу: <https://www.php.net/manual/ru/function.scandir.php>.



## ГЛАВА 15

# Вывод графических изображений средствами PHP

## 15.1. Библиотека GD

Библиотека GD предназначена для работы с изображениями и графическими файлами форматов JPEG, GIF, PNG, WBMP, XPM и WebP. Используя эту библиотеку, можно программно создавать изображения, сохранять их в разных форматах или сразу выводить в браузер.

Тем не менее, хотя PHP и имеет в наличии все функции, необходимые для создания графических шедевров, гораздо удобнее это делать в том же Adobe Photoshop. Ведь в графическом редакторе сразу видно, где окажется та или иная линия, а вот при работе с PHP вам придется в буквальном смысле слова представлять, как будет выглядеть картинка попиксельно (в качестве домашнего задания предлагаю вам программным способом нарисовать пингвина размером 100×100 пикселей, не прибегая к помощи Photoshop. Шутка!). Вот поэтому практически никто не создает в PHP изображения с нуля, а модифицирует уже готовые картинки, нарисованные в полноценном графическом редакторе. Предположим, вам требуется счетчик. В графическом редакторе вы создаете его фоновое изображение, а с помощью PHP просто в нужном месте выводите значение.

Сначала мы опишем некоторые очень полезные функции, необходимые нам для реализации графической галереи, а далее будут рассмотрены два практических примера: изменение размера изображения и нанесение на него водяных знаков.

### 15.1.1. Получение информации об изображении

Для определения формата графического файла служит функция `exif_imagetype()`, которой нужно передать имя графического файла:

```
int exif_imagetype (string filename)
```

Функция возвращает числовое значение, соответствующее формату файла (табл. 15.1).

Если формат файла определить не удалось, то функция возвращает `false`.

Таблица 15.1. Значения, возвращаемые функцией `exif_imagetype()`

Значение	Константа	Значение	Константа
1	IMAGETYPE_GIF	9	IMAGETYPE_JPC
2	IMAGETYPE_JPEG	10	IMAGETYPE_JP2
3	IMAGETYPE_PNG	11	IMAGETYPE_JPX
4	IMAGETYPE_SWF	12	IMAGETYPE_JB2
5	IMAGETYPE_PSD	13	IMAGETYPE_SWC
6	IMAGETYPE_BMP	14	IMAGETYPE_IFF
7	IMAGETYPE_TIFF_II	15	IMAGETYPE_WBMP
8	IMAGETYPE_TIFF_MM	16	IMAGETYPE_XBM

Еще одна очень полезная функция — `exif_read_data()`, позволяющая прочесть заголовки JPEG- и TIFF-файлов и вернуть ассоциативный массив, содержащий полную информацию о графическом файле:

```
array exif_read_data (string filename
                        [, string sections
                        [, bool arrays
                        [, bool thumbnail]]])
```

Первый параметр — это имя файла, второй — список секций (через запятую), которые нужно вернуть. Список разделов и содержащихся в них параметров графического файла приведен в табл. 15.2.

Таблица 15.2. Разделы и параметры графического файла

Раздел	Параметры
FILE	FileName — имя файла; FileSize — размер файла; FileDateTime — дата создания файла; SectionsFound — секции файла
COMPUTED	html — HTML-строка для тега IMG (<width> и <height>) Width — ширина картинки Height — высота
IFD0	Все данные с тегами IFD0
THUMBNAIL	Содержит миниатюру (уменьшенное изображение картинки), если она создана, и информацию о ней
COMMENT	Комментарии для графического файла (поддерживаются форматом JPEG)
EXIF	Детальная информация об изображении. Формируется, как правило, цифровыми фотоаппаратами

Третий параметр, `arrays`, определяет, будет ли каждый раздел оформлен в виде отдельного массива (`arrays = true`) или нет (`arrays = false`). Разделы `FILE`, `COMPUTED` и `THUMBNAİL` всегда формируются как массивы.

Последний параметр, `thumbnail`, определяет, нужно ли загружать саму миниатюру (`true`) или только информацию о ней (`false`).

Рассмотрим пример использования функции `exif_read_data()` (листинг 15.1).

**Листинг 15.1. Пример использования функции `exif_read_data()`**

```
<?php
$info = exif_read_data ('av.jpg','', true, false);

echo "<p>Раздел <b>FILE</b>";
echo "<br>Имя файла " . $info['FILE']['FileName'];
echo "<br>Размер файла " . $info['FILE']['FileSize'];
echo "<br>Дата создания " . $info['FILE']['FileDateTime'];
echo "<br>Секции " . $info['FILE']['SectionsFound'];
echo "<p>Раздел <b>COMPUTED</b>";
echo "<br>HTML-строка " . $info['COMPUTED']['html'];
echo "<br>Ширина " . $info['COMPUTED']['Width'];
echo "<br>Высота " . $info['COMPUTED']['Height'];
echo "<br>";
var_dump($info['COMPUTED']);
echo "<p>Раздел <b>IFD0</b><p>";
var_dump($info['IFD0']);
echo "<p>Раздел <b>THUMBNAİL</b><p>";
var_dump($info['THUMBNAİL']);

echo "<p>Раздел <b>COMMENT</b><p>";
var_dump($info['COMMENT']);

echo "<p>Раздел <b>EXIF</b><p>";
var_dump($info['EXIF']);

?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-1.php*), представленный в каталоге *Glava\_15* сопровождающего книгу электронного архива (см. *приложение 4*).

Результат выполнения сценария представлен на рис. 15.1.

```

Раздел FILE
Имя файла av.jpg
Размер файла 12768
Дата создания 1159095403
Секции ANY_TAG, IFD0, THUMBNAIL, EXIF, INTEROP

Раздел COMPUTED
HTML-строка width="100" height="75"
Ширина 100
Высота 75
array(10) ( ["html"]=> string(23) "width="100" height="75" ["Height"]=> int(75) ["Width"]=> int(100) ["IsColor"]=> int(1) ["ByteOrderMotorola"]=> int(0) ["ApertureFNumber"]=> string(5) "f2.8" ["UserComment"]=> string(1) " ["UserCommentEncoding"]=> string(9) "UNDEFINED" ["Thumbnail.FileType"]=> int(2) ["Thumbnail.MimeType"]=> string(10) "image/jpeg" )

Раздел IFD0
array(11) ( ["ImageDescription"]=> string(31) "OLYMPUS DIGITAL CAMERA." ["Make"]=> string(21) "OLYMPUS IMAGING CORP." ["Model"]=> string(12) "FE-120,X-700" ["Orientation"]=> int(1) ["XResolution"]=> string(4) "72/1" ["YResolution"]=> string(4) "72/1" ["ResolutionUnit"]=> int(2) ["Software"]=> string(27) "ACD Systems Digital Imaging" ["DateTime"]=> string(19) "2006:09:24 14:01:30" ["YCbCrPositioning"]=> int(1) ["Exif_IFD_Pointer"]=> int(277) )

Раздел THUMBNAIL
array(3) ( ["Compression"]=> int(6) ["JPEGInterchangeFormat"]=> int(2950) ["JPEGInterchangeFormatLength"]=> int(5236) )

Раздел COMMENT
NULL

Раздел EXIF

```

Рис. 15.1. Информация о JPEG-файле (фрагмент вывода)

## 15.1.2. Конвертирование графических форматов

Функция `jpeg2wbmp()` осуществляет преобразование файла из формата JPEG в формат Windows BMP:

```

int jpeg2wbmp (string jpegname,
               string wbmpname,
               int d_height,
               int d_width,
               int threshold)

```

Первый параметр — это имя файла изображения в формате JPEG, второй — имя результирующего BMP-файла. Параметры `d_height`, `d_width` задают соответственно высоту и ширину изображения BMP-формата. Последний параметр обычно можно установить в ноль.

Преобразование файла формата PNG в формат Windows BMP осуществляет функция `png2wbmp()`. Ее параметры аналогичны параметрам предыдущей функции:

```

int png2wbmp (string pngname,
               string wbmpname,
               int d_height,
               int d_width,
               int threshold)

```

Но если необходимо преобразовать файл в формат, отличный от BMP, придется создавать конвертер вручную. К счастью, это очень просто. Рассмотрим следующий пример (листинг 15.2).

**Листинг 15.2. Простейший графический конвертер**

```
<?php
$image = ImageCreateFromPng("image.png");
Header("Content-type: image/gif");
ImageGif($image);
ImageDestroy($image);
?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-2.php*), представленный в каталоге *Glava\_15* сопровождающего книгу электронного архива (см. приложение 4).

Да, именно так выглядит простейший графический конвертер. Обратите внимание — мы с помощью функции `ImageCreateFromPng()` открываем PNG-файл, а затем с помощью функции `ImageGif()` выводим это изображение в браузер как GIF-файл. Функция `Header()` информирует браузер, что мы сейчас будем выводить GIF-изображение, а не какое-то другое. Если наш PHP-файл называется *15-2.php*, то в теге `IMG` вы смело можете написать:

```
<img src=http://localhost/15-2.php>
```

Браузер отобразит GIF-картинку.

Если вам нужно не выводить картинку в браузер, а сохранить ее на диске в другом формате, следует указать еще один параметр для функции `ImageGif()` — имя файла:

```
ImageGif($image, 'image.gif');
```

Понятно, что функция `Header()` уже не потребуется, поскольку мы указали имя файла, и вывод функции будет направлен в этот файл, а не во входной поток браузера. Другие функции, пригодные для конвертирования графических форматов, приведены в табл. 15.3.

**Таблица 15.3. Функции загрузки и сохранения графических файлов**

Функция	Описание
<code>resource imagecreate (int x_size, int y_size)</code>	Создает пустое изображение указанного размера. Именно эту функцию нужно использовать, если у вас появится желание нарисовать свой "шедевр" с нуля
<code>resource imagecreatefromgd (string filename)</code>	Загружает изображение формата GD из файла или URL
<code>resource imagecreatefromgd2 (string filename)</code>	Загружает изображение формата GD2 из файла или URL
<code>resource imagecreatefromgif (string filename)</code>	Загружает изображение из файла формата GIF или URL
<code>resource imagecreatefromjpeg (string filename)</code>	То же, JPEG

Таблица 15.3 (окончание)

Функция	Описание
<code>resource imagecreatefrompng (string filename)</code>	То же, PNG
<code>resource imagecreatefromwbmp (string filename)</code>	То же, WBMP
<code>resource imagecreatefromxbm (string filename)</code>	То же, XBM
<code>resource imagecreatefromxpm (string filename)</code>	То же, XPM
<code>int imagegd (resource image [, string filename])</code>	Сохраняет изображение на диск в формате GD. Если имя файла не задано, то картинка передается на стандартный вывод, т. е. в браузер. Аналогично работают остальные функции сохранения графических файлов
<code>int imagegd2 (resource image [, string filename [, int chunk size [, int type]]])</code>	Сохраняет изображение на диск в формате GD2
<code>int imagegif (resource image [, string filename])</code>	Сохраняет изображение на диск в формате GIF
<code>int imagejpeg (resource image [, string filename [, int quality]])</code>	Сохраняет изображение на диск в формате JPEG. Последний параметр, <code>quality</code> , задает качество изображения (от 0 до 100). Чем оно выше, тем больше размер файла. По умолчанию качество равно 75
<code>int imagepng (resource image [, string filename])</code>	Сохраняет изображение на диск в формате PNG
<code>int imagewbmp (resource image [, string filename [, int foreground]])</code>	Сохраняет изображение на диск в формате WBMP. Параметр <code>foreground</code> задает цвет переднего плана (по умолчанию черный, т. е. <code>foreground = 0</code> ). Для вывода такого изображения в браузер нужно передать заголовок: <code>header('Content-type: image/vnd.wap.wbmp ');</code>
<code>int imagexbm (resource image, string filename [, int foreground])</code>	Сохраняет изображение на диск в формате XBM. Параметр <code>foreground</code> задает цвет переднего плана (по умолчанию 0)

### 15.1.3. Вывод текста поверх картинки. Задание цвета

Для вывода строки служит функция `imagestring()`:

```
int imagestring (resource image,
                int font,
                int x,
                int y,
                string s,
                int color)
```

Эта функция выводит строку *s*, расположенную по горизонтали, начиная с координат *x* и *y* (цвет букв задается параметром *color*). Параметр *font* определяет шрифт, который будет использоваться для вывода строки. Параметры *x* и *y* задают координаты верхнего левого угла первого символа.

Если требуется вывести строку вертикально, то вызывается функция `imagestringup()`:

```
int imagestringup (resource image,
                  int font,
                  int x,
                  int y,
                  string s,
                  int col)
```

Параметры *x* и *y* задают координаты верхнего левого угла самого верхнего (первого) символа строки.

#### **ПРИМЕЧАНИЕ**

Для вывода одного символа поверх картинки можно использовать функции `imagechar()` и `imagecharup()`. Но, как правило, в этих функциях нет необходимости — из соображений универсальности для вывода строки, состоящей из одного символа, лучше применить функции `imagestring()` и `imagestringup()`.

Задать цвет можно с помощью функции `imagecolorallocate()`:

```
int imagecolorallocate (resource image,
                       int red,
                       int green,
                       int blue)
```

Цвет задается в формате RGB (Red, Green, Blue). В табл. 15.4 приведены RGB-значения для основных цветов.

**Таблица 15.4.** Основные шестнадцать цветов

Название	RGB-значение
Aqua (морская волна)	(000,255,255)
Black (черный)	(000,000,000)
Blue (голубой)	(000,000,255)
Fuchsia (фуксия)	(255,000,255)
Gray (серый)	(128,128,128)
Green (зеленый)	(000,128,000)
Lime (ярко-зеленый)	(000,255,000)
Maroon (темно-бордовый)	(128,000,000)
Navy (темно-синий)	(000,000,128)
Olive (оливковый)	(128,128,000)
Purple (фиолетовый)	(128,000,128)

Таблица 15.4 (окончание)

Название	RGB-значение
Red (красный)	(255,000,000)
Silver (серебряный)	(192,192,192)
Teal (серо-зеленый)	(000,128,128)
White (белый)	(255,255,255)
Yellow (желтый)	(255,255,000)

Получить цвет можно и из самого изображения. Для этого служит функция `imagecolorat()`, которая возвращает значение цвета пиксела с координатами `$x` и `$y` из картинки `$image`:

```
int imagecolorat ( resource $image , int $x , int $y )
```

Эта функция аналогична инструменту «Пипетка» в графических редакторах.

Вот небольшой пример вывода строки красным цветом:

```
$color = imagecolorallocate ($image, 255, 0, 0);
imagestring ($image, 1, 5, 5, 'Test', $color);
```

Параметр `font` функций `imagestring()` и `imagestringup()` задает шрифт, которым будет выведена строка. По умолчанию вам доступно пять шрифтов с номерами от 1 до 5 (1 — самый мелкий шрифт, 5 — самый крупный).

Нужно отметить, что у функций `imagestring()` и `imagestringup()` есть один большой недостаток — они не позволяют вывести русский текст. Обычно это не критично, поскольку выводятся различные числовые значения — например, показания счетчика, но если вам потребуются символы кириллицы, тогда вам не обойтись без функции `imagettftext()`:

```
array imagettftext (resource image,
    int size,
    int angle,
    int x,
    int y,
    int color,
    string fontfile,
    string text)
```

Функция `imagettftext()` позволяет вывести текст заданным TTF-шрифтом (в каталоге `Windows\Fonts` вы найдете TTF-шрифты на любой вкус). Файл шрифта задается параметром `fontfile`, размер шрифта (в пикселах) — параметром `size`.

Параметр `angle` задает угол поворота текста против часовой стрелки. Параметры `x`, `y` задают координаты точки, с которой начинается вывод текста.

Параметр `color` — это идентификатор цвета, а `text` — строка, которую нужно вывести.

Вам необходимо помнить, что большинство TTF-шрифтов рассчитаны на использование кодировки Windows-1251, поэтому строка `text` должна быть написана именно в этой кодировке.

Рассмотрим небольшой пример вывода текста на русском языке (листинг 15.3).

#### Листинг 15.3. Вывод текста на русском языке

```
<?php

// создаем пустое изображение
$image = imagecreate(500, 500);

// фон (белый)
$back = imagecolorallocate($image, 255, 255, 255);

// цвет шрифта (синий)
$color = imagecolorallocate($image, 0, 0, 255);

// выводим строку "Привет"
// размер шрифта — 20 пунктов
// цвет — синий
// гарнитура — Arial (файл arial.ttf находится в одном каталоге
// со сценарием)
// угол — 0
// 10, 10 — координаты верхнего левого угла первой буквы
imagettftext($image, 20, 0, 10, 10, $color, "arial.ttf", "Привет");

// выводим картинку в формате PNG в браузер
Header('Content-type: image/png');

ImagePng($image);

?>
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-3.php*), представленный в каталоге *Glava\_15* сопровождающего книгу электронного архива (см. приложение 4).

## 15.1.4. Прозрачность

Частенько приходится задавать прозрачный цвет, чтобы он не отображался, а заменялся цветом фона. Самый простой пример: у нас есть изображение на черном фоне, а фон нашего сайта — белый (или синий, зеленый — не важно, но не черный). Поэтому мы можем задать черный цвет прозрачным, и он не будет отображаться. Для этого существует функция `imagecolortransparent()`:

```
int imagecolortransparent (resource image [, int color])
```

Всё, что нужно, — это задать цвет, который необходимо сделать прозрачным. Вот небольшой пример:

```
// делаем черный цвет прозрачным
$black_color = imagecolorallocate($image, 0, 0, 0);
imagecolortransparent ($image, $black_color);
```

Помните, что если другая часть изображения (а не только фон) тоже черная, то она также станет прозрачной.

## 15.1.5. Рисование графических примитивов

В составе библиотеки GD имеются функции, позволяющие нарисовать графические примитивы: линии, прямоугольники, эллипсы и т. д. Начнем с рисования одного пиксела:

```
bool imagesetpixel (resource $image, int $x, int $y, int $color)
```

Первый параметр — идентификатор картинki, далее координаты пиксела, последний параметр — цвет.

Далее нарисуем линию:

```
bool imageline (resource $image, int $x1, int $y1, int $x2, int $y2, int $color)
```

Первый параметр — идентификатор изображения, затем следуют координаты линии, которая будет нарисована от точки (x1, y1) до (x2, y2). Последний параметр задает цвет линии.

### **ПРИМЕЧАНИЕ**

Для рисования пунктирной линии ранее применялась функция `imagedashedline()`. Теперь она упразднена, и вместо нее нужно использовать комбинацию функций `imagesetstyle()` и `imageline()`.

Перед рисованием линии можно задать ее стиль (функция `imagesetstyle`), толщину (`imagesetthickness`) и кисть (`imagesetbrush`), которой будет нарисована линия. Начнем с функции `imagesetstyle()`:

```
bool imagesetstyle (resource $image, array $style)
```

Первый параметр — изображение, второй — массив пикселей. Рассмотрим пример. Нарисуем пунктирную линию:

```
/* 5 красных и 5 белых пикселей */
$wh = imagecolorallocate($im, 255, 255, 255);
$red = imagecolorallocate($im, 255, 0, 0);
$style = array($red, $red, $red, $red, $red, $wh, $wh, $wh, $wh, $wh);
imagesetstyle($im, $style);
imageline($im, 0, 0, 100, 100, IMG_COLOR_STYLED);
```

Толщина линии устанавливается функцией `imagesetthickness()`:

```
bool imagesetthickness (resource $image, int $thickness)
```

Тут все просто: первый параметр — изображение, а второй — толщина в пикселах. Осталось разобраться, как устанавливается кисть для рисования:

```
bool imagesetbrush ( resource $image , resource $brush )
```

Кисть — это, по сути, другое изображение, которым будет нарисована линия. Небольшой пример:

```
// загружаем кисть
$brush = imagecreatefrompng('./brush.png');
// создаем главное изображение
$im = imagecreatetruecolor(100, 100);
// устанавливаем кисть
imagesetbrush($im, $brush);
// рисуем линию
imageline($im, 50, 50, 50, 60, IMG_COLOR_BRUSHED);
```

Когда мы знаем, как нарисовать точку и линию, можно перейти к более сложным примитивам: прямоугольнику, эллипсу и дуге:

```
bool imagerectangle (resource $image, int $x1, int $y1, int $x2, int $y2,
int $color)
bool imageellipse (resource $image, int $cx, int $cy, int $width, int $height,
int $color)
bool imagearc (resource $image, int $cx, int $cy, int $width, int $height,
int $start, int $end, int $color)
```

Как обычно, первый параметр всех этих функций задает идентификатор рисунка, последний — цвет, которым будет нарисован примитив. В случае с прямоугольником (функция `imagerectangle()`) вам нужно задать координаты верхнего левого  $x_1$  и  $y_1$  и нижнего правого ( $x_2$  и  $y_2$ ) углов.

Эллипс (`imageellipse()`) тоже рисуется несложно. Параметры  $cx$  и  $cy$  — это координаты центра,  $width$  и  $height$  — ширина и высота эллипса (в пикселах) соответственно.

Функция `imagearc()` рисует дугу (часть эллипса) с центром в  $cx$ ,  $cy$  (у самого верхнего левого пиксела координаты 0, 0),  $width$  и  $height$  — ширина и высота воображаемого эллипса. Параметры  $start$  и  $end$  — это начальная и конечная точки, указываются в градусах. Помните, что 0 находится в позиции «три часа», а сама дуга рисуется против часовой стрелки.

У функций `imagerectangle()`, `imageellipse()` и `imagearc()` есть filled-аналоги:

```
bool imagefilledrectangle (resource $image, int $x1, int $y1, int $x2, int $y2,
int $color)
bool imagefilledellipse (resource $image, int $cx, int $cy, int $width,
int $height, int $color)
bool imagefilledarc (resource $image, int $cx, int $cy, int $width,
int $height, int $start, int $end, int $color, int $style)
```

Разница в том, что обычные функции рисуют неокрашенные версии фигур, а их filled-версии окрашивают фигуры внутри цветом  $color$ .

Самая сложная функция — `imagepolygon()`, рисующая многоугольник. Прототип этой функции следующий:

```
bool imagepolygon (resource $image, array $points, int $num_points, int $color)
```

Первый и последний параметры, надеюсь, понятны. Массив `$points` задает координаты точек в формате:

```
points[0]    = x0
points[1]    = y0
points[2]    = x1
points[3]    = y1
...
```

То есть четный элемент массива — это координата *x* точки, а нечетный элемент — координата *y*. Параметр `$num_points` задает общее количество точек (число вершин многоугольника).

Аналог этой функции — `imagefilledpolygon()`. Рисует окрашенный многоугольник, если, конечно, он замкнутый.

### 15.1.6. Поворот изображения

Для поворота изображения служит функция `imagerotate()`:

```
resource imagerotate (resource $image, float $angle, int $bgd_color [, int  
$ignore_transparent = 0 ])
```

Как обычно, первый параметр — это изображение. Второй параметр — угол поворота. Центром поворота всегда является центр изображения. Повернутое изображение может отличаться размером от оригинального. Если при повороте остается пустая зона, то она окрашивается цветом `$bgcolor`. Если последний параметр не равен 0, то прозрачность игнорируется.

## 15.2. Изменение размера изображения

Вспомним, как организована любая графическая галерея. Сначала выводятся миниатюрные изображения. Как правило, размер миниатюры равен 100×80 пикселей. При щелчке на миниатюре выводится увеличенное промежуточное изображение, размер которого зависит от самого изображения и от дизайнера галереи — важно, чтобы картинка вписывалась в окно браузера. Промежуточный размер обычно равен 480×280 или 640×480 пикселей. Еще один щелчок — и пользователь может увидеть оригинальное изображение в полном размере. Оригинальное изображение обычно открывается в отдельном окне, поэтому о дизайне никто не думает — важно вывести изображение в таком виде, в каком пользователь поместил его в галерею.

В PHP нет встроенной функции, позволяющей изменять размер изображения, но мы можем написать ее самостоятельно (листинг 15.4).

**Листинг 15.4. Функция изменения размера картинки**

```
function imageresize($src, $dest, $width, $height, $quality) {  
  
    $im=imagecreatefromjpeg($src);  
    $im1=imagecreatetruecolor($width, $height);  
    imagecopyresampled($im1,$im,0,0,0,0,$width,$height,imagesx($im),imagesy($im));  
  
    imagejpeg($im1,$dest,$quality);  
  
    imagedestroy($im);  
    imagedestroy($im1);  
}
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-4.php*), представленный в каталоге *Glava\_15* сопровождающего книгу электронного архива (см. *приложение 4*).

Разберемся с параметрами функции. Первый параметр — это имя исходного (большого) файла. Второй параметр — имя результирующего (маленького) файла. Параметры `width` и `height` задают ширину и высоту результирующего файла. Параметр `quality` задает качество результирующего JPEG-файла.

В этой функции мы задействуем вспомогательные функции, входящие в библиотеку GD:

- ☐ `imagesx()` — возвращает ширину изображения;
- ☐ `imagesy()` — возвращает высоту изображения;
- ☐ `imagecopyresampled()` — копирует прямоугольную часть одного изображения на другое изображение, при этом изменяет размер и следит за качеством изображения — чтобы при уменьшении размера картинки это не отразилось на ее качестве.

Внимательное изучение кода нашей функции (см. листинг 15.4) показывает, что фактически изменяет размер изображения именно функция `imagecopyresampled()`, копирующая прямоугольные области с одного изображения на другое. Наша функция только подготавливает все для ее работы.

Вот пример вызова этой функции:

```
imageresize('big_file.jpg','big_file_thumb.jpg',160,100,75);
```

Созданная нами функция работает только с JPEG-файлами, но при желании вы без проблем можете переделать ее для работы с GIF- и PNG-файлами.

На рис. 15.2 изображен результат работы нашей функции на примере реально работающего проекта.



Рис. 15.2. Вывод миниатюр

## 15.3. Создание водяных знаков

Водяные знаки на банкнотах защищают их от подделки. Водяные знаки на изображениях позволяют защитить изображения от несанкционированного копирования. Однажды с подобной проблемой ко мне обратился один мой хороший знакомый. Он всю ночь сканировал каталог с продукцией, для некоторых товаров делал авторские фотографии, а через пару дней обнаружил свои же изображения на сайте конкурента. Непорядок!

Защитить изображение можно путем нанесения поверх него какой-то надписи. Но если надпись нанести по центру, она будет перекрывать само изображение, что нежелательно. От надписи, нанесенной внизу или сверху (слева или справа — вертикально), очень просто избавиться в любом самом примитивном графическом редакторе.

А вот от водяного знака избавиться будет сложно. Он полупрозрачный, и его можно поместить по центру нашей картинки — таким образом и картинку защитим, и изображенный на ней объект не будет закрыт нашей надписью.

Чтобы не изобретать велосипед заново, в Интернете я нашел класс `watermark`, который и пригодится нам для создания водяных знаков. Рассмотрим листинг 15.5, который использует класс `watermark` для нанесения водяного знака.

**Листинг 15.5. Нанесение водяного знака**

```

<?php
// подключаем класс
include 'api.watermark.php';

// создаем объект wmark
$wmark = new watermark();
// загружаем основное изображение
$main_img = imagecreatefromjpeg('test.jpg');

// загружаем водяной знак
$watermark_img = imagecreatefrompng('./watermark.png');
// накладываем водяной знак на оригинальную картинку
$res_img = $wmark->create_watermark($main_img, $watermark_img, 66);

// выводим "защищенную" картинку (качество 50%)
header('Content-Type: image/jpeg');
header('Content-Disposition: inline; filename=' . $_GET['src']);
imagejpeg($res_img, '', 50);

?>

```

**Рис. 15.3.** Изображение с водяным знаком

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *15-5.php*), представленный в каталоге *Glava\_15* сопровождающего книгу электронного архива (см. *приложение 4*). Код класса *watermark* также находится в этом каталоге (файл *api.watermark.php*).

Как видите, все достаточно просто. Сначала создаем объект *\$wmark*, затем загружаем исходную картинку, после этого загружаем картинку, которая будет использоваться в качестве водяного знака. Картинка водяного знака должна быть в формате PNG, без фона (фон — прозрачный), а поверх прозрачного фона наносится любая надпись — водяной знак. Уровень прозрачности водяного знака задается последним параметром метода *create\_watermark()*. В нашем случае уровень прозрачности равен 66.

После наложения водяного знака все, что остается нам, — это вывести изображение в браузер. Результат работы сценария показан на рис. 15.3.

## 15.4. Поддержка графического формата WebP

О формате WebP хочется поговорить отдельно. Во-первых, это довольно молодой графический формат, который только набирает обороты. Во-вторых, он разработан компанией Google, поэтому успех ему гарантирован — при поддержке такого гиганта, как Google. В-третьих, раз разработчики PHP добавили поддержку этого формата в библиотеку GD, значит, считают его важным и перспективным. Ведь заметьте, что есть много других графических форматов, которые библиотекой GD не поддерживаются — видимо, за ненадобностью.

Формат WebP обеспечивает значительное сжатие графических изображений — как с потерей, так и без потери качества. По сравнению с форматом JPEG формат WebP обеспечивает то же качество изображения, но позволяет экономить до 25 % дискового пространства. Такая экономия особенно чувствуется при хранении в этом формате фотографий высокого качества (когда размер файла составляет несколько мегабайтов). Подробно описывать здесь особенности формата WebP я не стану, а предоставлю ссылку, пройдя по которой вы сможете сравнить качество форматов JPEG и WebP, а также занимаемое изображениями в этих форматах дисковое пространство: <http://www.andrewmunsell.com/blog/jpg-vs-webp>.

Полную информацию о формате WebP можно получить на сайте разработчиков: <https://developers.google.com/speed/webp/>.

Конечно, никто не спорит — в настоящее время формат JPEG все еще популярен и будет таковым оставаться в ближайшие годы. Однако формат WebP, учитывая, что он спонсируется Google, нельзя просто так списывать со счета.

Чтобы убедиться, что ваша версия PHP поддерживает формат WebP, вызовите функцию *phpinfo()* и убедитесь, что в секции *gd* прописана поддержка WebP (*WebP support enabled*).

Для вывода изображений в браузер или файл в формате WebP служит следующая функция:

```
bool imagewebp ( resource $image , string $filename )
```

Первый параметр — это уже сформированное средствами PHP изображение. Вторым параметром — имя файла. Если этот параметр не указан или равен `NULL`, будет произведен вывод в браузер.

Загрузить изображение в формате WebP с диска можно функцией `imagecreatefromwebp()`:

```
resource imagecreatefromwebp ( string $filename )
```

Используя эти две функции: `imagewebp()` и `imagecreatefromwebp()` — вы можете загружать и сохранять изображения в формате WebP. А после загрузки изображения применять к нему любые функции из библиотеки GD — например, добавить текст, нарисовать поверх геометрическую фигуру и т. д.



## ГЛАВА 16

# Работа с сетевыми сокетами в PHP. Сетевые функции

### 16.1. Еще раз о том, что такое сокет

Все мы знаем, как работать с файлами в PHP. Функцией `fopen()` мы открываем файл. Если файл открыт успешно, то функция возвращает его дескриптор, через который производятся все дальнейшие операции с файлом (чтение, запись).

Сокет очень похож на дескриптор файла, но он является дескриптором соединения, т. е. служит для передачи файлов по сети. Примечательно, что с сетевым соединением в PHP можно работать как с обычным файлом, с помощью стандартных функций для работы с файлами.

Но все равно не следует путать сокеты и файловые дескрипторы (ведь то, что с сокетами осуществляется работа посредством файловых функций, многих вводит в заблуждение). Как мы помним, функция `fopen()` умеет открывать файлы на удаленных серверах, но ведь сокеты используются не для работы с файлами, а для обмена данными с различными серверами. С помощью сокетов можно подключиться к любому TCP-серверу, например к веб-, FTP-серверу, почтовому и т. д. Конечно, функции PHP и так могут работать с веб-, FTP-серверами, а для работы с почтовыми серверами уже есть готовые классы. Так для чего нужны сокеты? Для реализации нестандартного обмена. Предположим, что вам необходимо обменяться данными с собственным сервером, реализующим свой собственный протокол передачи данных, который знаете только вы. Вот тут вам и пригодятся сокеты — ведь можно со 100%-ной уверенностью сказать, что класса для работы с вашим сервером точно в Сети не появится, если, конечно, его не сделаете вы сами.

### 16.2. Функция *fsockopen()*

Функция `fsockopen()` открывает сокет, т. е. устанавливает соединение с удаленным сервером:

```
fsockopen(string $host, int $port [, int $err] [, string $err_msg])
```

Назначение параметров, думаю, понятно. Первый параметр — это имя или IP-адрес удаленного узла, второй — порт службы. Последние два параметра не обязательны.

Если вы их указали, то в переменную `$err` будет записан код ошибки, а во вторую — текст с сообщением об ошибке.

После того как мы открыли сокет, с ним можно работать с помощью обычных функций для работы с файлами: `fwrite()`, `fread()`, `fgets()`, `fputs()` и др. Для вашего удобства прототипы и описания этих четырех функций сведены в табл. 16.1.

Таблица 16.1. Прототипы функций `fwrite()`, `fread()`, `fgets` и `fputs()`

Функция	Описание
<code>string fgets ( resource descriptor [, int lenght] )</code>	Читает строку из файла (сокета), первый параметр — это дескриптор файла (сокета), второй — длина строки в байтах (по умолчанию длина равна 1024 байтам)
<code>int fwrite ( resource handle, string string [, int length] )</code>	Записывает в файл (сокет), заданный дескриптором, строку, переданную с помощью второго параметра. Если указана длина строки, то запись будет остановлена после того, как будет записано указанное количество байтов (или будет достигнут конец строки). Функция <code>fputs()</code> является псевдонимом функции <code>fwrite()</code>
<code>int fputs ( resource handle, string string [, int length] )</code>	См. функцию <code>fwrite()</code>
<code>string fread ( resource handle, int length )</code>	Читает указанное количество байтов из файла (сокета), заданного дескриптором. Используется преимущественно для чтения бинарных данных

Если вы помните, то при вызове функции `fopen()` нужно было задать режим открытия файла. Сокет же открывается для двухстороннего обмена, т. е. и для чтения, и для записи.

## 16.3. Примеры работы с сокетами

### 16.3.1. Работаем с протоколом HTTP

Рассмотрим небольшой пример, демонстрирующий общую концепцию работы с сокетами. Мы подключимся к веб-серверу, передадим ему запрос и прочитаем ответ сервера. Как уже было отмечено, сокет открывается как для чтения, так и для записи, а вся работа с ним осуществляется в режиме «запрос–ответ». Вы отправляете запрос серверу и сразу же читаете его ответ, затем ответ необходимо проанализировать (вдруг он содержит сообщение об ошибке) или просто вывести в браузер (листинг 16.1).

Листинг 16.1. Работа через сокет с протоколом HTTP

```
<?php
// открываем сокет, устанавливаем соединение с сервером localhost,
// порт 80
$sock = fsockopen("localhost",80);
```

```
// отправляем команду GET — получить index.html
fputs($sock, "GET /index.html HTTP/1.0\n\n");

// читаем ответ сервера (1024 байтов) и сразу выводим его в браузер
while (!feof($sock))
    echo fgets($sock);

// закрываем сокет
fclose($sock);

?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *16-1.php*), представленный в каталоге *Glava\_16* сопровождающего книгу электронного архива (см. приложение 4).

Для того чтобы обмениваться информацией с сервером, вам нужно знать протокол обмена. В Интернете можно найти специальные документы — запросы комментариев (RFC, Request For Comments), в которых описаны все стандартные протоколы. В табл. 16.2 приведены номера документов RFC и ссылки на описание протоколов.

Таблица 16.2. Некоторые RFC-документы

Номер	Описание	Ссылка
2068, 2616	Протокол HTTP/1.1. Документ 2616 — более свежий, чем 2068	<a href="http://www.faqs.org/rfcs/rfc2616.html">http://www.faqs.org/rfcs/rfc2616.html</a>
1870	Протокол SMTP	<a href="http://www.faqs.org/rfcs/rfc1870.html">http://www.faqs.org/rfcs/rfc1870.html</a>
1939	Протокол POP3	<a href="http://www.faqs.org/rfcs/rfc1939.html">http://www.faqs.org/rfcs/rfc1939.html</a>

Рассмотрим еще один пример использования сокетов. Напишем сценарий, который подключится к серверу **whois.ripe.net**, передаст интересующий нас IP-адрес и выведет информацию об этом IP-адресе, полученную от **whois.ripe.net** (листинг 16.2). Вызывать сценарий нужно так:

```
http://<сервер>/soc.php?ip=<IP-адрес>
```

Листинг 16.2. Сценарий soc.php

```
<?php

if ($ip!="")
{
    // подключаемся к whois.ripe.net, порт 43
    $sock = fsockopen ("whois.ripe.net", 43, $errno, $errstr);
    if (!$sock)
    {
        // в случае ошибки выводим сообщение об ошибке
        echo("$errno($errstr)");
    }
}
```



```
// Создаем функции для отправки команды и чтения ответа
// Функция отправки команды
function server_write($data)
{
    global $s, $CRLF;
    fwrite($s, $data.$CRLF, strlen($data)+2);
}

// Функция чтения ответа
function server_read()
{
    $return = "";
    global $line, $CRLF, $loops, $s;
    while((strpos($return, $CRLF) === FALSE OR substr($line,3,1) !== ' ')
    AND $loops < 100){
        $line = fgets($s, 512);
        $return = $line;
        echo $line;
        $loops++;      // читаем максимум 100 строк
    }
}

// Открываем сокет для работы с сервером
$s = fsockopen($server,"25", $r, $m);

// Читаем приветствие сервера
server_read();

// Представляемся серверу
server_write("HELO $myhost");

// Читаем ответ
server_read();

// Указываем адрес отправителя
server_write("MAIL FROM: $from");

// Читаем ответ
server_read();

// Указываем адрес получателя
server_write("RCPT TO: $to");

// Читаем ответ
server_read();

// Начинаем отправлять сообщение
server_write("DATA");
```

```
// Читаем ответ
server_read();

// Отправляем сообщение "Hello", затем эмулируем нажатие Enter . Enter
server_write("Hello".$CRLF.".".$CRLF);

// Читаем ответ
server_read();

// Закрываем соединение
server_write("QUIT");

// закрываем соединение
fclose($s);
?>
```

Не пытайтесь запустить сценарий `send_mail.php` как есть — у вас он не заработает. Сначала вам нужно отредактировать глобальные переменные: указать реальные имена узлов (сервера и ваш), имя отправителя и получателя — иначе ничего не получится:

```
$server = "localhost";           // укажите имя сервера
$port = 25;                     // порт сервера
$myhost = "localhost";         // ваш хост
$from = "user@localhost";      // отправитель
$to = "user2@localhost";       // получатель
```

Этот сценарий «как есть» идеально будет работать, если вы запустите его на UNIX-хостинге, где запущена программа `sendmail` (или любая другая) и существуют пользователи `user` и `user2`.

### 16.3.3. Простейший клиент/сервер

Сейчас мы сделаем то, что обычно на PHP никто не делает, а именно — реализуем простейшее приложение клиент/сервер. Это будет однопоточный сервер (на PHP можно написать и многопоточный, но это выходит за рамки книги для начинающих), к которому сможет подключиться клиент и передать данные.

Начнем с приложения сервера — файла `server.php` (листинг 16.4).

#### Листинг 16.4. Сценарий `server.php`

```
<?php
// Создаем сокет, и слушаем его
// Мы должны прописать путь соединения примерно такой
// "tcp://127.0.0.1:8080" - это означает, что мы используем
// порт 8080 на 127.0.0.1 (локальный компьютер)
```

```
$socket = stream_socket_server(
    "unix:///tmp/simple-socket",
    $errno,
    $errstr,
    STREAM_SERVER_BIND | STREAM_SERVER_LISTEN
);

// Если сокет не создан
if ( ! $socket )
{
    // Выводим сообщение об ошибках
    echo "$errstr ( $errno )\n";
}
else
{
    // Поступающее соединение от клиента принимаем
    while ( $conn = stream_socket_accept($socket) )
    {
        $data = "Time = " . date("H:i:s") . PHP_EOL;
        fwrite( $conn, $data );
        fclose( $conn );
    }
    fclose($socket);
}
```

Как видите, все довольно просто. Теперь напомним приложение-клиент — файл `client.php` (листинг 16.5).

#### Листинг 16.5. Сценарий `client.php`

```
<?php
// Тут все аналогично tcp://127.0.0.1:8080
$socket = stream_socket_client(
    "unix:///tmp/simple-socket",
    $errno,
    $errstr,
    30,
    STREAM_CLIENT_ASYNC_CONNECT|STREAM_CLIENT_CONNECT
);

// Если сокет не создан, выводим сообщение об ошибке
if ( ! $socket )
{
    echo "$errstr ($errno)\n";
}
else
{
    // Здравствуемся с сервером
    fwrite( $socket, "HI" );
}
```

```
// Читаем ответ сервера и выводим его
while ( ! feof( $socket ) )
{
    echo fgets( $socket, 1024 );
}
fclose( $socket );
}
```

Сначала на одной из консолей запустите сценарий `server.php`, а затем перейдите на другую консоль (или откройте второе окно терминала) и запустите `client.php`.

## 16.4. Блокирующий и неблокирующий режимы сокета

Существуют два режима работы сокета: блокирующий и неблокирующий. В первом режиме функции чтения из сокета (`fgets()` и `fread()`) будут ждать до тех пор, пока не прочитают данные из сокета. Во втором, если данных в сокете нет, то функции чтения завершают свою работу.

Для переключения в блокирующий режим нужно вызвать функцию `socket_set_blocking()`, указав дескриптор соединения и передав `true` в качестве второго параметра:

```
socket_set_blocking($sock, TRUE);
```

Отключить блокирующий режим можно так:

```
socket_set_blocking($sock, FALSE);
```

### **ПРИМЕЧАНИЕ**

Функция `socket_set_blocking()` появилась только в версии PHP 4.3. Если у вас версия PHP младше, то данной функции у вас не будет.

## 16.5. DNS-функции

Для разрешения IP-адреса в доменное имя и наоборот служат две следующие функции:

```
string gethostbyaddr ( string $ip_address )
string gethostbyname ( string $hostname )
```

Первая функция возвращает доменное имя узла по заданному IP-адресу, например:

```
<?php
// получаем IP-адрес удаленного пользователя, преобразуем в имя и выводим
$hostname = gethostbyaddr($_SERVER['REMOTE_ADDR']);

echo $hostname;
?>
```

Вторая функция возвращает IP-адрес по заданному доменному имени:

```
<?php
$ip = gethostbyname('example.com');

echo $ip;

?>
```

Есть еще и третья DNS-функция:

```
array gethostbyname1 ( string $hostname )
```

Дело в том, что одному и тому же доменному имени могут соответствовать несколько IP-адресов. С помощью функции `gethostbyname1()` мы можем получить все эти IP-адреса. Вот пример использования этой функции:

```
<?php

$hosts = gethostbyname1('www.example.com');

print_r($hosts);

?>
```



## ГЛАВА 17

# Собственные функции

### 17.1. Зачем нужны собственные функции?

До сих пор мы рассматривали встроенные функции PHP, а теперь настало время поговорить о создании собственных функций. Спрашивается, зачем нужны собственные функции? Ответ прост — если вы не можете найти нужную вам функцию среди стандартных функций PHP, тогда вам остается только разработать собственную.

Например, в PHP нет функции для проверки правильности адреса электронной почты. Используя функции для работы с регулярными выражениями, вы можете с легкостью создать такую функцию. Вот, например:

```
function is_email($email) {  
    if (! preg_match( '/^[A-Za-z0-9!#$%&\'*+,-/=/?^`{|}~]+@[A-Za-z0-9-]+(\.[A-Za-z0-9-]+)+[A-Za-z]$/', $email)) {  
        return false;  
    }  
    else {  
        return true;  
    }  
}
```

Конечно, если вам надо проверить правильность адреса только один раз и в одном сценарии, можно обойтись и без собственной функции, а просто вызвать функцию `preg_match()` и проанализировать ее результат. Но если вам нужно вызывать этот код несколько раз в разных частях сценария и даже в разных файлах, то целесообразно создать свою функцию и вынести ее в отдельный PHP-сценарий, который можно будет подключить к любому сценарию, где требуется функция проверки почты.

### 17.2. Особенности функций в PHP

PHP — весьма гибкий язык программирования, и эта гибкость проявляется во всем, в том числе и при объявлении функций:

- ❑ вы можете объявить произвольное число параметров и вызвать функцию с произвольным числом параметров. Отсутствующие аргументы будут заменены аргументами по умолчанию, которые вы укажете при объявлении функции;
- ❑ вы можете использовать как локальные, так и глобальные переменные (области видимости переменных будут рассмотрены далее);
- ❑ функции могут вкладываться друг в друга;
- ❑ возвращаемое значение функции задается инструкцией `return`;
- ❑ функция может возвращать значение любого типа;
- ❑ функция может изменять значения переменных, которые были переданы ей в качестве значений параметров.

Все эти особенности функций мы сейчас и рассмотрим.

## 17.3. Объявление функции

Пользовательская функция в РНР объявляется так:

```
function имя([параметр1[=значение1], ..., параметрN[=значениеN]])
{
    операторы
}
```

Функцию можно объявить в любой части сценария, но до места первого ее использования. *Имя функции* должно соответствовать следующим требованиям:

- ❑ оно может содержать только английские символы, цифры и знак подчеркивания;
- ❑ в нем не должно быть пробелов и других пробельных символов;
- ❑ оно должно начинаться с буквы — т. е. `f1()`, но не `1f()`.

К именам параметров требования такие же, как и к именам переменных. Функция вообще может не содержать параметров.

При объявлении функции вы можете задать параметры по умолчанию, например:

```
function sum($a, $b=0)
{
    return $a + $b;
}
```

Вызвать функцию `sum()` можно так:

```
$num=7;
echo sum(5);           // результат 5
echo sum(5, 5);        // результат 10
echo sum($num);        // результат 7
echo sum($num, 5);     // результат 12
```

Другими словами, в качестве аргументов функции вы можете передавать как значения, так и переменные, — на результат функции это никак не повлияет. Парамет-

ры по умолчанию вы можете не указывать, тогда функция будет использовать заданные вами значения по умолчанию.

Начиная с PHP 7, больше нельзя определить два параметра с одинаковыми именами. Вам кажется такое определение параметров абсурдным? До 7-й версии оно было вполне допустимо, и программист мог бы определить функцию так:

```
public function foo($a, $b, $unused, $unused) {  
    // ...  
}
```

Однако в PHP 7/8 нужно использовать уникальные имена параметров:

```
public function foo($a, $b, $unused1, $unused2) {  
    // ...  
}
```

Говоря об аргументах функции, нельзя не упомянуть о необязательных и именованных аргументах. Начнем с первого. *Необязательным аргументом* является любой аргумент, для которого указано значение по умолчанию. Если он не передан, функция будет использовать заданное значение по умолчанию. Важно, чтобы необязательные параметры объявлялись после обязательных, иначе вы получите сообщение об ошибке:

```
// все параметры могут быть необязательными  
function f1 ($a = 1, $b = 2, $c = 3)  
// необязательные объявляются после обычных  
function f1 ($a, $b = 2, $c = 3)  
// так нельзя:  
function f1 ($a = 1, $b = 2, $c)
```

В PHP 8 появились так называемые *именованные* параметры. С помощью таких параметров аргументы функции можно передавать по имени, при этом порядок следования самих аргументов не имеет значения.

Именованные аргументы можно использовать для пропуска нескольких необязательных параметров. Начиная с PHP 8.0.0, объявление обязательных аргументов после необязательных аргументов считается устаревшим. Обычно это можно решить, отказавшись от значения по умолчанию, поскольку оно никогда не будет использоваться.

Синтаксис такой:

```
function название_функции(название_параметра1: значение1,  
название_параметра2: значение2,  
...)
```

Пример:

```
// Использование позиционных аргументов:  
array_fill(0, 100, 50);  
  
// Использование именованных аргументов:  
array_fill(start_index: 0, count: 100, value: 50);
```

## 17.4. Области видимости функции

Для начала уточним понятия локальных и глобальных переменных. *Глобальной* называется переменная, объявленная вне функции. Соответственно *локальной* называется переменная, объявленная в теле функции.

Предположим, у вас есть глобальная переменная `$i`, служащая в качестве счетчика цикла `for`, и есть локальная переменная с таким же именем и используемая для этих же целей, но внутри функции `f1()`:

```
function f1()
{
    for ($i=0; $i < 3; $i++) echo $i;
}

for ($i=0; $i < 3; $i++) f1();
```

Приведенный код будет прекрасно работать, поскольку функция `f1()` использует свою собственную переменную `$i`, которая не имеет никакого отношения к глобальной переменной с таким же именем.

Но если у вас есть необходимость обратиться к глобальной переменной из функции, то придется в теле функции использовать инструкцию `global`:

```
global <имя_переменной>;
```

Например:

```
global $i;
```

Нужно отметить, что в PHP 7 ключевое слово `global` может принимать только простые переменные. То есть вместо следующей конструкции:

```
global $$foo->bar;
```

нужно будет использовать вот такую:

```
global ${$foo->bar};
```

## 17.5. Вложенность функций

Вы можете объявить одну функцию внутри другой функции, например:

```
function f1()
{
    echo "Это функция f1 <br>\n";

    function f2()
    {
        echo "Это функция f2 <br>\n";
    }
}
```

Однако результат окажется совсем не таким, как вы ожидали. Можно предположить, что функция `f2()` является локальной функцией функции `f1()` и будет доступна только в пределах функции `f1()`. Но на самом деле PHP все равно, где объявлена функция, — все функции объявляются как глобальные и становятся доступны в любой части сценария. Например, за пределами функции `f1()` вы можете вызвать обе функции:

```
f1();  
f2();
```

Вывод будет таким:

```
Это функция f1 <br>  
Это функция f2 <br>
```

Поскольку никакой разницы, где объявлять функцию, нет, лучше не использовать вложенность функций, иначе вы ненароком можете создать непрямую рекурсию. А если вы не предусмотрите условие выхода из такой рекурсии, сценарий заикнется. Хорошо, что интерпретатор прервет его выполнение через 180 секунд!

Уточним, что такое *рекурсия*. Прямая рекурсия (или просто рекурсия) — это состояние, когда функция вызывает саму себя. Рассмотрим простейшую функцию, вычисляющую факториал числа `$n`:

```
function f($n)  
{  
    if ($n==0) return 1;  
    else return $n*f($n-1);  
}
```

Заметьте, что мы предусмотрели условие выхода: если `$n = 0`, то функция прекращает рекурсию. Поскольку значение `$n` уменьшается при каждом новом вызове функции, то рано или поздно `$n` станет равным 0.

А вот если бы мы забыли уменьшать `$n`, то наш сценарий заикнулся бы:

```
// не запускайте эту функцию!!!  
function f($n)  
{  
    if ($n==0) return 1;  
    else return $n*f($n);  
}
```

*Непрямая рекурсия* — это когда одна функция запускает вторую функцию, а она, в свою очередь, вызывает первую функцию:

```
function f1()  
{  
  
    echo "Это функция f1 <br>\n";
```

```
f2();

function f2()
{
    echo "Это функция f2 <br>\n";
    f1();
}
}
```

Если вызвать любую из этих функций, произойдет заикливание сценария, поскольку мы не предусмотрели условие выхода из не прямой рекурсии.

Рекурсия — мощный инструмент, но в то же время он довольно опасен в неопытных руках. Вообще любой рекурсивный алгоритм можно преобразовать в аналогичный нерекурсивный, т. е. без рекурсии можно спокойно жить. Учитывая всю коварность рекурсии, лучше от нее отказаться.

## 17.6. Переменное число аргументов

Далеко не всегда мы знаем, сколько параметров понадобится функции. PHP позволяет написать действительно универсальную функцию с переменным числом параметров. Вы можете передать такой функции один параметр, два параметра, сто параметров. Получить доступ ко всем параметрам функции позволяет функция `func_get_args()`. Вот пример ее использования:

```
function f()
{
    foreach(func_get_args() as $argn => $argv)
        echo "Параметр $argn Значение $argv <br>\n";
}
```

Вызвать функцию можно так:

```
f("аргумент", 5, 10);
```

Вывод функции будет таким:

```
Параметр 0 Значение аргумент
Параметр 1 Значение 5
Параметр 2 Значение 10
```

### **ВНИМАНИЕ!**

Как видите, мы передали функции параметры разных типов: одну строку и два числа. Поэтому перед использованием аргументов функции нужно установить их тип. Получить количество переданных параметров можно с помощью функции `func_num_args()`.

В PHP 7/8 изменено поведение функций `func_get_arg()` и `func_get_args()`. Теперь эти функции возвращают не исходное значение, переданное в функцию, а текущее значение (которое может быть модифицировано). Например:

```
function foo($x) {
    $x++;
    var_dump(func_get_arg(0));
}
foo(1);
```

В PHP 5 такой код вывел бы 1, поскольку функция `foo()` вызвана с параметром 1. В PHP 7/8 код выведет значение 2, поскольку переменная-параметр была увеличена на единицу. Чтобы вернуть старое поведение, нужно или изменить сам код, увеличивая `x` после вызова `func_get_arg()`:

```
function foo($x) {
    var_dump(func_get_arg(0));
    $x++;
}
```

или просто избегать изменения переданных в функцию параметров:

```
function foo($x) {
    $newX = $x + 1;
    var_dump(func_get_arg(0));
}
```

Аналогично в `backtrace` теперь будет отображаться текущее значение, а не то значение, с которым была вызвана функция. То есть в результате следующего кода:

```
function foo($x) {
    $x = 42;
    throw new Exception;
}
foo("string");
```

в стеке вызовов окажется следующий результат:

```
Stack trace:
#0 file.php(4): foo(42)
#1 {main}
```

Хотя ранее стек вызовов был бы таким:

```
Stack trace:
#0 file.php(4): foo('string')
#1 {main}
```

## 17.7. Передача массивов в качестве параметров

Вы можете в качестве аргумента функции передать ей массив. Вот, например, функция, определяющая индекс минимального элемента списка:

```
function array_min($Arr)
{
    $min = $Arr[0];
    $min_ind = 0;

    foreach($Arr as $k => $v)
        if ($v < $min) { $min_ind = $k; }

    return $min_ind-1;
}
```

Рассмотрим еще один пример, а именно функцию `array_info()` (листинг 17.1). Функция будет работать не с обыкновенной скалярной переменной, а с целым массивом. Получив массив, она вычисляет номера минимального и максимального элементов, а затем рассчитывает среднее арифметическое элементов массива. В итоге функция возвращает список, состоящий из трех элементов:

- номера максимального элемента;
- номера минимального элемента;
- среднего арифметического всех элементов.

**Листинг 17.1. Функция `array_info()`**

```
<?php
function array_info($Arr)
{
    $min = $max = $Arr[0];    // Min = Max = первый элемент
    $mn_ind = $mx_ind = 0;    // Индексы минимального и максимального элементов
    $avg = 0;                 // Среднее арифметическое

    foreach($Arr as $k=>$v)
    {
        if ($max < $v) { $max=$v; $mx_ind=$k; }; // Вычисление $max
        if ($min > $v) { $min=$v; $mn_ind=$k; }; // Вычисление $min
        $avg = $avg + $v;
    }

    $avg = $avg / count ($Arr);    // Сумма / количество элементов

    $Res[]=$mx_ind;
    $Res[]=$mn_ind;
    $Res[]=$avg;
}
```

```
return $Res;                                //Передача результата
}

for($i=1; $i<11; $i++) $Arr[]=$i;           //Заполнение массива Arr[]

foreach(array_info($Arr) as $v) echo "$v ";

?>
```

Функция `array_info()` принимает список, обрабатывает его и возвращает также список. Обратите внимание на обработку списка в основной программе. Мы могли бы присвоить возвращаемое функцией значение другому массиву, а потому работать с этим массивом:

```
$Result = array_param($Arr);

// выводим значения элементов массива $Result
foreach($Result as $r) echo "$r ";
```

Круглые скобки вокруг переменных или вызовов функции больше не имеют влияния на поведение. Например, следующий код, где результат вызова функции передан функции ссылкой:

```
function getArray() { return [1, 2, 3]; }

$last = array_pop(getArray());
$last = array_pop((getArray()));
```

теперь породит ошибку стандартизации, независимо от того, используются ли круглые скобки. Ранее во втором случае не было никакого уведомления.

Как видите, работать с параметрами-массивами так же просто, как и с обычными параметрами.

## 17.8. Передача аргументов по ссылке

По умолчанию аргументы передаются в функцию по значению, т. е. если вы измените значение аргумента в функции, то в основной программе значение останется прежним. Если же нужно, чтобы функция изменяла переданные в нее значения и эти значения передавались бы в основную программу, тогда нужно использовать передачу параметров по ссылке. Для этого используйте `&` перед именем аргумента:

```
function fl (&$a)
```

В современном мире передача параметров по ссылке считается дурным тоном, особенно когда объектно-ориентированный стиль практически вытеснил процедурный. В *главе 24* мы поговорим о классах, объектах, свойствах и методах — вы узнаете, как корректно обмениваться значениями с основной программой посредством свойств объекта, а не использованием `&` в описании функции.

## 17.9. Генераторы

*Генератор* — это функция, возвращающая не одно значение, а последовательность значений. Обратите внимание: не массив значений, а именно *последовательность значений*. При каждом новом вызове функции-генератора возвращается следующее значение из последовательности.

Напишем простейшую функцию-генератор:

```
function squares($start, $end, $step = 1) {
    for ($i = $start; $i <= $end; $i += $step) {
        yield $i*$i;
    }
}

foreach (squares(1, 10) as $n) {
    echo $n . " ";
}
```

Генератор `squares` генерирует последовательность квадратов чисел от `$start` до `$end` с шагом `$step`. Каждое значение возвращается не инструкцией `return`, а инструкцией `yield`. Обратите внимание, что генератор не возвращает всю последовательность сразу. При каждом вызове генератора возвращается следующее значение. Наш генератор выведет:

```
1 4 9 16 25 36 49 64 81 100
```

В PHP 7/8 произошли также существенные изменения и в генераторах.

- ❑ Во-первых, инструкция `yield` более не требует скобок при использовании в контексте выражения. Она теперь представляет собой правоассоциативный оператор с приоритетом между `print` и `=>`. В некоторых случаях это нововведение может повлиять на поведение сценария, например:

```
echo yield -1;
// Ранее интерпретировалось как
echo (yield) - 1;
// А теперь интерпретируется как
echo yield (-1);

yield $foo or die;
// Ранее интерпретировалось так:
yield ($foo or die);
// Сейчас интерпретируется так:
(yield $foo) or die;
```

Такие спорные случаи всегда могут быть разрешены путем добавления дополнительных круглых скобок.

- Во-вторых, изменения в генераторах также коснулись и оператора `return`. Теперь в PHP есть так называемые *выражения* генератора `return`<sup>1</sup>, позволяющие возвращать значение после успешного завершения работы генератора.

До PHP7, если вы пытались что-нибудь вернуть в генераторе, это приводило к ошибке. Однако теперь вы можете вызвать `$generator->getReturn()`, чтобы получить возвращаемое значение. При этом:

- если генератор еще не завершился или выбросил «непойманное» исключение, вызов `$generator->getReturn()` сгенерирует исключение;
- если же генератор завершен, но не объявлен `return`, то метод вернет `NULL`.

Рассмотрим небольшой пример:

```
function gen() {
    yield "Hello";
    yield " ";
    yield "World!";

    return "Goodbye World!";
}

$gen = gen();

foreach ($gen as $value) {
    echo $value;
}

// Выведет "Hello" на итерации 1,
// " " на итерации 2 и "World!" на итерации 3

echo $gen->getReturn(); // будет присвоено значение Goodbye World!
```

- Но и это еще не все изменения — в PHP 7/8 появилась революционная возможность, которая называется *делегированием генератора*. Она позволяет вернуть другую итерабельную структуру — например, массив, итератор или другой генератор.

Синтаксис делегирования генератора такой: `yield from <expression>`. Посмотрим это на примере:

```
function hello() {
    yield "Hello";
    yield " ";
    yield "World!";

    yield from goodbye();
}
```

<sup>1</sup> См.: <https://wiki.php.net/rfc/generator-return-expressions>.

```
function goodbye() {  
    yield "Goodbye";  
    yield " ";  
    yield "World";           // без !  
}  
  
$gen = hello();  
foreach ($gen as $value) {  
    echo $value;  
}
```

Вот что будет выведено при каждой итерации:

```
"Hello"  
" "  
"World!"  
"Goodbye"  
" "  
"World"
```

Стоит упомянуть еще один нюанс: поскольку субструктуры привносят свои собственные ключи, вполне возможно, что один и тот же ключ будет возвращен за несколько итераций. Недопущение подобного — ответственность программиста.

## 17.10. Полезные примеры

Рассмотрим несколько полезных примеров собственных функций.

### 17.10.1. Получение реального IP-адреса клиента

Не всегда в `$_SERVER['REMOTE_ADDR']` содержится реальный IP-адрес пользователя. Если пользователь находится за прокси-сервером, его реальный IP-адрес нужно искать в `$_SERVER['HTTP_CLIENT_IP']` или в `$_SERVER['HTTP_X_FORWARDED_FOR']`. Напишем функцию, которая будет вычислять реальный IP-адрес пользователя, даже если он находится за прокси-сервером (листинг 17.2).

#### Листинг 17.2. Функция `getRealIpAddr()`

```
function getRealIpAddr()  
{  
    if (!empty($_SERVER['HTTP_CLIENT_IP']))  
    {  
        $ip=$_SERVER['HTTP_CLIENT_IP'];  
    }  
    elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR']))  
    {  
        $ip=$_SERVER['HTTP_X_FORWARDED_FOR'];  
    }  
}
```

```
else
{
    $ip=$_SERVER['REMOTE_ADDR'];
}
return $ip;
}
```

Использовать функцию нужно так:

```
$ip = getReadIpAddr();
```

## 17.10.2. Генерирование сложного пароля

Благодаря использованию следующей простой функции, у вас больше никогда не будет простых паролей. Эта функция генерирует сложный пароль длины \$l (листинг 17.3).

### Листинг 17.3. Функция `generate_rand`

```
function generate_rand($l = 10){
    $c= "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    srand((double)microtime()*1000000);
    $rand = '';
    for($i=0; $i<$l; $i++) {
        $rand.= $c[rand()%strlen($c)];
    }
    return $rand;
}
```

## 17.10.3. Рекурсивное удаление каталога

PHP может удалить только пустой каталог. Следовательно, перед удалением каталога из него нужно удалить все файлы и все подкаталоги. С файлами все достаточно просто, а вот с подкаталогами — нет. Ведь нужно «зайти» в каждый каталог, удалить из него все файлы, затем проверить наличие в каталоге подкаталогов, после чего «зайти» в каждый подкаталог, удалить все из него, а потом удалить и сам подкаталог.

Функция `destroyDir()` демонстрирует работу рекурсии — ситуации, когда функция вызывает саму себя. Работает функция так: она получает список элементов (файлов и каталогов) удаляемого каталога, после чего «проходится» по этому списку. Если элемент представляет собой ссылку на текущий (".") или родительский ("..") каталог, то происходит переход к следующему элементу. Если элемент — каталог, функция вызывает сама себя, но уже для этого элемента (для подкаталога). Если элемент — файл, функция удаляет его с помощью функции `unlink()` (листинг 17.4).

**Листинг 17.4. Функция `destroyDir()`**

```
function destroyDir($dir, $virtual = false)
{
    $ds = '/'; // или '\' для Windows
    $dir = $virtual ? realpath($dir) : $dir;
    $dir = substr($dir, -1) == $ds ? substr($dir, 0, -1) : $dir;
    if (is_dir($dir) && $handle = opendir($dir))
    {
        while ($file = readdir($handle))
        {
            if ($file == '.' || $file == '..')
            {
                continue;
            }
            elseif (is_dir($dir.$ds.$file))
            {
                destroyDir($dir.$ds.$file);
            }
            else
            {
                unlink($dir.$ds.$file);
            }
        }
        closedir($handle);
        rmdir($dir);
        return true;
    }
    else
    {
        return false;
    }
}
```

Использование функции:

```
destroyDir("temp");
```

## 17.10.4. Отправка файла в браузер

Наверняка вы видели сценарии, которые инициируют загрузку того или иного файла. Как правило, такие сценарии используются на разных файлообменниках. Чтобы начать загрузку файла (т. е. заставить сервер передать клиенту файл), нужно отправить несколько HTTP-заголовков. Мы напишем простую функцию, чтобы каждый раз при «отдаче» файла вам не приходилось отправлять эти заголовки вручную (листинг 17.5).

**Листинг 17.5. Функция `force_download()`**

```
function force_download($file)
{
    if ((isset($file)&&(file_exists($file))) {
        header("Content-length: ".filesize($file));
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename="' . $file . '"');
        readfile("$file");
    } else {
        echo "No file selected";
    }
}
```

Все, что вам нужно, — это передать имя существующего файла:

```
force_download("45.php");
```

Дальнейшее развитие событий зависит от браузера и его настроек. По умолчанию Edge отображает диалоговое окно, предлагающее что-то сделать с загружаемым файлом: открыть, сохранить или отказаться от загрузки. Браузер Opera сразу загружает файл в каталог загрузок, а Chrome — отображает окно сохранения файла, предлагающее сохранить файл.

### 17.10.5. Сжатие файла «на лету»

Возможности PHP кажутся безграничными, особенно с учетом всевозможных дополнительных библиотек. PHP может даже работать с ZIP-архивами. Следующая функция создает архив по массиву с именами файлов (листинг 17.6).

**Листинг 17.6. Функция `create_zip()`**

```
function create_zip($files = array(), $destination = '', $overwrite = false) {
    // если архив существует и overwrite=false, возвращает false
    if(file_exists($destination) && !$overwrite) { return false; }
    $valid_files = array();
    // если файлы были переданы
    if(is_array($files)) {
        // итерируем по каждому файлу
        foreach($files as $file) {
            // перед добавлением в $valid_files нужно убедиться, что файл
            // существует
            if(file_exists($file)) {
                $valid_files[] = $file;
            }
        }
    }
}
```

```
// если у нас есть файлы для упаковки
if(count($valid_files)) {
    // создаем архив
    $zip = new ZipArchive();
    if($zip->open($destination,$overwrite ? ZIPARCHIVE::OVERWRITE :
ZIPARCHIVE::CREATE) !== true) {
        return false;
    }
    // добавляем файлы
    foreach($valid_files as $file) {
        $zip->addFile($file,$file);
    }
    // раскомментируйте для отладки
    //echo 'The zip archive contains ', $zip->numFiles, ' files with a status
        of ', $zip->status;

    // закрываем архив
    $zip->close();

    // нужно убедиться, что созданный файл существует (мало ли чего -
    // вдруг произошла ошибка в библиотеке и архив не был создан)
    return file_exists($destination);
}
else
{
    return false;
}
}
```

Использовать функцию нужно так:

```
$files=array('file1.jpg', 'file2.jpg', 'file3.png');
create_zip($files, 'zipfile.zip', true);
```



# **I. РАЗДЕЛ 5**

## **База данных MySQL**

<b>Глава 18.</b>	Установка MySQL на VDS
<b>Глава 19.</b>	Основы SQL
<b>Глава 20.</b>	Функции для работы с MySQL
<b>Глава 21.</b>	Работа с базой данных в Laravel



## ГЛАВА 18

# Установка MySQL на VDS

### 18.1. Несколько вводных слов

Виртуальные выделенные серверы (VDS) прочно вошли в мир IT-разработки, и сегодня представить себе какой-либо серьезный проект без VDS практически невозможно. Почему? Потому что VDS — это удобно (для разработчика) и дешево (для владельца проекта). Содержание (обеспечение непрерывной работы) физического сервера — та еще задача, и она не может быть качественно реализована за пределами дата-центра. Да и покупка «физики» вместе с поддержанием режима 24/7 выльется в копеечку. Дешевле арендовать VDS в публичном дата-центре и сосредоточиться на развитии проекта, а не на создании своего мини дата-центра.

Удобство для разработчика в случае с VDS тоже трудно переоценить. По сути, VDS — это обычная виртуальная машина, с которой можно сделать все, что захочется. Например, сделать моментальный снимок (снапшот) текущего состояния перед внесением важных изменений и затем откатиться на него за считанные секунды. Или же сделать клон (копию) сервера и использовать его как тестовый вариант, а когда все будет готово — подменить им основной вариант. То есть ввод новой версии ПО будет занимать не так уж и много времени.

В этой главе мы поговорим как раз о том, как установить СУБД на VDS под управлением Ubuntu. В процессе установки современных версий MySQL встречаются несколько подводных камней, с которыми мы и разберемся далее.

### 18.2. Установка сервиса MySQL

Для установки сервиса MySQL введите команды:

```
sudo apt install mysql-server  
sudo systemctl start mysql.service
```

Первая команда — устанавливает сервер MySQL, а вторая — запускает сервис (службу) `mysql.service`. Однако использовать сервер сразу после установки нельзя. По умолчанию у пользователя пароль `root` не установлен, а доступ как `root` разрешен отовсюду. Долго ваша база данных с такой конфигурацией не проработает.

## 18.3. Настройка MySQL

Если вы подумали, что в этом разделе мы поговорим о скрипте `mysql_secure_installation`, который нужно запустить, чтобы навести порядок с безопасностью (установить пароль `root`, удалить БД `test` и т. д.), вы угадали. Но немного не в том контексте. Дело в том, что в современных версиях MySQL в этом скрипте есть ошибка. Впервые она была замечена в Ubuntu 20.04, но в версии 22.04 (последняя на текущий момент) она все еще не исправлена.

Скрипт запускается, но при попытке установить пароль `root` вы видите сообщение:

```
... Failed! Error: SET PASSWORD has no significance for user 'root'@'localhost'
as the authentication method used doesn't store authentication data in the
MySQL server. Please consider using ALTER USER instead if you want to change
authentication parameters.
```

Чтобы исправить ситуацию, выполните команду:

```
sudo mysql
```

Далее введите SQL-запрос:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'ваш_пароль';
```

После этого осуществите выход (команда `exit`) и снова запустите `mysql_secure_installation` — на этот раз скрипт будет выполнен без ошибок. Скрипт задаст ряд вопросов, на которые нужно ответить `<Y>` или `<y>`, если вы хотите ответить утвердительно, или же нажать любую клавишу, если вы собираетесь отказаться от предлагаемой возможности. Итак, вот эти вопросы:

- ❑ **Connecting to MySQL using a blank password** — можно ли подключаться к базе данных с пустым паролем. Нужно нажать любую клавишу, чтобы запретить подключение с пустым паролем;
- ❑ **VALIDATE PASSWORD COMPONENT...** — нажмите `<Y>`, если вы хотите включить компонент проверки паролей пользователей MySQL. Если вы ответите `<Y>`, далее нужно будет выбрать уровень сложности пароля:
  - **LOW (0)** — пароли с длиной  $\geq 8$ ;
  - **MEDIUM (1)** — пароли с длиной  $\geq 8$ , содержащие символы в разном регистре, цифры, специальные символы;
  - **STRONG (2)** — пароли с длиной  $\geq 8$ , содержащие символы в разном регистре, цифры, специальные символы и не содержащие словарных слов. Самый безопасный уровень;
- ❑ **Please set the password for root here** — нужно ввести пароль пользователя `root` базы данных (не путать с системным пользователем) и его подтверждение. Пароль должен состоять из букв разного регистра и цифр. Минимальная длина — 8 символов. Вы можете подтвердить пароль, установленный ранее с помощью команды `ALTER USER`. Если же вы введете другой пароль, то именно он будет использоваться в качестве пароля `root`;

- ☐ **Remove anonymous users** — чтобы удалить анонимных пользователей, нажмите <Y>;
- ☐ **Dissallow root login remotely** — опция отключает вход пользователя root удаленно. Нажмите <Y>;
- ☐ **Remove test database and access to it** — опция удаляет тестовую базу данных и доступы к ней. Нажмите <Y>;
- ☐ **Reload privilege tables now** — нажмите <Y>, чтобы перезагрузить таблицы привилегий прямо сейчас.

## 18.4. Создание MySQL-пользователя

Работать под root — плохая идея, хотя бы даже потому, что пароль root в конфигурационных файлах веб-приложений будет храниться в открытом виде. Поэтому нужно создать базу данных для веб-приложения (желательно для каждого приложения создавать свою БД) и пользователя, от имени которого веб-приложение будет работать с базой данных. Создавать ли отдельного пользователя для каждой базы данных — решайте сами. Из соображений безопасности и распределения нагрузки на сервер рекомендуется для каждого проекта (сайта) создавать отдельную БД и отдельного пользователя.

Введите команду:

```
mysql -u root -p
```

Эта команда открывает консоль MySQL от имени root (`-u root`) с использованием пароля (`-p`). Пароль root (имеется в виду MySQL-пользователь root, а не системный) вам нужно ввести (вы его задавали в *разд. 18.3*).

Настало время вводить SQL-запросы:

```
CREATE DATABASE mydb;  
CREATE USER myuser@localhost IDENTIFIED BY 'S6d#dbh2023!';  
GRANT ALL ON mydb.* TO myuser@localhost;  
ALTER USER myuser@localhost WITH MAX_QUERIES_PER_HOUR 1000;  
FLUSH PRIVILEGES;  
exit
```

Первый MySQL-запрос создает базу данных mydb. Название, разумеется, вы можете изменить. Второй запрос создает пользователя myuser@localhost и задает его пароль. Пароль специально здесь приведен так, чтобы вы видели, что он должен содержать буквы в разных регистрах и цифры. Интересно, что в MySQL 8 вы можете задать слабый пароль при создании пользователя, но пользователь не сможет подключиться к серверу с использованием слабого пароля.

Третий запрос — это предоставление всех полномочий пользователю myuser в базе данных mydb. Обратите внимание: синтаксис этой команды изменился в MySQL 8 — ранее команда предоставления всех полномочий выглядела иначе. Также заметьте, что мы предоставляем все полномочия только к заданной базе данных, а не ко всем базам.

Следующий запрос ограничивает количество запросов в час. За час наш пользователь сможет сделать не более 1000 запросов. В реальной жизни этого маловато, но принцип вам понятен — с помощью такого запроса вы можете регулировать этот параметр. Можно пока его вообще не вводить.

Следующая команда выполняет обновление полномочий, чтобы произведенные изменения вступили в силу, а последняя — производит выход из MySQL-клиента.

После этого введите команду:

```
mysql -u myuser -p
```

Если вы все сделали правильно, вы должны увидеть приглашение MySQL, но уже для пользователя `myuser`.

Созданные аутентификационные данные (имя пользователя, пароль, база данных) нужно указать в настройках вашего веб-приложения.

## 18.5. Запуск и останов сервера

Команды запуска, останова и перезапуска сервера MySQL выглядят так:

```
sudo systemctl start mysql.service  
sudo systemctl stop mysql.service  
sudo systemctl restart mysql.service  
sudo systemctl status mysql.service
```

Последняя команда выводит состояние MySQL-сервера в текущий момент.



## ГЛАВА 19

# Основы SQL

### 19.1. Немного истории

В далеком 1970-м компания IBM разработала экспериментальную систему управления базами данных System R. В ее основе лежал язык SEQUEL (Structured English Query Language, структурированный английский язык запросов). Чуть позже этот язык по юридическим и политическим соображениям был переименован в просто SQL (Structured Query Language).

Разработкой SQL занимались Чэмбэрлин (Chamberlin) и Рэй Бойс (Ray Boyce). Кроме них над SQL вели работу Пэт Селинджер (Pat Selinger), которая разрабатывала оптимизатор, и Рэймонд Лори (Raymond Lorie), создававший компилятор запросов. Их главной целью было создание языка запросов к базе данных, с которым мог бы разобраться любой пользователь, не имеющий никаких навыков программирования.

В 1986 году Американский национальный институт стандартизации (ANSI, American National Standards Institute) принял первый стандарт языка SQL. Принятие стандарта должно было способствовать дальнейшему развитию языка запросов. В 1987 году стандарт был уточнен (появился так называемый SQL level 1). Еще одно изменение в стандарте произошло в 1989 году (SQL level 2).

Развитие и совершенствование систем управления базами данных послужило причиной принятия в 1992 году следующего стандарта (он назывался ANSI SQL-92 или просто SQL-2). Последующие стандарты были приняты в 1999 (SQL-99) и 2003 годах (SQL-3).

### 19.2. Преимущества SQL

Как уже отмечалось, основное преимущество SQL — простота его использования. Предположим, что SQL нет, но нам необходима база данных. Что делать? Сначала нужно разработать формат двоичного файла (с двоичным файлом операции поиска выполняются быстрее, чем с текстовым): продумать заголовок файла, разделители полей и индексы. На все это уйдет много времени. Еще больше времени займет

разработка набора функций, которые будут работать с нашим двоичным форматом. Спустя какое-то время этот набор функций мы создадим. Затем напишем приложение с их использованием. Допустим, нам потребуется просмотреть весь файл данных и выбрать из него записи с определенным значением поля — например, чтобы поле «Количество» было больше нуля. Алгоритм обработки предполагается следующим — нужно в цикле пересмотреть все записи и вывести только те, которые не содержат ноль в поле «Количество». Не очень удобно, но все же выполнимо. Но вот незадача — а что делать, если возникнет необходимость радикально изменить формат файла? Ведь при разработке мы могли допустить ошибку, точнее, недоработку, которая «всплыла» бы несколько месяцев спустя. Тогда нам пришлось бы заново переделывать весь наш набор функций. Причем делать все так, чтобы это не затронуло работу уже существующих приложений, иначе нужно будет переписывать и их.

В случае с SQL все гораздо проще. Вам не придется изобретать велосипед заново: забудьте о формате данных и наборе функций — для вас этих проблем просто не существует. Все, что вам необходимо знать, — это то, что есть база данных, которая выполнит любой ваш приказ, сформулированный по правилам языка SQL. Программировать вам тоже ничего не понадобится — зачем циклы, если можно попросить SQL: «Просмотри все записи и выбери те, у которых поле “Количество” равно нулю». Даже если возникнет необходимость перейти на другую систему управления базами данных, SQL-запросы останутся прежними.

Таким образом, преимущества SQL следующие:

- независимость от системы управления базами данных — благодаря тому, что приняты стандарты SQL, его запросы будут одинаково выполняться что в Oracle, что в MySQL. Конечно, есть определенные отличия синтаксиса некоторых СУБД, но они, как правило, незначительны;
- полноценность SQL как языка управления данными — с помощью SQL можно не только производить выборку данных, но и полноценно управлять ими: добавлять новые данные, изменять и удалять уже имеющиеся, создавать базы данных. Выходит, что кроме знания SQL вам для работы с СУБД не понадобятся никакие другие дополнительные сведения.

## 19.3. Как выглядят запросы?

Поскольку SQL является структурированным языком запросов, все запросы в нем имеют четкую структуру, которой рекомендуется придерживаться, если вы хотите, чтобы ваш код был удобен для чтения. Вот пример запроса `SELECT` (выборка данных):

```
SELECT Q_NO, NAME, PHONE  
FROM CUSTOMERS  
WHERE CITY = 'KIEV';
```

Как видите, не зря SQL когда-то назывался `SEQUEL` — уж очень он похож на обычный английский язык. Понять суть этого запроса можно, даже не имея пред-

ставления об SQL. Очевидно, что здесь запрос выбирает информацию обо всех клиентах, проживающих в Киеве.

В конце каждого запроса обязательна точка с запятой. Но когда вы будете передавать запросы в функцию PHP, разрешается точку с запятой не ставить и записывать запросы в одну строку. Поступайте так, как вам будет удобнее:

```
// можно так
$q = 'select * from clients';
```

```
// а можно так
$q = 'select *
from clients';
```

```
$r = mysql_query($q);
```

## 19.4. Что такое база данных?

Что такое система управления базами данных (СУБД), понятно из самого названия. А вот что такое *база данных*? База данных — это набор связанных (не всегда) между собой таблиц. С технической точки зрения базой данных может быть обычный каталог на диске, в котором размещены файлы таблиц, или же один большой двоичный файл, содержащий всю информацию, — тут все зависит от реализации формата базы данных. Например, в MySQL базой данных является каталог с файлами таблиц, а вот в InterBase — это один большой файл.

Теперь поговорим о связи таблиц. Отличительным признаком реляционной (от *англ.* relations — отношения) базы данных является как раз «набор взаимосвязанных таблиц». Предположим, что у нас есть база данных интернет-магазина, содержащая информацию о товарах, клиентах и заказах. В первой таблице содержится информация о товарах: номер товара, описание и цена. Во второй — информация о клиентах: номер клиента, фамилия, электронный и почтовый адреса. А вот третья таблица связывает две предыдущих — в ней находится информация о заказах, т. е. номер клиента, номер товара и количество (это минимальный набор полей).

В теории так оно и есть — все таблицы базы данных связаны между собой. Но на практике наблюдается совершенно противоположная ситуация. Количество баз данных часто ограничивается. Нет, не самой СУБД, а хостинг-провайдерами. Например, для минимальных хостинг-планов предоставляется всего одна база данных. А ведь хочется установить и форум, и интернет-магазин, и еще многое другое. Вот и получается, что в одной базе данных окажется несколько наборов таблиц, связанных только внутри себя, но не с другими наборами. Например, мы создали в одной и той же базе данных два набора таблиц: один для форума, а второй — для интернет-магазина. Таблицы форума будут связаны между собой, но они не будут иметь никакого отношения к таблицам интернет-магазина. Поэтому базу данных лучше рассматривать просто как набор таблиц без учета связей между ними.

Теперь поговорим о более «мелких» элементах базы данных — таблицах. Таблица — это набор записей одной структуры. Структура записей, т. е. набор полей,

задается при создании таблицы. Запись — это набор полей, содержащих связанную информацию. Например, `C_NO`, `NAME`, `CITY`, `EMAIL` — данный набор полей содержит информацию о человеке: его персональный номер (`C_NO`), имя (`NAME`), город (`CITY`), электронный адрес (`EMAIL`). Связь заключается в том, что вся информация в одной записи относится к одному и тому же объекту. Например, запись

```
1 Иванов Москва ivanov@i.ru
```

говорит о том, что персональный номер Иванова — 1, Иванов проживает в Москве и его электронный адрес — **ivanov@i.ru**.

Поле — это базовый элемент базы данных, из полей формируются записи. Поле содержит информацию ранее заданного типа (тип поля задается при создании таблицы).

Кроме полей, записей и таблиц в базе данных могут быть индексы (ключи), они не менее важны, чем поля и таблицы. Индексы используются для быстрого поиска нужной записи и для управления порядком отображения записей. Существуют первичный и вторичный индексы.

□ *Первичный индекс* (primary key, primary index) управляет порядком отображения записей в таблице. Первичный индекс строится по одному полю, значение которого должно быть уникальным. В приведенном примере на звание первичного индекса претендует поле `C_NO`, содержащее уникальный номер человека в системе. По полю `NAME` первичный индекс строить нельзя, поскольку Ивановых на просторах нашей страны очень много, и в таблицу могут попасть несколько Ивановых, — как тогда понять, кто есть кто? А вот номер `C_NO` будет уникальным в нашей системе (об этом может даже позаботиться сама система), поэтому `C_NO` — идеальное поле для первичного индекса. Первичный индекс обычно задается при создании таблицы.

□ *Вторичный индекс* (secondary index) не обязательно должен быть уникальным, к тому же он может строиться по нескольким полям сразу. Вторичные индексы, как правило, служат для связывания таблиц. Вторичный ключ еще иногда называют внешним ключом.

Вот теперь можно приступить к рассмотрению основ SQL. Сразу отмечу, что приведенного далее материала вполне хватит для разработки даже сложных проектов, но он никак не претендует на полноту описания SQL.

## 19.5. Создание таблиц

Для создания таблицы предназначен SQL-оператор `CREATE`:

```
CREATE TABLE Имя_таблицы
(
Имя_поля1          Тип          Модификатор,
. . .
Имя_поляN          Тип          Модификатор,
[Первичный        ключ, ]
[Внешний          ключ]
)
```

Оператор `CREATE` в SQL используется для создания не только таблиц, но и других объектов базы данных. В этой книге мы будем говорить только о создании таблиц, но для интересующихся в табл. 19.1 описаны альтернативные операторы `CREATE`, позволяющие создавать другие объекты базы данных.

Таблица 19.1. Синтаксис оператора `CREATE`

Синтаксис оператора <code>CREATE</code>	Объект
<code>CREATE DATABASE &lt;имя&gt;</code>	Создает базу данных. Создать базу данных не всегда возможно. Во-первых, у вас должны быть права на ее создание. Во-вторых, многие хостинг-провайдеры устанавливают лимит на количество баз данных
<code>CREATE DEFAULT &lt;имя&gt;</code> <code>AS &lt;выражение&gt;</code>	Создает константу по умолчанию
<code>CREATE FUNCTION &lt;имя&gt;</code> <code>RETURNS</code> <code>&lt;значение&gt; AS &lt;операторы_SQL&gt;</code>	Создает пользовательскую функцию
<code>CREATE INDEX &lt;имя&gt;</code> <code>ON &lt;таблица_или_представление&gt;</code> <code>&lt;индексируемые_поля&gt;</code>	Создает индекс (в таблице или представлении)
<code>CREATE PROCEDURE &lt;имя&gt;</code> <code>AS &lt;операторы_SQL&gt;</code>	Создает пользовательскую процедуру
<code>CREATE RULE &lt;имя&gt;</code> <code>AS &lt;выражение&gt;</code>	Создает правило
<code>CREATE SCHEMA AUTHORIZATION</code> <code>&lt;владелец&gt; &lt;объекты&gt;</code>	Таблицы, представления и разрешения будут созданы как один объект
<code>CREATE TABLE &lt;имя&gt; (определение таблицы)</code>	Создает таблицу
<code>CREATE VIEW &lt;имя&gt;</code> <code>AS &lt;оператор_выборки&gt;</code>	Создает представление

Вернемся к созданию таблицы. Для определения поля нужно указать его имя, тип и модификатор поля. Допустимые типы данных приведены в табл. 19.2.

Таблица 19.2. Часто используемые типы данных

Тип	Описание
TINYINT	Целый тип, может принимать значения из диапазона -128...+127
SMALLINT	Целый тип, может принимать значения из диапазона -32 768...+32 767
MEDIUMINT	Целый тип, может принимать значения из диапазона -8 388 608...+8 388 607

Таблица 19.2 (окончание)

Тип	Описание
INT	Целый тип, может принимать значения из диапазона –2 147 483 648...+2 147 483 647
BIGINT	Целый тип, может принимать значения из диапазона –9 223 372 036 854 775 808...+9 223 372 036 854 775 807
FLOAT (<Д>, <З>) [UNSIGNED]	Вещественный тип, малая точность. Для каждого вещественного типа можно указать его длину (Д), т. е. количество знаков-мест, в которых будет размещено число, а также количество знаков (З) после десятичной запятой, которое будет учитываться. Модификатор UNSIGNED говорит о том, что знак числа учитываться не будет
DOUBLE (Д, З) [UNSIGNED]	Вещественный тип, наибольшая точность
NUMERIC (Д, З) [UNSIGNED]	Дробное число, хранящееся в виде строки
DECIMAL (Д, З) [UNSIGNED]	Псевдоним для NUMERIC
REAL (Д, З) [UNSIGNED]	Псевдоним для DOUBLE
VARCHAR (N)	Строка, N — число символов (не более 255)
TINYTEXT	Небольшой текст, максимум 255 символов
TEXT	Текст (до 64 Кбайт)
MEDIUMTEXT	Текст, максимум 16 777 215 символов
LONGTEXT	Текст, максимум 4 294 967 295 символов
BLOB	Используется для хранения бинарных данных, максимальный размер которых — 64 Кбайт. Бинарные данные не перекодируются "на лету", если включена перекодировка символов
MEDIUMBLOB	Используется для хранения бинарных данных, максимальный размер — 16 777 215 символов
LONGBLOB	Используется для хранения бинарных данных, максимальный размер — 4 294 967 295 символов
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате timestamp, выводится в виде ГГГГММДДЧЧММСС. Формат не очень удобный, поэтому лучше использовать DATETIME
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС

После типа нужно указать модификатор поля (хотя это необязательно):

- ☐ NOT NULL — поле не может содержать пустое значение, т. е. при добавлении записи нужно обязательно указать значение этого поля;

- ❑ `PRIMARY KEY` — поле будет первичным ключом, хотя индексы лучше определять по-другому (об этом поговорим чуть позже);
- ❑ `AUTO_INCREMENT` — при добавлении новой записи значение этого поля будет автоматически увеличено на единицу (данный модификатор допустим только для целых типов);
- ❑ `DEFAULT` — задает значение по умолчанию, если значение поля при добавлении записи не задано, например `DEFAULT 0`.

Рассмотрим пример создания таблицы (листинг 19.1).

#### Листинг 19.1. Пример создания таблицы

```
CREATE TABLE friends
(
  id            int            auto_increment,
  name         varchar(50)    not null,
  email        varchar(30)    not null,
  comment text,
  primarykey (id)
);
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *19-1.sql*), представленный в каталоге *Glava\_19* сопровождающего книгу электронного архива (см. приложение 4).

Итак, наша таблица `friends` будет содержать четыре поля:

- ❑ `id` — может принимать целые значения (это будет номер записи), значения поля увеличиваются автоматически (`auto_increment`). Это сделано для нашего удобства — при добавлении новой записи нам не нужно будет помнить номер последней, т. к. его MySQL укажет за нас;
- ❑ `name` — текстовое поле для имени вашего друга, максимальная длина поля 50 символов;
- ❑ `email` — тоже текстовое поле, но с длиной строки 30 символов;
- ❑ `comment` — поле типа `TEXT`, может содержать небольшой текст объемом до 64 Кбайт.

Еще раз поговорим о поле `id`. Это поле объявлено как поле-счетчик (`AUTO_INCREMENT`), оно же будет первичным индексом. Если индексом должно быть не целочисленное поле или же не задан модификатор `AUTO_INCREMENT`, то следует обязательно задать модификатор `NOT NULL`, поскольку индекс не может содержать пустого значения.

## 19.6. Добавление записей в таблицу

Для добавления записей служит оператор `INSERT`:

```
INSERT INTO Имя_таблицы [(Список полей)]  
VALUES (Список констант);
```

Рассмотрим пример добавления записи в нашу таблицу:

```
INSERT INTO friends  
VALUES (0, 'Denis', 'denis@example.com', 'some comment');
```

Поскольку поле `id` является счетчиком, то вместо значения можно просто указать `0`. Поля `NAME` и `EMAIL` обязательны для заполнения. Список полей мы не указывали, чтобы упростить запрос. Но можно указать поля явно (причем порядок полей может быть не такой, какой был указан при создании таблицы):

```
INSERT INTO friends (COMMENT, EMAIL, NAME)  
VALUES ('some comment', 'denis@example.com', 'Denis');
```

Главное, чтобы указанные значения соответствовали полям.

Для вставки даты удобно использовать встроенную SQL-функцию `TO_DATE`:

```
TO_DATE('01/02/77', 'DD/MM/YY')
```

Например:

```
INSERT INTO BIRTHDAY (ADATE, NAME)  
VALUES (TO_DATE('01/02/77', 'DD/MM/YY'), 'Andrew');
```

## 19.7. Обновление записей

Обновление записей осуществляется оператором `UPDATE`, синтаксис которого следующий:

```
UPDATE таблица  
SET список_полей_и_значений  
WHERE условие;
```

Список полей и значений задается так:

```
поле = значение
```

Например:

```
UPDATE friends  
SET name='Igor', email='777@example.com'  
...
```

В случае с оператором `UPDATE` мы обязательно должны указать условие, иначе заданные нами значения полей будут установлены для всех записей, а это нежелательно. Чтобы изменить запись, мы должны ее идентифицировать. Поскольку у нас есть ключ, то сделать это достаточно просто — нужно указать значение ключевого поля:

```
UPDATE friends
SET name='Igor', email='777@example.com'
WHERE id=1;
```

Этот запрос найдет запись, для которой поле `id` равно единице, и изменит значения ее полей `name` и `email`. Поле `comment` останется без изменений, как и поле `id`.

Совсем другая была бы ситуация, если бы поле `id` не было ключевым, т. е. могло содержать повторяющиеся значения. Тогда теоретически в нашей таблице могли бы существовать две записи с одинаковым значением этого поля. Поскольку нам нужно обновить только одну запись, а не все, которые содержат такое же значение поля `id`, следует указать дополнительное условие — `LIMIT`:

```
UPDATE friends
SET name='Igor', email='777@example.com'
WHERE id=1 LIMIT 1;
```

Условие `LIMIT` задает количество записей, которые будут затронуты в результате обработки запроса. В этом случае будет изменена одна из записей (обычно первая попавшаяся). Но что делать, если нам требуется изменить не любую запись с заданным условием, а конкретную запись? Думаю, вы уже догадались — мы будем идентифицировать запись по другому полю, например, по `email`:

```
UPDATE friends
SET name='Igor', email='777@example.com'
WHERE email='user@someexample.com' LIMIT 1;
```

Понятно, что если запись со значением `'user@someexample.com'` поля `email` не будет найдена, то и обновления не произойдет. Для большей точности можно дополнительно указать поле `id`:

```
UPDATE friends
SET name='Igor', email='777@example.com'
WHERE (id=1) and (email='user@someexample.com')
LIMIT 1;
```

Вот теперь точно будет обновлена нужная запись.

## 19.8. Выборка записей

Выборка записей из таблицы выполняется с помощью оператора `SELECT`. Синтаксис этого оператора довольно сложный, поэтому рассмотрим его сокращенную версию:

```
SELECT список_полей
FROM таблица
[WHERE условие];
```

Например:

```
SELECT *
FROM friends;
```

В приведенном примере оператор `SELECT` выведет все записи из таблицы `friends`. Вместо списка полей мы указали `*`, поэтому будут выведены все поля. Понятно, что в нашей таблице есть поле `comment`, которое может содержать большой текст, и его вывод, мягко говоря, нежелателен, поэтому целесообразнее следующий оператор:

```
SELECT id, name, email
FROM friends;
```

Данный оператор также выведет все записи (пока у нас только одна запись) из таблицы `friends`, но при этом будут выведены все поля, кроме `comment`. Чтобы увидеть поле `comment`, которое соответствует определенной записи, нам нужно задать условие оператора `SELECT`. Предположим, что вам требуется вывести поле `comment` для первой записи (обычно ее `id` будет равен единице):

```
SELECT comment
FROM friends
WHERE id=1;
```

### **ПРИМЕЧАНИЕ**

Полный синтаксис оператора `SELECT` выглядит так:

```
SELECT [DISTINCT|ALL] {*| [поле1 AS псевдоним] [, ..., полеN AS псевдоним]}
FROM Имя_таблицы1 [, ..., Имя_таблицыN]
[WHERE условие]
[GROUP BY список полей] [HAVING условие]
[ORDER BY список полей]
```

О том, как использовать все его возможности, вы узнаете чуть позже в этой главе.

## **19.9. Удаление записей**

Условия удаления записи устанавливаются аналогично условиям обновления записи.

Удаление записей осуществляется оператором `DELETE`:

```
DELETE
FROM таблица
WHERE условие;
```

Вот небольшой пример:

```
DELETE
FROM friends
WHERE id=1;
```

Этот оператор удаляет первую запись в таблице. Удалить все записи из таблицы можно, указав оператор:

```
DELETE
FROM friends;
```

## 19.10. Встроенные функции

В SQL есть встроенные функции, например COUNT, MIN, MAX, AVG, SUM (существуют и другие, но эти — самые полезные). Функция COUNT вычисляет число элементов, MIN и MAX — минимальное и максимальное значения, AVG — среднее значение, а SUM — сумму элементов.

Использовать эти функции очень просто. Предположим, что у нас имеется таблица, созданная оператором CREATE (естественно, будем считать, что таблицу сразу после создания заполнили реальными данными):

```
CREATE TABLE books (  
  bid          int          auto_increment,  
  title        varchar(50)  not null,  
  autor        varchar(50)  not null,  
  price        double,  
  primary key (bid)  
);
```

Следующий оператор выведет число записей в таблице:

```
SELECT COUNT(*)  
FROM books;
```

Вообще, в этом операторе особого смысла нет, поскольку существует функция `mysql_num_rows()`, но иногда он тоже может пригодиться.

Функция MAX удобна для вычисления максимального значения указанного поля:

```
SELECT MAX(price)  
FROM books;
```

У MySQL очень много разных функций. Чтобы описать их все, нужно написать еще одну книгу, посвященную самому синтаксису SQL. Отмечу еще две полезные функции:

- `RAND()` — используется для выборки случайных записей;
- `LAST_INSERT_ID()` — возвращает ID последней добавленной записи.

Небольшой пример. Допустим, нужно выбрать 10 случайных товаров из категории 1:

```
SELECT * FROM items WHERE catid = 1  
ORDER BY RAND()  
LIMIT 10;
```

Первая часть запроса выбирает товары из категории 1, вторая обеспечивает сортировку в случайном порядке, а третья — ограничивает количество записей.

### ПРИМЕЧАНИЕ

Подробное описание всех функций MySQL на русском языке вы найдете по адресу <http://phpclub.ru/mysql/doc/index.html>.

У MySQL также очень богатый набор функций для работы с датой и временем. Практически все, что связано с обработкой дат, можно выполнить средствами

MySQL, не прибегая к использованию PHP. Прямая ссылка на описания функций даты и времени: <http://phpclub.ru/mysql/doc/date-and-time-functions.html>.

## 19.11. Группировка записей. Сложные запросы

Ранее мы выполняли запросы только к одной таблице. В самом простом случае этого достаточно. Но рано или поздно вам потребуется создать сложный проект, в котором придется выбирать записи из нескольких таблиц сразу. Тогда вам будет необходимо или выбирать все записи из нужных таблиц и писать процедуру обработки данных на PHP, или же просто учить SQL. Ведь намного проще передать запрос, а потом вывести его результат, чем вручную обрабатывать данные, — так что пусть за нас это сделает СУБД.

Предположим, что мы создаем небольшой интернет-магазин. Поэтому у нас будет три таблицы: клиенты, товары и заказы (листинг 19.2).

### Листинг 19.2. Таблицы базы данных интернет-магазина

```
CREATE TABLE CUSTOMERS
(
  C_NO      int          auto_increment,
  LOGIN     varchar(32)  NOT NULL,
  PASS      varchar(32)  NOT NULL,
  EMAIL     varchar(32)  NOT NULL,
  FIO       varchar(40)  NOT NULL,
  ADDRESS   varchar(50)  NOT NULL,
  CITY      varchar(25)  NOT NULL,
  PHONE     varchar(11)  NOT NULL,
  PRIMARY KEY (C_NO)
);

CREATE TABLE GOODS
(
  G_NO      int          auto_increment,
  DESCR     varchar(40)   NOT NULL,
  PRICE     double(9,2)   NOT NULL,
  QTY       int          NOT NULL,
  PRIMARY KEY (G_NO)
);

CREATE TABLE ORDERS
(
  O_NO      int          auto_increment,
  DT        date,
  C_NO      int          NOT NULL,
  G_NO      int          NOT NULL,
  QTY       int          NOT NULL,
```

```
primary key(O_NO)
);
```

### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *19-2.sql*), представленный в каталоге *Glava\_19* сопровождающего книгу электронного архива (см. приложение 4).

Думаю, назначение полей этих таблиц понятно и без моих комментариев. Сделаю только несколько небольших замечаний. Во-первых, мы продавать будем не сахар и не картошку, поэтому количество товара (поле `QTY`) везде целочисленное: две книги, один мобильный телефон и т. д. Во-вторых, индексы определены сразу, следовательно, `C_NO`, `G_NO` и `O_NO` — уникальные номера клиента, товара и заказа соответственно. В-третьих, поле `DESCR` содержит название товара. Конечно, на самом деле оно должно содержать еще и описание, а кроме того, должно быть поле `TITLE` с названием товара и поля `ART` и `PHOTO`, содержащие соответственно артикул товара и имя файла фотографии, но ведь это экспериментальная база и наша цель — не создание реального интернет-магазина, а изучение SQL, поэтому я упростил структуру таблиц до допустимого минимума.

После создания таблиц заполните их данными по своему усмотрению. Предположим, что в нашем магазине зарегистрировалось минимум два клиента, каждый из которых сделал несколько заказов. Нужно вычислить общую сумму всех заказов для каждого клиента. Для этого нам потребуется информация из всех трех таблиц:

- ☐ из `CUSTOMERS`, чтобы вывести информацию о клиенте (для простоты будем выводить только фамилию);
- ☐ из `GOODS`, ведь нам нужно знать цену каждого товара;
- ☐ из `ORDERS`, чтобы узнать, сколько заказали того или иного товара.

Введите следующий запрос:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE CUSTOMERS.C_NO = ORDERS.C_NO
GROUP BY ORDERS.C_NO;
```

Сразу скажу, что этот запрос неправильный, однако на его примере рассмотрим несколько важных моментов. Мы выбираем только фамилию клиента, затем функция `SUM` подсчитывает сумму заказа, умножая цену товара (`GOODS.PRICE`) на количество заказанного товара (`ORDERS.QTY`). Результат работы этой функции будет представлен как поле `A` (вы можете назвать это поле по-своему).

Обратите внимание на то, что выборка производится из всех трех таблиц (`CUSTOMERS`, `GOODS` и `ORDERS`). Мы выбираем тех клиентов, которые есть в таблице заказов (`CUSTOMERS.C_NO = ORDERS.C_NO`), а группировку записей осуществляем по полю `ORDERS.C_NO`. Вы уже заметили, как легко указать СУБД, к какой таблице обращаться: для этого просто нужно написать имя таблицы, а через точку — имя поля?

А теперь — почему этот запрос неправильный. Мы выполнили группировку заказов, и если в таблице `ORDERS` есть несколько заказов от одного и того же клиента, вы заметите, что группировка выполняется верно, но сумма заказа вычисляется неправильно. Это происходит потому, что мы забыли указать, что следует подсчитывать только тот товар, который есть в таблице заказов, и для этого потребуется добавить еще одно условие:

```
ORDERS.G_NO = GOODS.G_NO
```

Полная версия SQL-оператора, выполняющего нужное нам действие, выглядит так:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (CUSTOMERS.C_NO = ORDERS.C_NO) AND (ORDERS.G_NO = GOODS.G_NO)
GROUP BY ORDERS.C_NO;
```

Результат выполнения этого оператора представлен на рис. 19.1.

Показывает записи 0 - 1 (2 всего, Запрос занял 0.0009 сек)

SQL-запрос:

```
SELECT CUSTOMERS.FIO, SUM( GOODS.PRICE * ORDERS.QTY ) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (
    CUSTOMERS.C_NO = ORDERS.C_NO
)
AND (
    ORDERS.G_NO = GOODS.G_NO
)
GROUP BY ORDERS.C_NO
LIMIT 0, 30
```

[Правка] [Описать SQL] [Создать PHP-код] [Обновить]

Показать : 30 рядов от 0

В горизонтальном режиме, заголовки после каждых 100 ячеек

← T →	
FIO	A
Denis	950.00
Alex	1300.00

Рис. 19.1. Группировка записей: вывод общей суммы всех заказов с группировкой по клиенту

Группировку записей в таблице `ORDERS` выполняет оператор `GROUP BY`. Его действие можно ограничить с помощью оператора `HAVING`. По сути, это аналог `WHERE`, но для `GROUP BY`. Синтаксис `HAVING` аналогичен `WHERE`:

```
HAVING <условие>
```

Предположим, что нам нужно вывести всех клиентов, общая сумма всех заказов которых превышает 500 (рублей, долларов — не важно). Модифицированный запрос будет выглядеть так:

```
SELECT CUSTOMERS.FIO, SUM(GOODS.PRICE * ORDERS.QTY) AS A
FROM CUSTOMERS, GOODS, ORDERS
WHERE (CUSTOMERS.C_NO = ORDERS.C_NO) AND (ORDERS.G_NO = GOODS.G_NO)
GROUP BY ORDERS.C_NO
HAVING A > 500;
```

Теперь, когда вы знаете основные SQL-операторы, можно перейти к рассмотрению функций PHP для работы с MySQL.

## 19.12. Копирование записей из одной таблицы в другую

Помню, при коллективной работе над одним проектом возникла не очень приятная ситуация. В базе данных было две таблицы: `book_alist` и `book_list`. В первой таблице предполагалось хранить список статей, а во второй — список книг.

Один нехороший человек поместил статьи в таблицу `book_list`. Целых 3800 записей. Сами понимаете, вручную копировать их никто не собирался. Писать PHP-сценарий тоже не хотелось, поэтому возникшую проблему я решил средствами SQL, тем более что это совсем не сложно. В нашем примере в поле `CID` хранится номер категории, в которую были добавлены все статьи. Теперь мы их перемещаем в таблицу `book_alist`:

```
INSERT INTO book_alist
SELECT * FROM book_list WHERE CID = 14
```

Запрос, как видите, очень прост. Мы берем все записи, где `CID = 14`, и вставляем их в таблицу `book_alist`.

Если же структура таблиц разная, то нужно указать поля, которые следует задействовать при выборке и вставке (в реальной задаче обратите внимание на порядок, в котором вы указываете поля: поля из `SELECT` должны соответствовать полям, указанным в `INSERT`):

```
INSERT INTO table1(CID, TITLE, PAGES)
SELECT BID, TNAME, PG FROM table2 WHERE CID = 14
```

## 19.13. Кеширование запросов

В предыдущих изданиях книги в этой главе были описаны только основы SQL. Теперь мы попробуем окунуться в более сложные материи. Ведь книга должна быть интересной не только для новичков, но и для уже состоявшихся PHP-программистов.

Кеширование запросов позволяет существенно повысить производительность PHP-приложений. Ведь рано или поздно все мы сталкиваемся с проблемой оптимизации. Вроде бы написали сценарий, он работает, хорошо работает, но до определенного

момента — когда нагрузка на сервер становится такой, что сценарий начинает тормозить.

Здесь мы рассмотрим встроенный в MySQL механизм кеширования запросов. По умолчанию этот режим выключен. Запустите программу `mysql`, чтобы просмотреть параметры сервера прямо сейчас:

```
mysql> show variables like 'query_cache%';
+-----+-----+
| Variable_name      | Value      |
+-----+-----+
| query_cache_limit   | 1048576    |
| query_cache_min_res_unit | 4096      |
| query_cache_size    | 0          |
| query_cache_type    | ON         |
| query_cache_wlock_invalidate | OFF      |
+-----+-----+
```

За кеш отвечает параметр `query_cache_size`. Как можно видеть, сейчас он равен 0, т. е. кеш выключен. Для включения кеша запросов выполните запрос:

```
set @@global.query_cache_size= 67108864;
```

Здесь мы устанавливаем размер кеша равным 64 Мбайт ( $64 \times 1024 \times 1024$ ), чего вполне достаточно для большинства случаев.

Для сохранения изменений нужно отредактировать файл `my.cnf` и добавить в раздел `[mysqld]` строку:

```
query_cache_size=64M
```

Кроме параметра `query_cache_size`, вы можете включить параметр `query_cache_limit`, задающий максимальный объем результата, который может быть помещен в кеш. Результаты, превышающие этот лимит, кешироваться не будут.

После включения кеш начинает работать автоматически. Больше ничего делать не нужно, запросы менять тоже не нужно.

Алгоритм работы кеша:

- при каждом запросе `SELECT` вычисляется хеш-сумма строки запроса, далее она ищется в кеше, и если она там есть — сразу возвращается результат. Если нет, вычисляется результат и заносится в кеш. При следующем таком же запросе результат будет взят из кеша, что сократит время выполнения запроса;
- при каждом запросе `INSERT`, `UPDATE`, `REPLACE`, `DELETE`, `TRUNCATE` из кеша удаляются все записи, касающиеся таблицы, которая подверглась обновлению.

При составлении запросов помните, что кеш чувствителен к регистру символов, поэтому запросы `SELECT * FROM tab` и `select * from tab` — два разных запроса. Чтобы вы потом не удивлялись, почему кеш не работает.

Не кешируются следующие запросы:

- ☐ в которых используются значения локальных переменных;
- ☐ использующие недетерминированные функции — например: `RAND()`, `NOW`, `SLEEP()`, `CURTIME()`, `LAST_INSERT_ID()` и т. д.;
- ☐ в которых вызываются определенные пользователем процедуры;
- ☐ генерирующие предупреждения;
- ☐ некоторые другие.

Чтобы отключить кеш для определенного запроса, добавьте в него директиву `SQL_NO_CACHE`, например:

```
SELECT SQL_NO_CACHE * FROM news;
```



## ГЛАВА 20

# Функции для работы с MySQL

## 20.1. Способы работы с базой данных

В PHP существуют разные способы работы с базой данных:

- ❑ через расширение `mysql` — ранее самый популярный способ, который и рассматривался в предыдущих изданиях этой книги. Проблема в том, что из PHP 7 это расширение удалено совсем, и вам придется использовать расширения `mysqli` или `PDO` (см. далее);
- ❑ через расширение `mysqli` — с этим расширением можно работать как в PHP 5, так и в PHP 7/8, в некоторых аспектах оно даже удобнее, чем `PDO`, и предоставляет объектно-ориентированный способ работы с базой данных;
- ❑ через расширение `PDO` (PHP Data Objects) — универсальное расширение, позволяющее работать с разными СУБД, а не только с MySQL. Если не хотите привязываться к MySQL, используйте `PDO` — тогда, если нужно будет перейти на другую СУБД, все, что вам придется изменить, — это строку подключения, а не переписывать все приложение;
- ❑ вы также можете задействовать технологию ORM<sup>1</sup> (Object-Relational Mapping), входящую в состав используемого фреймворка PHP, — например, ORM `Propel` или `Doctrine`. О том, как работать с базой данных во фреймворке `Laravel` мы поговорим в следующей главе.

К какому способу прибегнуть? Самый простой способ — использовать расширение `mysqli`, если, конечно, это то, что вам нужно. Если же вы создаете серьезное приложение, которое может поддерживать несколько СУБД, тогда вам придется обратиться к `PDO`. А вот если вы используете фреймворк, тогда вам нужно задействовать средства, предоставляемые фреймворком, а не расширения PHP.

---

<sup>1</sup> ORM (Object-Relational Mapping) — технология программирования, связывающая базы данных с концепциями объектно-ориентированных языков программирования, создающая «виртуальную объектную базу данных».

## 20.2. Расширение *mysqli*

### 20.2.1. Подключение к серверу MySQL

Прежде чем начать работу с базой данных, нужно выбрать ее и подключиться к серверу баз данных. Это можно сделать так:

```
$mysqli = new mysqli($dbhost,$dbuser,$dbpass,$db);
if ($mysqli->connect_errno) {
    die($mysqli->connect_error);
}
```

В конструктор объекта `mysqli` нужно передать имя сервера БД, имя пользователя, пароль и название базы данных. Конструктор выглядит так:

```
public mysqli::__construct(
    string $hostname = ini_get("mysqli.default_host"),
    string $username = ini_get("mysqli.default_user"),
    string $password = ini_get("mysqli.default_pw"),
    string $database = "",
    int $port = ini_get("mysqli.default_port"),
    string $socket = ini_get("mysqli.default_socket")
)
```

Кроме имени базы данных, вы можете задать также порт (он указывается после базы данных — по умолчанию 3306) и имя сокета. А можете вообще ничего не указывать, если все эти параметры сохранены в файле `php.ini`. По умолчанию в `php.ini` определены следующие параметры доступа:

```
mysqli.default_host=127.0.0.1
mysqli.default_user=root
mysqli.default_pw=""
mysqli.default_port=3306
mysqli.default_socket=/tmp/mysql.sock
```

Как правило, на практике используется приведенный в самом начале этого раздела способ подключения. Когда вы получили соединение с базой данных в переменную `$mysqli`, все взаимодействие с базой данных будет осуществляться через эту переменную.

Для закрытия соединения служит метод `close()`. Пример:

```
$mysqli->close();
```

Нужно помнить, что открытые непостоянные соединения MySQL и наборы результатов автоматически закрываются при уничтожении их объектов. Явное закрытие открытых соединений и освобождение наборов результатов не обязательно. Открыть постоянное соединение можно, добавив префикс `p:` к имени узла — например, `p:localhost`. Однако при использовании таких соединений вам нужно закрывать их вручную и не забывать об этом, иначе можно легко превысить лимит на количество соединений с базой данных.

### 20.2.2. Передача запросов серверу

Теперь рассмотрим метод `query()`, которая передает запрос серверу MySQL:

```
public mysqli::query(string $query, int $result_mode = MYSQLI_STORE_RESULT):  
mysqli_result|bool
```

Строка `query` должна содержать SQL-запрос. Вот пример:

```
$result = $mysqli->query('select * from users');
```

Переменная `$result` будет содержать результат выполнения запроса. Для его анализа используются методы, представленные в табл. 20.1.

Таблица 20.1. Функции для работы с результатом SQL-запроса

Функция	Описание
<code>int mysql_num_rows</code>	Содержит количество записей результата запроса
<code>public mysqli_result::fetch_array(int \$mode = MYSQLI_BOTH): array null false</code>	Возвращает следующую запись результата запроса в виде массива. Очень удобный метод, мы будем часто ее использовать. Тип массива указывается с помощью параметра:  <code>MYSQLI_ASSOC</code> — ассоциативный массив;  <code>MYSQLI_NUM</code> — численный массив;  <code>MYSQLI_BOTH</code> — оба типа массива (по умолчанию)
<code>public mysqli_result::fetch_assoc(): array null false</code>	Возвращает следующую запись результата запроса в виде ассоциативного массива
<code>public mysqli_result::fetch_field(): object false</code>	Возвращает полную информацию о конкретном поле записи.
<code>public mysqli_result::fetch_row(): array null false</code>	Обрабатывает следующую запись результата запроса и возвращает неассоциативный массив.
<code>public mysqli_result::fetch_object(string \$class = "stdClass", array \$constructor_args = []): object null false</code>	Возвращает следующую запись результата запроса в виде объекта.

В табл. 20.1 приведены далеко не все доступные методы, с дополнительными вы можете ознакомиться по адресу: <https://www.php.net/manual/ru/book.mysql.php>.

Предположим, что у нас есть таблица `users`, содержащая всего три поля:

- ☐ Username;
- ☐ Email;
- ☐ Password.

Рассмотрим следующий пример вывода всех записей этой таблицы с помощью метода `fetch_array()` (листинг 20.1).

**Листинг 20.1. Пример использования метода `fetch_array()`**

```
// считаем, что к серверу мы уже подключились и базу данных выбрали

// это наш SQL-запрос
$q = 'select * from users';

// получаем результат
$r = $mysqli->query($q);

// по запросу мы получили все записи, поэтому значение, возвращенное
// num_rows, равно количеству записей в таблице
echo "Количество записей в таблице: " . $r->num_rows();

// выводим результат как HTML-таблицу
echo "<TABLE>";

// вызываем fetch_array в цикле while, второй параметр можно
// не указывать
while ($row = $r->fetch_array(MYSQLI_ASSOC)) {

    // забудем об оптимизации — так красивее и проще код
    echo "<TR>";
    echo "<TD>$row[username]</TD>";
    echo "<TD>$row[email]</TD>";
    echo "<TD>$row[password]</TD>";
    echo "</TR>";

}

// таблица users выведена
echo "</TABLE>";
```

Теперь рассмотрим пример вывода этой же таблицы, но с помощью метода `fetch_object()` (листинг 20.2).

**Листинг 20.2. Пример использования метода `fetch_object()`**

```
// отличия от листинга 20.1 начинаются отсюда

// вызываем fetch_object в цикле while
while ($row = $r->fetch_object()) {

    echo "<TR>";
    echo "<TD>$row->username</TD>";
    echo "<TD>$row->email</TD>";
    echo "<TD>$row->password</TD>";
    echo "</TR>";

}
```

```
// таблица users выведена  
echo "</TABLE>";
```

По быстрдействию эти методы практически одинаковы, поэтому особой разницы вы не заметите.

### 20.2.3. Метод *real\_escape\_string()*

Предположим, что у нас есть следующий оператор:

```
$q = "insert into tab values(0, \"\$content\");
```

Этот оператор формирует запрос, добавляющий в таблицу `tab` значение переменной `$content`. Вроде бы все здесь нормально, и запрос должен выполняться без особых проблем. Когда в переменной `$content` нет кавычек, то так оно и будет, например:

```
$content = "Фирма Рога и копыта";  
$q = "insert into tab values(0, \"\$content\");  
$mysqli->query($q);
```

Но если в переменной `$content` есть кавычки:

```
$content = 'Фирма "Рога и копыта" — лучший производитель стульев';
```

наш запрос выполнен не будет.

Чтобы избавиться от этой проблемы, нужно вызвать функцию `mysql_real_escape_string()` для обработки значения переменной `$content` перед помещением информации в базу данных:

```
$content = 'Фирма "Рога и копыта" — лучший производитель стульев';  
$content = $mysqli->real_escape_string($content);  
$q = "insert into tab values(0, \"\$content\");  
$mysqli->query($q);
```

## 20.3. Расширение PDO

### 20.3.1. Соединение с базой данных

Способ подключения к базе данных у PDO немного отличается от привычного. Прежде всего, нужно сформировать так называемую *DSN-строку* — строку с именем источника данных (Data Source Name)<sup>1</sup>.

Выглядит DSN-строка так:

```
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
```

В ней мы указываем имя сервера БД (`host`), имя самой БД (`dbname`), а также кодировку.

---

<sup>1</sup> См.: [https://en.wikipedia.org/wiki/Data\\_source\\_name](https://en.wikipedia.org/wiki/Data_source_name).

После формирования DSN-строки можно сформировать массив атрибутов (опций):

```
$opt = array(
    PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
);
```

В этом массиве задаются различные атрибуты (опции) соединения. Например, в нашем случае мы задаем режим выдачи ошибок и режим получения (`FETCH_MODE`) по умолчанию. Я настоятельно рекомендую на время разработки и отладки включить режим выдачи ошибок в виде исключения (как показано в примере), поскольку в остальных режимах PDO не сообщает никакой полезной информации об ошибке. Когда сценарий будет протестирован, тогда можно этот режим не использовать.

Режим получения по умолчанию удобнее задать в опциях, чтобы потом не писать его в каждом запросе. Вам это быстро надоест.

Массив с опциями, как вы уже догадались, можно и не указывать. А можно его указать при подключении, что удобно делать, когда опций всего одна:

```
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true
));
```

Атрибут `ATTR_PERSISTENT` позволяет создать постоянные соединения с базой данных, чтобы избежать повторных подключений каждый раз, когда нужно организовать с ней обмен. Кроме того, постоянные соединения помогают снизить нагрузку на сервер БД.

Расширение PDO обладает огромным количеством predefined констант и атрибутов. Ознакомиться с полным их списком можно по адресу: <https://php.net/manual/ru/pdo.constants.php>.

Итак, рассмотрим процесс подключения:

```
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$opt = array(
    PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
);
$pdo = new PDO($dsn, $user, $pass, $opt);
```

Здесь видно, что сформирована DSN-строка, массив атрибутов и выполнено подключение к базе данных с использованием имени пользователя и пароля, содержащихся в переменных `$user` и `$pass`.

При подключении можно обрабатывать исключительные ситуации, чтобы завершить выполнение сценария, если подключение не удалось:

```
try {
    $pdo = new PDO($dsn, $user, $password);
```

```
} catch (PDOException $e) {  
    die('Подключение не удалось: ' . $e->getMessage());  
}
```

Впрочем, поскольку мы ранее включили режим `ERRMODE_EXCEPTION`, PHP и так выведет ошибку, поэтому обработка исключений в момент подключения к базе данных будет излишней.

## 20.3.2. Выполнение запросов и чтение результата

Выполнить запрос можно двумя способами: первый — через метод `query()` и второй — через подготовленные выражения. Разберемся, когда нужно использовать тот или иной.

Если в запрос не передаются никакие переменные, то проще использовать первый способ. Пример:

```
$stmt = $pdo->query('SELECT name FROM products');  
while ($row = $stmt->fetch())  
{  
    echo $row['name'] . "\n";  
}
```

- Метод `query()` возвращает объект PDO statement. Он похож на `mysql resource`, который возвращала функция `mysql_query()`. После того как у нас появился объект этого типа, мы можем, как и в случае с расширением `mysql`, использовать цикл `while` для перебора результата (можно также использовать и цикл `foreach` — как вам удобнее).
- Если же в запрос передается хотя бы одна переменная, нужно использовать подготовленные выражения — обычные SQL-запросы, в которых вместо переменной ставится заменитель `?`.

PDO поддерживает как позиционные заменители `?`, для которых важен порядок передаваемых переменных, так и именованные `:name`, для которых порядок не важен. Рассмотрим примеры:

```
$sql = 'SELECT name FROM products WHERE hold = ?';  
$sql = 'SELECT name FROM products WHERE price = :price';
```

Чтобы выполнить такой запрос, его нужно сначала подготовить методом `prepare()`:

```
$stmt = prepare('SELECT name FROM products WHERE hold = ?');  
$stmt->execute(array($hold));  
$stmt = prepare('SELECT name FROM products WHERE price = :price');  
$stmt->execute(array('price' => $price));
```

Поначалу такой способ кажется чуть сложнее, чем при использовании расширения `mysql`, но к нему быстро привыкаешь. Итак, подготавливаем запрос, а затем в метод `execute()` передаем массив значений, которые будут в запрос подставлены.

Нужно отметить, что расширение PDO поддерживает возможность привязки переменной к запросу с помощью методов `bindValue()` и `bindParam()`. С их использова-

нием в метод `execute()` уже ничего передавать не нужно. Примеры привязки можно посмотреть в руководстве по PDO: <https://php.net/manual/ru/pdostatement.bindvalue.php>.

Рассмотрим пример, позаимствованный из руководства:

```
$calories = 150;
$colour = 'red';
$stmt = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$stmt->bindValue(':calories', $calories, PDO::PARAM_INT);
$stmt->bindValue(':colour', $colour, PDO::PARAM_STR);
$stmt->execute();
```

Здесь мы связываем заменитель `:calories` с переменной `$calories` типа `PDO::PARAM_INT`, а переменная `$colour` типа `PDO::PARAM_STR` связывается с заменителем `colour`. После этого в метод `execute()` ничего передавать не нужно. Вы вольны использовать тот метод, который вам больше нравится.

На мой взгляд, если запросы несложные, можно обойтись без метода `bindValue()`, но если запросы сложные и содержат много переменных, удобнее использовать связывание переменных с запросом.

После вызова метода `execute()` обработать результат можно таким же способом, как и раньше:

```
$stmt = $pdo->prepare('SELECT name FROM products WHERE hold = ?');
$stmt->execute([$GET['hold']]);
foreach ($stmt as $row)
{
    echo $row['name'] . "\n";
}
```

Прелесть подготовленных выражений совсем не в том, чтобы усложнить жизнь программиста, а, наоборот, сделать ее проще. Подготовленные выражения можно выполнять множество раз с разными наборами данных. Например:

```
$data = array(
    1 => 1000,
    4 => 1500,
    7 => 300,
);

$stmt = $pdo->prepare('UPDATE products SET bonus = bonus + ? WHERE id = ?');
foreach ($data as $id => $bonus)
{
    $stmt->execute([$id, $bonus]);
}
```

Такой гибкости при использовании расширения `mysql` просто не было. Здесь, как видно из кода, мы устанавливаем для группы продуктов бонус, который получит

клиент, купивший тот или иной продукт. Массив `data` задает как ID продукта, так и назначенный ему бонус.

### 20.3.3. Получение данных

Ранее уже был рассмотрен метод `fetch()`, который мы использовали для последовательного получения записей из базы данных. Этот метод является аналогом функции `mysql_fetch_array()`, но работает иначе: вместо множества функций, которые были в расширении `mysql`, здесь используется одна функция, но работает она по-разному. Ее поведение контролируется переданным параметром. Например, если использовать параметр `PDO::FETCH_LAZY`, то в этом режиме не будет тратиться лишняя память, и к колонкам можно будет обращаться любым из способов: по имени, по свойству или просто через индекс.

Пример:

```
$stmt = $pdo->prepare('SELECT name FROM products WHERE bonus > ?');
$stmt->execute([$GET['bonus']]);
while ($row = $stmt->fetch(PDO::FETCH_LAZY))
{
    echo $row[0] . "\n";
    echo $row['name'] . "\n";
    echo $row->name . "\n";
}
```

В PDO имеется метод `fetchColumn()`, позволяющий получить значение единственной колонки. В предыдущем примере мы получали только одну колонку с именем `name`. В таком случае удобнее воспользоваться методом `fetchColumn()`:

```
$stmt = $pdo->prepare("SELECT name FROM products WHERE bonus>?");
$stmt->execute(array($bonus));
$name = $stmt->fetchColumn();
```

Можно еще использовать метод `fetchAll()`, возвращающий массив, состоящий из всех строк, которые вернул запрос. Используйте этот метод с осторожностью, особенно если запрос может вернуть много данных. В этом случае лучше использовать обычный цикл и в нем вызывать метод `fetch()`.

Метод `fetchAll()` становится просто незаменимым при использовании шаблонов — программисту не нужно писать циклы вручную, что позволяет сократить количество кода.

Получить пары «ключ-значение» можно, передав константу `PDO::FETCH_KEY_PAIR`:

```
$data = $pdo->query('SELECT id, name FROM products')->fetchAll(PDO::FETCH_KEY_PAIR);
```

Результат:

```
array (
    101 => 'Fly EraStyle 3',
```

```

110 => 'Samsung Galaxy J3',
125 => 'Nokia 3110'
)

```

При желании можно получить только одну колонку. Для этого нужно использовать константу `PDO::FETCH_COLUMN`:

```
$data = $pdo->query('SELECT name FROM users')->fetchAll(PDO::FETCH_COLUMN);
```

Результат:

```

array (
  0 => 'Fly EraStyle 3',
  1 => 'Samsung Galaxy J3',
  2 => 'Nokia 3110'
)

```

### 20.3.4. Особенности использования операторов *LIKE*, *LIMIT* и *IN*

У PDO есть одна неприятная особенность: заменителем может быть только строка или число, — вы не можете через заменитель вставить ни часть строки, ни набор строк, ни идентификатор. Именно поэтому для оператора `LIKE` нужно сначала подготовить строку полностью, а потом передать ее в запрос:

```

$name = "%$name%";
$stmt = $pdo->prepare("SELECT * FROM products WHERE name LIKE ?");
$stmt->execute(array($name));
$data = $stmt->fetchAll();

```

С оператором `LIMIT` дела обстоят сложнее. PDO работает в режиме эмуляции, и все данные, которые передаются напрямую в метод `execute()`, формируются как строки, а именно: обрамляются кавычкам и экранируются. Именно поэтому строка `LIMIT ?, ?` превращается в `LIMIT '5', '10'`, что вызывает ошибку синтаксиса.

Обойти проблему можно двумя способами:

❑ первый способ — отключить режим эмуляции:

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

❑ второй способ — использовать метод `bindValue()` и принудительно указывать числовым переменным тип `PDO::PARAM_INT`:

```

$stmt = $pdo->prepare('SELECT * FROM products LIMIT ?, ?');
$stmt->bindValue(1, $limit_from, PDO::PARAM_INT);
$stmt->bindValue(2, $per_page, PDO::PARAM_INT);
$stmt->execute();
$data = $stmt->fetchAll();

```

Поскольку вместо заменителя можно использовать только число или строку, то предоставить набор данных в оператор `IN` не получится. Что делать? Придется

динамически формировать две переменные: набор вопросов через запятую — по количеству элементов в IN, а также массив данных для подстановки:

```
$arr = array(1,2,3);
$in  = str_repeat('?', count($arr) - 1) . '?';
$sql = "SELECT * FROM table WHERE column IN ($in)";
$stmt = $db->prepare($sql);
$stmt->execute($arr);
$data = $stmt->fetchAll();
```

### 20.3.5. Имена таблиц и полей при работе с PDO

К сожалению, здесь все по старинке... Вам придется вручную указывать идентификаторы: имена таблиц и полей. При ручном форматировании идентификаторов нужно заключать их в обратные одинарные кавычки (') и экранировать их путем удвоения:

```
$field = "`".str_replace("`","``",$_GET['field'])."`";
$sql   = "SELECT * FROM `table` ORDER BY $field";
```

Такой код защищает нас от классической инъекции, но все равно шанс остается. Именно поэтому настоятельно рекомендуется использовать SafeMySQL — специальный класс для безопасной работы с MySQL: <http://phpfaq.ru/safemysql>.

### 20.3.6. Запросы вставки и обновление

Самое «вкусное» я оставил напоследок. Запросы INSERT и UPDATE, особенно те, которые вы увидите в официальных руководствах, могут вызвать взрыв головного мозга, особенно если вы до этого привыкли использовать расширение mysql и передавать запросы INSERT и UPDATE в функцию mysql\_query().

Сократить код запросов INSERT и UPDATE можно с помощью функции dataSet(), но она будет правильно работать только в том случае, если имена полей в форме будут соответствовать именам полей в таблице. Тогда имена можно будет перечислить всего один раз и использовать функцию dataSet() для сборки запросов INSERT и UPDATE. Код функции приводится далее:

```
function dataSet($allowed, &$amp;values, $source = array()) {
    $set = '';
    $values = array();
    if (!$source) $source = &$_POST;
    foreach ($allowed as $field) {
        if (isset($source[$field])) {
            $set.="`".str_replace("`","``",$field)."`". "=".$field, ";";
            $values[$field] = $source[$field];
        }
    }
    return substr($set, 0, -2);
}
```

После этого код вставки новой записи из формы будет таким:

```
$allowed = array("id", "name", "price");           // список допустимых полей
$sql = "INSERT INTO products SET ".dataSet($allowed,$values);
$stmt = $dbh->prepare($sql);
$stmt->execute($values);
```

Обновить запись можно так:

```
$allowed = array("id", "name", "price");           // список допустимых полей
$sql = "UPDATE products SET ".dataSet($allowed,$values)." WHERE id = :id";
$stmt = $dbh->prepare($sql);
$values["id"] = $_POST['id'];
$stmt->execute($values);
```

Конечно, может такое решение и не совсем эффективно, но тем не менее оно работает.

Напоследок напомним адрес официального руководства по PDO:

**<https://www.php.net/manual/ru/book.pdo.php>**



## ГЛАВА 21

# Работа с базой данных в Laravel

## 21.1. Способы работы с базой данных

Фреймворк Laravel поддерживает три способа работы с базой данных:

- ❑ с помощью сырых запросов — вы передаете заранее подготовленный SQL-код (как это мы делали с помощью расширения `mysqli`), отправляете запрос, получаете и обрабатываете результат;
- ❑ с помощью конструктора запросов — вы указываете только таблицу, уточняющие параметры, а также действие, которое хотите выполнить. При этом сам SQL-запрос вы не пишете. Это наиболее удобный способ работы с базой данных в небольших проектах;
- ❑ посредством системы Eloquent — здесь каждой таблице в базе данных соответствует свой класс-модель, который используется для работы с этой таблицей. Подойдет для более крупных проектов, где нужна большая абстракция, чем в небольших проектах.

Что же касается самой СУБД, то Laravel поддерживает MySQL (и его варианты вроде MariaDB), PostgreSQL, SQLite, SQLServer.

Параметры базы данных вашего приложения хранятся в файле `config/database.php`. СУБД по умолчанию выбирается так:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Вместо значения `'mysql'` вы можете указать следующие значения: `'sqlite'`, `'pgsql'`, `'sqlsrv'`. Каждое из этих значений соответствует используемому типу СУБД. Далее в массиве `connections` указываются параметры доступа к той или иной базе данных. Рассмотрим параметры доступа к MySQL:

```
'mysql' => [  
    'driver' => 'mysql',  
    'url' => env('DATABASE_URL'),  
    'host' => env('DB_HOST', '127.0.0.1'),  
    'port' => env('DB_PORT', '3306'),  
    'database' => env('DB_DATABASE', 'forge'),
```

```

'username' => env('DB_USERNAME', 'forge'),
'password' => env('DB_PASSWORD', ''),
'unix_socket' => env('DB_SOCKET', ''),
'charset' => 'utf8mb4',
'collation' => 'utf8mb4_unicode_ci',
'prefix' => '',
'prefix_indexes' => true,
'strict' => true,
'engine' => null,
'options' => extension_loaded('pdo_mysql') ? array_filter([
    PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
]) : [],
],

```

Назначение параметров можно понять по их названию. Как правило, вам не нужно будет изменять параметры доступа к базе данных в файле `database.php`, поскольку конструкция `env()` говорит о том, что параметры нужно взять из файла `.env`:

```

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=laravel
DB_PASSWORD=P@ssW0rd2023!

```

Чтобы ваш контроллер получил возможность использовать базу данных, нужно определить использование фасада DB:

```
use Illuminate\Support\Facades\DB;
```

или просто:

```
use DB;
```

## 21.2. Сырые (прямые) запросы

Принцип сырых (raw) запросов заключается в том, что разработчик сам определяет SQL-запрос к базе данных и передает его в фасад DB, который, в свою очередь, подключается к БД и отправляет запрос, получает ответ и предоставляет его разработчику. Отличие от расширения `mysql` заключается в том, что в этом расширении для всех типов запросов используется один и тот же метод — `query()`, а фасад DB определяет сразу пять методов — по одному для каждого типа запроса: `select()`, `update()`, `insert()`, `delete()`, `statement()`. Точнее, первые четыре — предназначены для соответствующих SQL-запросов, а последний — для выполнения произвольного запроса.

❑ Метод `select()` используется для выполнения запроса `SELECT`. Вот как можно выбрать все активные записи блога:

```

$post = DB::select('select * from blog where active = 1');
return view('blog.index', ['posts' => $posts]);

```

Метод `select()` возвращает массив результатов. Каждый результат в массиве будет объектом PHP `stdClass`, что позволяет обращаться к значениям результатов так:

```
foreach ($posts as $item) {
    echo $item->title;
}
```

- ❑ Метод `insert()` используется для вставки данных. Первым аргументом он принимает запрос, вторым — привязки параметров. Привязки обеспечивают защиту от SQL-инъекций:

```
DB::insert('insert into users (id, name) values (?, ?)', [0, 'Ivan']);
```

Здесь вместо первого значения (для параметра `id`) будет использоваться 0, а для второго — `'Ivan'`.

- ❑ Метод `update()` предназначен для обновления записи:

```
$up = DB::update('update blog set active = 0 where tags = ?', ['macOS']);
```

Этот запрос деактивирует все записи блога, для которых в качестве тега выступает строка `'macOS'`.

- ❑ Для запросов, которые не возвращают никаких значений и не относятся к запросам `SELECT`, `INSERT` или `UPDATE`, принято использовать метод `statement()`:

```
DB::statement('drop table blog');
```

## 21.3. Конструктор запросов

*Конструктор запросов* — гораздо более удобный способ работы с базой данных, чем сырые запросы. Принцип его следующий: конструктор запросов самостоятельно создает SQL-запрос, а разработчику нужно только указать имя таблицы, задать уточняющие условия (методом `where()`) и сказать, что он хочет сделать с записями таблицы (методами `get()`, `delete()`, `update()` и др.).

Но обо всем по порядку. Представим, что у нас есть таблица `blog` со следующими полями:

- ❑ `id` — числовое поле, идентификатор записи, первичный ключ;
- ❑ `dt` — дата/время (тип `datetime`) написания записи;
- ❑ `title` — заголовок записи блога;
- ❑ `content` — контент записи блога;
- ❑ `active` — содержит 1, если запись включена и должна отображаться на сайте, и 0, если запись выключена.

Табличка простая, но ее будет вполне достаточно, чтобы продемонстрировать использование конструктора запросов. В методе, выводящем все записи блога, нам нужно получить все активные записи блога и передать их в представление:

```
$posts = DB::table('blog')->where('active', 1)->get();  
return view('list', ['posts' => $posts]);
```

Далее, в представлении `main.blade.php`, мы можем вывести почту блога так:

```
@foreach ($posts as $item)  
    <h2>{{ $item->title }}</h2>  
    <a href="/post/{{ $item->id }}">Читать далее</a>  
@endforeach
```

В контроллере, отвечающим за отображение конкретной записи блога, вам нужно выбрать не все записи блога, а только одну, которая соответствует заданному условию. Для этого используется метод `first()`:

```
$post = DB::table('blog')->where('id', $id)->first();  
if (is_object($post))  
    return view('post', ['post' => $post]);  
else  
    return abort(404);
```

Узнать количество записей в таблице `blog` можно с помощью метода `count()`:

```
$num_rows = DB::table('blog')->count();
```

Также вы можете использовать методы `max()`, `min()`, `avg()`, позволяющие получить значения, возвращаемые агрегатными функциями:

```
$min_price = DB::table('products')->min('price');  
$avg_price = DB::table('products')->avg('price');  
$max_price = DB::table('products')->max('price');
```

Метод `select()` позволяет выбрать только определенные поля. Представим, что нам нужно выбрать только поля `title` и `content` из таблицы `news`:

```
$titles = DB::table('blog')->select('id', 'title')->get();
```

Для построения запроса вида `SELECT DISTINCT`, т. е. для получения отличающихся результатов:

```
$posts = DB::table('blog')->distinct()->get();
```

Метод `where()`, который можно вызвать на экземпляре конструктора запроса, используется для добавления в запрос условий. В самом простом случае нужно указать три аргумента: имя столбца, оператор (должен поддерживаться базой данных), значение для сравнения со столбцом:

```
$posts = DB::table('blog')->where('active', '>', 0)->get();
```

Для удобства, если вы хотите использовать оператор `'='`, вы можете ничего не указывать в качестве оператора:

```
$post = DB::table('blog')->where('id', $id)->first();
```

Аналогичным образом можно использовать оператор `like`. Найдем все новости, в заголовке которых есть строка `$search`:

```
$posts = DB::table('blog')->where('title', 'like', '%' . $search . '%')->get();
```

При необходимости можно передать массив условий (все они будут объединены логическим оператором AND):

```
$posts = DB::table('blog')->where([
    ['active', '=', 1],
    ['title', 'like', '%Apple%']
])->get();
```

При необходимости добавить второе условие с использованием оператора OR нужно использовать метод `orWhere()`:

```
$users = DB::table('users')
    ->where('active', '>', 0)
    ->orWhere('login', 'like', 'admin%')
    ->get();
```

Конструктор запросов предоставляет несколько методов, дающих возможность существенно облегчить проектирование запросов `where()`. Например, метод `whereBetween()` позволяет проверить принадлежность значения столбца к определенному диапазону значений:

```
$post = DB::table('post_views')->whereBetween('views', [100,200])->get();
```

Здесь мы получаем записи, где количество просмотров находится в диапазоне от 100 до 200.

Для сортировки результатов поиска используется метод `orderBy()`:

```
$posts = DB::table('blog')
    ->orderBy('dt', 'desc')
    ->get();
```

Для вставки записей — метод `insert()`:

```
DB::table('blog')->insert([
    'id' => $id,
    'dt' => $dt,
    'title' => $title,
    'content' => $content,
    'active' => 1
]);
```

Обновить записи можно с помощью метода `update()`:

```
DB::table('blog')->where('id', $id)->update([
    $title => $new_title
]);
```

Для удаления записей из таблицы служит метод `delete()`:

```
// Удаляет все записи из таблицы
DB::table('blog')->delete();
// Удаляет все неактивные записи из таблицы
DB::table('blog')->where('active', 0)->delete();
```

Мы рассмотрели далеко не все, а только базовые методы конструктора запросов. Тем не менее этого материала достаточно, чтобы вы смогли оценить удобство использования конструктора.

## 21.4. Система Eloquent

Фреймворк Laravel также поддерживает систему объектно-реляционного отображения Eloquent. В этой системе каждая таблица имеет соответствующий класс-модель, который используется для работы с этой таблицей. Модели позволяют выбирать данные из таблиц, а также вставлять в них новые записи.

Работа с Eloquent начинается с создания модели. Все модели наследуют класс `Illuminate\Database\Eloquent\Model`. Для создания модели служит команда `make:model`:

```
php artisan make:model Article
```

Давайте создадим пример модели `Articles` на основе таблицы `articles`, которая была создана посредством этого SQL-оператора:

```
CREATE TABLE IF NOT EXISTS `articles` (  
  `id` varchar(255) NOT NULL,  
  `title` varchar(100) NOT NULL,  
  `content` varchar(255) NOT NULL,  
  `views` int(11) DEFAULT 0,  
  `active` int(11) DEFAULT 1,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Эту модель мы будем использовать для получения и хранения информации из таблицы `articles`:

```
<?php  
namespace App;  
use Illuminate\Database\Eloquent\Model;  
class Article extends Model  
{  
    //  
}
```

На текущий момент мы не указали, какую таблицу будем привязывать к нашей модели. Учитывая, что мы создали модель `Articles`, то Eloquent будет использовать таблицу `article`. Но у нас такой таблицы нет, а есть таблица `articles`, поэтому нужно или использовать модель `Articles`, или же определить свойство `$table`, позволяющее задать произвольную таблицу:

```
<?php  
namespace App;  
use Illuminate\Database\Eloquent\Model;
```

```
class Article extends Model
{
    //
    protected $table = 'articles';
}
```

Модель Eloquent предполагает, что первичный индекс будет называться 'id'. Если имя первичного ключа отличается от id, нужно использовать свойство `$primaryKey`, чтобы задать произвольное имя:

```
class Article extends Model
{
    //
    protected $table = 'articles';
    protected $primaryKey = 'id';
}
```

Наша таблица использует как раз имя 'id', поэтому определять поле `$primaryKey` надобности нет. Но Eloquent предполагает, что первичный ключ является инкрементным числом, у нас же это не так, поэтому нужно установить свойство `$incrementing` в `false`:

```
class Article extends Model
{
    //
    protected $table = 'articles';
    protected $incrementing = false;
    protected $keyType = 'string';
}
```

Мы также установили свойство `$keyType` в 'string', чтобы указать, что первичный индекс будет строкой, а не числом.

Кроме того, Eloquent ожидает наличия в ваших таблицах столбцов `created_at` и `updated_at` — в эти поля Eloquent будет записывать дату создания и дату обновления. Нам такого не нужно, поэтому мы установим свойство `$timestamps` класса модели в `false`:

```
class Article extends Model
{
    //
    protected $table = 'articles';
    protected $incrementing = false;
    protected $keyType = 'string';
    protected $timestamps = false;
}
```

Теперь, когда наша модель создана, вы можете начать получать данные из базы. Каждая модель представляет собой конструктор запросов, позволяющий удобно выполнять запросы к связанной таблице. Например, сейчас мы получим все записи:

```
<?php
use App\Article;
$articles = App\Article::all();
foreach ($articles as $item) {
    echo "<p>$item->title\n";
}
```

Конечно, в нашем случае лучше было бы назвать модель `Articles`, но нам нужно было продемонстрировать использование другого названия таблицы — отличного от названия по умолчанию.

Если нужно получить не все записи, тогда мы можем использовать ограничения. Синтаксис здесь такой же, как у обычного конструктора запросов:

```
$articles = App\Article::where('active', 1)
    ->orderBy('title', 'desc')
    ->limit(10)
    ->get();
```

Для обновления данных в модели используются методы `fresh()` и `refresh()`:

- ❑ `fresh()` — делает запрос в базу данных и возвращает такую же модель с новыми данными. Существующая модель не изменяется;
- ❑ `refresh()` — перезаписывает текущую модель новыми данными из базы. Все загруженные отношения также будут обновлены.

Попробуем записать данные из модели и рассмотрим, как можно добавить данные в модель:

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use App\Article;
use Illuminate\Http\Request;
class MyController extends Controller
{
    /**
     * Создание нового экземпляра статьи.
     *
     * @param Request $request
     * @return Response
     */
    public function store(Request $request)
    {
        // Создание нового экземпляра
        $newArticle= new Article;
        $newArticle->title = $request->input('title');
        $newArticle->content = $request->input('content');
        $newArticle->active = 0;
```

```
        // Сохраняем модель
        $newArticle->save();
    }
}
```

Мы здесь просто создаем новый объект нужного нам класса, затем задаем значения его полей, после чего вызываем метод `save()` для сохранения модели в базу данных.

### **ПРИМЕЧАНИЕ**

Отметки времени `created_at` и `updated_at` будут автоматически установлены при вызове `save()`, поэтому их не нужно задавать вручную.

А вот как можно изменить запись:

```
// Находит запись с заданным $id
$a = App\Article::find($articleId);
// Устанавливаем новый заголовок
$a->title = $title;
// Сохраняем изменения
$a->save();
```

Для массового изменения нужно использовать метод `update()`:

```
App\Article::where('active', 0)->update(['active' => 1]);
```

Этот оператор активирует все деактивированные ранее записи.

Для удаления записи сначала ее нужно найти — например, с помощью метода `find()`:

```
$a = App\Article::find($articleId);
$a->delete();
```

Здесь мы получили модель из БД, а затем вызвали метод `delete()`.

Как видите, использование моделей делает работу с базой данных еще удобнее. Какой способ для работы с базой данных использовать на практике: конструктор запросов или модели — решать только вам.



## **I. РАЗДЕЛ 6**

# **Инструменты для создания сложных проектов**

<b>Глава 22.</b>	Разработка собственного шаблонизатора
<b>Глава 23.</b>	Шаблонизатор Blade
<b>Глава 24.</b>	Объектно-ориентированное программирование
<b>Глава 25.</b>	Хранение данных в Cookies и сессиях
<b>Глава 26.</b>	Обработка исключений
<b>Глава 27.</b>	Контроль версий
<b>Глава 28.</b>	Тестирование PHP-сценариев



## ГЛАВА 22

# Разработка собственного шаблонизатора

### 22.1. Организация файлов и каталогов проекта

Основное отличие высшей лиги от всех предыдущих — это полный порядок. Порядок должен быть во всем: как в организации, так и в коде самих файлов. У вас в корневом каталоге уже лежит десятка три файлов (PHP, HTML — не важно)? А некоторые PHP-сценарии выводят с помощью `echo` HTML-код? Могу вас поздравить — у вас полный непорядок.

Прежде всего разнесем файлы по отдельным специализированным каталогам. Понимаю, что когда у вас 30 или даже более различных файлов, вы не знаете, за что взяться. Начнем с малого — с картинок. Все графические файлы выносим в отдельный каталог, например `images` (или `img` — сокращенно). Конечно, пути к картинкам изменятся, поэтому придется изменить несколько десятков тегов `IMG`, чтобы дизайн сохранил свою целостность.

После этого вынесем все HTML-файлы в каталог `html`. Но и это еще не все. HTML-файлы ведь содержат какую-то информацию. Поэтому целесообразно рассортировать их по содержанию. Например, статическую информацию, которая редко меняется («контакты», «о нас», «карта проезда» и т. д.), желательно поместить в подкаталог `static` (`html/static`). Если у вас есть какие-то обзоры, то для них создаем каталог `reviews`, для статей — каталог `articles` и т. д. Думаю, вы поняли принцип: главное, чтобы имена подкаталогов были значимыми. Ведь каталог можно назвать и просто `a` или `bbb`. Но через месяц вы не вспомните, что находится в этом каталоге, пока не откроете его. А вот когда каталог называется `articles` или `reviews`, однозначно можно сказать, что в нем лежит.

Аналогичным образом можно, кстати, рассортировать и каталог `images`. Изображения, имеющие отношение к дизайну сайта, пусть остаются в каталоге `images`, а вот картинки для других страниц — например, для статей или обзоров, желательно поместить в каталоги соответственно `images/articles` и `images/reviews`. Фотографии, понятно, помещаем в каталог `images/photos`.

Теперь самое сложное — рассортировать РНР-файлы. Можно, конечно, их оставить в корневом каталоге, если файлов немного. Но мне больше нравится, чтобы в корневом каталоге было как можно меньше файлов — например, только `index.php` и `config.php` (о нем поговорим позже). Все остальные файлы следует рассортировать так:

- ☐ если файл подключается на каком-то этапе работы сценария, тогда помещаем его в каталог `includes` (это подкаталог корневого каталога);
- ☐ если вы написали модульный движок, и РНР-файл фактически является модулем, тогда его нужно поместить в каталог `modules`;
- ☐ если файл относится к панели администрирования, тогда его помещаем в каталог `admin` или `adm` — кому как больше нравится;
- ☐ остальные файлы тоже группируем по смыслу.

В результате у нас должна получиться структура каталогов, подобная изображенной на рис. 22.1.

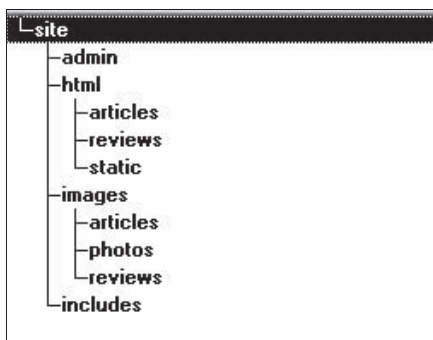


Рис. 22.1. Примерная структура каталогов

Настало время поговорить об именах файлов. Больше всего нас сейчас должны интересовать наши статьи и обзоры. Существуют две методики присвоения имен файлам. Первая заключается в том, что имя должно отображать содержимое файла. Предположим, у нас есть статья об установке Windows 11 — тогда файл этой статьи можно назвать, например, так: `windows_11_install.html`.

Удобно? Конечно — взглянул на имя файла и понял, что в нем, даже не открывая его.

Вторая методика предлагает именовать файлы так:

<подраздел\_номер>

или так:

<подраздел\_дата\_номер>

Например, у нас есть подраздел «Статьи» (Articles). Тогда файлы в нем можно именовать так:

`articles_номер` или `article_номер`

или так:

`article_дата_номер`

В этой методике тоже есть свои преимущества:

- ❑ эффект педантичности и порядка — ведь все файлы названы строго определенным способом;
- ❑ можно легко определить дату опубликования статьи — некоторым людям проще запомнить числа, например дату, чем символьное название. Вы помните, что статья приблизительно была опубликована в феврале 2022 г., поэтому вам нужно просмотреть файлы `article_DD_02_2022_N`, где `DD` — число, `N` — порядковый номер статьи (ведь в один и тот же день можно опубликовать несколько статей).

Лично мне проще ориентироваться, когда название файла отображает его содержание, поэтому для своего сайта я выбрал именно такую методику. Она не всегда оптимальна, но в моем случае вполне приемлема. Однако если вы пишете серьезный проект — например, систему управления контентом сайта (CMS), тогда такой подход отпадает. Рассмотрим ситуацию. Заходит на сайт пользователь и хочет опубликовать статью с названием «Установка Windows 11, или Проверяем, как будет тормозить система на 1 Гбайт оперативки». Вот теперь подумайте, какой алгоритм придется написать, чтобы это название было трансформировано в `windows_11_install`. Правильно. Очень сложный. Я не говорю, что эта задача невыполнима, но она не стоит потраченного на нее времени. Ведь статья будет выводиться одинаково вне зависимости от того, как назван файл. Поэтому здесь целесообразнее будет такой формат имени файла для статьи:

`article_DD_MM_GГГГ_Номер`

Согласитесь, такое имя файла сформировать намного проще. Можно и еще больше упростить задачу, назвав файлы статей так:

`article_номер`

Хотя, скорее всего, вы свои статьи будете хранить в базе данных, поэтому проблема с именами файлов для статей отпадет. Но останется проблема с именами файлов изображений, которые привязаны к статьям. Ведь картинку в базу данных не поместишь. Точнее, поместить-то можно (есть тип поля `BLOB` — не забываем), но тогда нужно будет придумать, как их оттуда извлекать при выводе той или иной статьи. Намного проще в таблице базы данных хранить HTML-код статьи, а графические файлы — отдельно на диске, в каталоге `/images/articles`.

Как именовать картинки? Предлагаю следующий способ:

`ID-статьи_номер_картинки_в_статье<расширение>`

Предположим, что в базе есть статья с номером 25, в которой три картинки формата JPEG. Тогда файлы картинок должны называться так: `25_01.jpg`, `25_02.jpg`, `25_03.jpg`. Просто и удобно.

## 22.2. Выносим параметры в отдельный файл

Наверняка в ваших PHP-сценариях будут какие-то константы, влияющие на работу всего сценария, — например, имя сервера MySQL, имя пользователя, пароль, количество сообщений на странице и т. д. Все эти параметры целесообразно вынести в отдельный файл, который обычно называется `config.php`.

Делается это из следующих соображений:

- ❑ если у вас два или более РНР-файлов задействуют эти параметры, то изменение параметров будет производиться проще, ведь параметр нужно будет изменить только один раз — в конфигурационном файле, а не в каждом файле проекта;
- ❑ даже если эти параметры необходимы только в одном файле, все равно желательно их вынести в `config.php`, чтобы пользователю, который потом скачает и будет использовать ваш проект, не пришлось копаться в вашем коде в поисках нужных параметров.

Рассмотрим небольшой пример файла `config.php` (листинг 22.1).

#### Листинг 22.1. Пример файла `config.php`

```
<?php

// имя сервера MySQL
$SERVER = 'localhost';

// имя пользователя и пароль
$USER = 'user';
$PASSWORD = '12345678';

// база данных
$DB = 'cms';

// число сообщений на странице (для страничного вывода)
$messages = 10;

// e-mail администратора:
$mailto = 'webmaster@example.com';

?>
```

#### **ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `config.php`), представленный в каталоге *Glava\_22* сопровождающего книгу электронного архива (см. приложение 4).

## 22.3. Подключение дополнительных файлов

### 22.3.1. Инструкции *include* и *require*

Инструкции `include`, `require`, `include_once`, `require_once` предназначены для подключения дополнительных файлов — например, того же файла конфигурации `config.php`.

Обычно дополнительные РНР-файлы подключаются с помощью инструкции `include`:

```
include "имя_файла.php";
```

Мы все привыкли к этой инструкции. Разберемся, как ее обрабатывает интерпретатор. Сначала PHP преобразует сценарий во внутреннее представление, именно поэтому его называют компилирующим интерпретатором или интерпретирующим компилятором, т. е. PHP — это нечто среднее между интерпретатором и компилятором. После преобразования PHP начинает выполнять код, строка за строкой. Если ему встретится инструкция `include`, то он загружает файл, указанный в этой инструкции, и полностью переключается на его выполнение — этот файл также преобразуется во внутреннее представление, а потом поэтапно выполняется.

Инструкция `require` работает иначе — файл, подключенный с ее помощью, транслируется до выполнения сценария. Иными словами, на момент выполнения сценария файл уже оттранслирован. Отсюда вывод — если использовать `require`, то ваши сценарии будут работать быстрее. Поэтому инструкция `require` предпочтительнее.

### 22.3.2. Альтернативный способ подключения сценариев

Нужно отметить, что у инструкций `include` и `require` имеется определенная особенность — подключенный файл становится неотъемлемой частью сценария. Предположим, у нас есть основной файл `index.php` (листинг 22.2) и дополнительный — `connect.php` (листинг 22.3).

#### Листинг 22.2. Основной файл — `index.php`

```
<?php

// выводим заголовок
echo join('', file('header.html'));

// подключаемся к базе данных
include "connect.php";

// выводим футер
echo join('', file('footer.html'));

?>
```

#### Листинг 22.3. Дополнительный файл — `connect.php`

```
<?php

$mysqli = new mysqli("localhost", "root", "", "test");

if ($mysqli->connect_errno) {
    die($mysqli->connect_error);
}

?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовые варианты (файлы *index.php* и *connect.php*), представленные в каталоге *Glava\_22* сопровождающего книгу электронного архива (см. приложение 4).

Допустим, что произошла ошибка подключения к серверу баз данных. Будет выполнена инструкция `die()`, которая прервет выполнение не дополнительного, а основного сценария, поскольку дополнительный сценарий инструкцией `include` просто встраивается в основной. В результате мы увидим сообщение об ошибке, но не увидим нижнюю часть дизайна. Это нехорошо. Выход из этой ситуации найти можно. Есть даже два выхода: во-первых, можно изменить сценарий `connect.php`:

```
$mysqli = new mysqli("localhost", "root", "", "test");
```

```
if ($mysqli->connect_errno) {  
    echo $mysqli->connect_error;  
}
```

Другими словами, сделать так, чтобы не было инструкции `die()`. Во-вторых, можно подключать файлы (сценарии) с помощью инструкции `file_get_contents()`. Но только при подключении сценария, который не выполняет сервисных или подготовительных действий для основного сценария. Так что в нашем случае инструкцию `file_get_contents()` использовать нельзя — ведь сценарий `connect.php` как раз и подключает основной сценарий к базе данных.

А вот при подключении сценария, который не выполняет сервисных или подготовительных действий для основного сценария, а просто выводит информацию и завершает свою работу, можно обойтись инструкцией `file_get_contents()`. Пусть имеется сценарий `news.php`, выводящий новости (листинг 22.4).

**Листинг 22.4. Сценарий `news.php`**

```
<?php  
  
$mysqli = new mysqli("localhost", "root", "", "test");  
  
if ($mysqli->connect_errno) {  
    die($mysqli->connect_error);  
}  
  
$sql = 'select * from news';  
  
$sql = "update login set pswd = \"\$login\" where phone=\"\$phone\"";  
  
$r = $mysqli->query($sql);  
  
while ($row = $r->fetch_array())  
    echo "<p>$row[dt] $row[txt]";  
  
>>
```

В этом случае сценарий `news.php` можно подключить с помощью инструкции `file_get_contents()`:

```
echo file_get_contents('http://example.com/news.php');
```

#### **ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `news.php`), представленный в каталоге *Glava\_22* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание — нужно указывать URL файла, а не путь к файлу, иначе вместо результата работы будет выведен исходный код файла.

Еще раз повторюсь, что данный способ не подходит для сервисных сценариев, без включения кода которых в основной сценарий дальнейшее продолжение работы последнего невозможно.

### **22.3.3. Инструкции `include_once` и `require_once`**

Помимо инструкций `include` и `require`, существуют инструкции однократного включения: `include_once` и `require_once`, которые работают так же, как и обычные `include` и `require`, но перед подключением файла проверяют, не был ли он уже подключен. Дело в том, что, развивая большой проект, вы можете забыть, когда нужно подключать тот или иной файл, и ошибочно подключить какой-то РНР-файл несколько раз. Чтобы такого не произошло, рекомендуется применять инструкции `include_once` и `require_once`.

## **22.4. Шаблоны**

Давайте на минутку остановимся, чтобы задуматься, какую работу мы проделали. Во-первых, мы навели порядок с именами файлов и каталогов. Во-вторых, вынесли лишний код из РНР-файлов в отдельные файлы — так намного удобнее. Теперь настал черед вынести из наших сценариев HTML-код.

Даже если вы начинающий РНР-программист, то, скорее всего, будете хранить HTML-код отдельно от сценария в HTML-файле. Ведь так удобнее. А в нужном месте с помощью инструкции `file_get_contents()` можно прочитать содержимое HTML-файла и оператором `echo` вывести его в браузер. Но иногда встраивать HTML-код в сценарий все же приходится.

Предположим, в нашем сценарии есть следующие операторы:

```
echo "<p>Добро пожаловать <b>$user</b>";  
echo "<p>Дата вашего последнего визита: <b>$last</b>";  
echo "<p>Дата регистрации: <b>$reg_date</b>";  
echo "<p>Вами оставлено сообщений: <b>$messages</b>";  
echo "<p>Новых личных сообщений: <b>$personal</b>";
```

Другими словами, наш РНР-сценарий содержит HTML-код, который нельзя поместить в обычный HTML-файл, поскольку его нужно модифицировать и передавать в браузер.

Если вы сами планируете поддерживать свой сценарий, то вроде бы ничего страшного здесь нет. Но вы только представьте лицо дизайнера, не имеющего никакого отношения к PHP и к программированию вообще, когда он откроет PHP-файл, чтобы изменить какой-то HTML-код. Ведь настоящий сценарий — это не эти простые пять строк, а сотни строк PHP-кода, среди которого иногда мелькает HTML-код. Разбираться во всем этом сложно, к тому же велика вероятность ошибки. Поэтому следует вообще исключить HTML-код из PHP-файла. Для этого нам придется использовать шаблоны, вынося HTML-код в отдельные файлы, как это сделано для нашего HTML-кода, который вынесен в файл `welcome.tpl` (листинг 22.5).

#### Листинг 22.5. Файл `welcome.tpl`

```
<p>Добро пожаловать <b>{USER}</b>
<p>Дата вашего последнего визита: <b>{LAST}</b>
<p>Дата регистрации: <b>{REG_DATE}</b>
<p>Вами оставлено сообщений: <b>{MESSAGES}</b>
<p>Новых личных сообщений: <b>{PERSONAL}</b>
```

#### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл `welcome.tpl`), представленный в каталоге `Glava_22` сопровождающего книгу электронного архива (см. приложение 4).

Наверняка вы уже сталкивались с такими файлами (подобный формат часто используется в различных проектах, написанных на PHP и не только). Дизайнерам сайтов этот формат тоже знаком, поэтому особого труда с редактированием файла шаблона у них не будет.

Далее в этой главе мы напишем очень простой шаблонизатор, т. е. сценарий, позволяющий обрабатывать TPL-шаблоны. Однако возможностей нашего шаблонизатора вполне хватит для создания на его основе более серьезных проектов.

Итак, шаблон у нас уже есть. Напишем файл `index.php`, который будет работать с шаблоном, используя наш еще не созданный шаблонизатор (листинг 22.6).

#### Листинг 22.6. Файл `index.php`

```
<?php
// подключаем шаблонизатор
require "template.php";

// определяем переменные, которые нужно внедрить в HTML-код
$USER = "Дмитрий";
$LAST = "23.03.2021";
$REG_DATE = "23.03.2020";
$MESSAGES = 54;
$PERSONAL = 3;
```

```
// открываем шаблон
$tpl->get_tpl('welcome.tpl');

// устанавливаем переменные шаблона
$tpl->set_value('USER',$USER);
$tpl->set_value('LAST',$LAST);

$tpl->set_value('REG_DATE',$REG_DATE);

$tpl->set_value('MESSAGES',$MESSAGES);

$tpl->set_value('PERSONAL',$PERSONAL);

// запускаем парсинг шаблона
$tpl->tpl_parse();
// выводим HTML
echo $tpl->html;
?>
```

### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *22-6.php*), представленный в каталоге *Glava\_22* сопровождающего книгу электронного архива (см. приложение 4).

Вот теперь полный порядок — из PHP-файла исключен HTML-код. Для открытия шаблона применяется метод `get_tpl()`, а для установки значения шаблона — `set_value()`. Первый параметр метода — это имя переменной в шаблоне, а второй — значение этой переменной. Метод `tpl_parse` выполняет парсинг нашего шаблона. Все, что нам остается сделать, — это вывести уже готовый HTML-код (рис. 22.2).

Осталось рассмотреть только сам файл шаблонизатора `template.php` (листинг 22.7).



Добро пожаловать Дмитрий

Дата вашего последнего визита: 23.03.2021

Дата регистрации: 23.03.2020

Вам оставлено сообщений: 54

Новых личных сообщений: 3

Рис. 22.2. Шаблонизатор в действии

**Листинг 22.7. Шаблонизатор**

```
<?php
// класс шаблона
class template_class
{
    var $values      = array();      // переменные шаблона
    var $html;        // HTML-код

// функция загрузки шаблона
    function get_tpl($tpl_name)
    {
        if(empty($tpl_name) || !file_exists($tpl_name))
        {
            return false;
        }
        else
        {
            $this->html = join('',file($tpl_name));
        }
    }

// функция установки значения
    function set_value($key,$var)
    {
        $key = '{' . $key . '}';
        $this->values[$key] = $var;
    }

// парсинг шаблона
    function tpl_parse()
    {
        foreach($this->values as $find => $replace)
        {
            $this->html = str_replace($find, $replace, $this->html);
        }
    }
}

// экземпляр класса
$tpl = new template_class;
?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *template.php*), представленный в каталоге *Glava\_22* сопровождающего книгу электронного архива (см. *приложение 4*).

Как видите, наш шаблонизатор примитивен. Весь парсинг шаблона заключается в обычной замене строк вида {название\_переменной} переданными значениями. Но тем не менее нужно отметить, что шаблонизатор работает, а большего нам и не надо. Если вам потребуется больше свободы, то вам поможет следующая глава, в которой мы рассмотрим шаблонизатор Blade. Его возможности намного шире, но в то же время он значительно сложнее нашего.

Напоследок добавлю, что для поддержания порядка нужно создать каталог `templates` и поместить в него все TPL-файлы.



## ГЛАВА 23

# Шаблонизатор Blade

### 23.1. Введение в Blade

Шаблонизатор Blade предоставляет простой и лаконичный синтаксис, который понравится разработчикам и верстальщикам интерфейса пользователя. Blade является частью фреймворка Laravel, а т. к. в этой книге делается упор именно на этот фреймворк, то вполне логично рассмотрение именно шаблонизатора Blade, а не какого-либо другого.

Несмотря на то что Blade входит в состав Laravel, вы можете использовать его и отдельно — если по каким-то причинам Laravel вас не устраивает. Получить только сам шаблонизатор можно по ссылке: <https://github.com/EFTEC/BladeOne>.

При использовании Laravel представления, написанные с использованием Blade, хранятся в каталоге (и его подкаталогах) `resources/views`, и их файлы имеют расширение `blade.php`. Именно так фреймворк «понимает», что представление нужно передать на обработку шаблонизатору Blade, а не интерпретатору PHP.

Шаблон Blade — это смесь из HTML-кода, инструкций Blade и инструкций PHP. С HTML-кодом все понятно. Инструкции PHP, как обычно, заключаются в теги `<?php ?>`, и внутри них вы можете написать практически любой PHP-код.

Инструкции (они же *директивы*) Blade начинаются с символа `@` — например: `@if`, `@foreach` и т. д. Также особое значение имеют фигурные скобки — в них заключается название переменной, значение которой нужно отобразить в представлении.

Рассмотрим для примера небольшой фрагмент кода:

```
<h1>{{ $title }}</h1>
@forelse ($posts as $item)
    <p><a href="/blog/{{ $item->id }}">{{ $item->title }}</a></p>
@empty
    Здесь пусто
@endforelse
```

Здесь мы сначала выводим заголовок блога (переменная `$title`, заключенная в двойные фигурные скобки), а затем в цикле `@forelse` — список записей блога.

Если записей нет, мы выводим сообщение **Здесь пусто**. Как бы вы ни старались, у вас не получится на чистом PHP написать такой лаконичный код.

## 23.2. Вывод значений скалярных переменных

*Скалярная переменная* — это переменная, содержащая одно значение. Для отображения значений таких переменных подойдут фигурные скобки. Для отображения массивов, а также других итерируемых объектов применяются циклы `@foreach`, `@forelse` и другие, внутри тела которых вы также будете использовать фигурные скобки, но для отображения текущего элемента итерируемого объекта.

Итак, пусть наш контроллер передает заголовок страницы:

```
return view('frontend/page', [  
    'title'=>$title,  
]);
```

Как мы его можем отобразить? Синтаксис фигурных скобок следующий:

```
{{ $переменная }}
```

Этот синтаксис аналогичен PHP-варианту:

```
<?= $переменная?>
```

По умолчанию все символы значения переменной экранируются функцией `htmlentities()` для защиты от вставки вредоносного кода. Это означает, что если ваша переменная содержит HTML-код (например, контент поста блога), то вместо форматированного текста вы увидите HTML-код — а это не то, что нам нужно.

Если вам надо вывести значение переменной как есть, используйте следующий синтаксис:

```
{!! $переменная !!}
```

Пример:

```
Отображаем заголовок {{ $title }}  
Заголовок без htmlentities() {!! $title !!}
```

## 23.3. Директивы Blade

### 23.3.1. Директива `@if`

Директива `@if` похожа на одноименный оператор PHP и работает она таким же образом. Пример использования этой директивы:

```
@if (count($posts) == 1)  
    // Выводим сразу статью  
@elseif (count($posts) == 0)  
    Блог пуст
```

```
@else
    // Выводим список записей с помощью @foreach
@endif
```

Обратите внимание: поскольку нет фигурных скобок, не забудьте использовать директиву `@endif`, чтобы сказать шаблонизатору, что текущий блок `@if` закончен. Внутри блока `@if` вы можете использовать директивы: `@elseif` и `@else` — их назначение такое же, как у одноименных инструкций PHP.

### 23.3.2. Директивы `@for`, `@foreach` и `@while`

Эти директивы представляют функциональность, аналогичную одноименным циклам в PHP. Начнем с цикла со счетчиком. Синтаксис, а точнее — назначение параметров — такое же как в PHP:

```
@for ($i = 0; $i < 100; $i++)
    {{ $i }}<br>
@endfor
```

Цикл `@for` вы будете в шаблонах использовать редко, чего не скажешь о цикле `@foreach`, который будет применяться для отображения каждого объекта из базы данных:

```
@foreach ($posts as $item)
<div class="product">
    <div class="product-title">{{ $item->title }}</div>
    <div class="product-image"></div>
    <div class="product-info">{{ $item->shortinfo }}</div>
</div>
@endforeach
```

А вот пример цикла `@while`:

```
@while ($item = array_pop($posts))
    {{ $item->title }}<br>
@endwhile
```

### 23.3.3. Директивы `@forelse` и `@endforelse`

Конструкция `@forelse` похожа на `@foreach`, но она будет выполнена даже, если итерируемый объект пуст. Пример:

```
@forelse ($posts as $item)
<div class="product">
    <div class="product-title">{{ $item->title }}</div>
    <div class="product-image"></div>
    <div class="product-info">{{ $item->shortinfo }}</div>
</div>
@empty
    Здесь пусто
@endforelse
```

В циклах `@foreach` и `@forelse` используется переменная `$loop`, недоступная в цикле `foreach` в PHP. Если она задействована в циклах `@foreach` и `@forelse`, то возвращает объект `stdClass` со следующими свойствами:

- ☐ `count` — количество элементов в цикле;
- ☐ `index` — отсчитанный от 0 индекс текущего элемента (нумерация начинается с 0);
- ☐ `iteration` — отсчитанный от 1 индекс текущего элемента (счетчик итерации — нумерация начинается с 1);
- ☐ `remaining` — количество элементов, оставшихся в цикле;
- ☐ `first` — логическое значение, указывающее, является ли текущий элемент первым в списке;
- ☐ `last` — логическое значение, указывающее, является ли текущий элемент последним в списке;
- ☐ `depth` — количество уровней в этом цикле: 1 — для цикла, 2 — для цикла внутри цикла и т. д.;
- ☐ `parent` — ссылка на переменную `$loop` в родительском цикле.

Вот пример использования переменной `$loop`:

```
<ul>
@foreach ($posts as $item)
  <li>{{ $loop->iteration }}: {{ $item->title }}
  @if ($item->hasChildren())
    <ul>
      @foreach ($item->children() as $child)
        <li>{{ $loop->parent->iteration }}
          .{{ $loop->iteration }}:
          {{ $child->title }}</li>
      @endforeach
    </ul>
  @endif
</li>
@endforeach
</ul>
```

## 23.4. Включение представлений.

### Директива `@include`

Разработчик может включать одни представления в другие. Например, вы можете создать представления `header.blade.php` и `footer.blade.php`, представляющие заголовок и футер вашего сайта, и вынести в них код, отвечающий за формирование заголовка и подвала страницы. Тогда вам не придется дублировать этот код в каждом представлении. Такая возможность часто используется и для подключения таблиц стилей CSS- и JS-скриптов.

В простейшем случае включение представления выглядит так:

```
@include('frontend/header')
```

При желании вы можете передавать во включаемое представление параметры. Рассмотрим небольшой пример:

```
<!-- resources/views/main.blade.php -->
@include('header', ['text' => 'Hello, world!'])

<!-- resources/views/header.blade.php -->
<h1>{{ $text }}</h1>
```

Здесь в представление `header` мы передали значение переменной `$text`, которая используется в этом представлении в качестве содержимого заголовка `<h1>`.

Blade позволяет включать представления по условиям. Для этого служат директивы `@includeIf`, `@includeWhen`, `@includeFirst`:

```
<!-- Включить представление, если оно существует -->
@includeIf('main.sidebar', ['data' => 'some_data'])

<!-- Включить представление, если определенная переменная равна true -->
@includeWhen($order->isPay(), 'thank-you', ['sum' => $order->sum])

<!-- Включить первое представление из массива представлений -->
@includeFirst(['header1', 'header2'], ['data' => 'test'])
```

Инструкция `@each` пригодится в случае, если нужно перебрать массив, коллекцию и включить часть для каждого элемента. Представим, что у нас есть боковая панель и нам надо включить несколько модулей со своим названием:

```
<!-- resources/views/sidebar.blade.php -->
<div class="sidebar">
    @each('bar.modules', $modules, 'module', 'bar.empty-mod')
</div>

<!-- resources/views/bar/module.blade.php -->
<div class="sidebar-module">
    <h1>{{ $module-->title }}</h1>
</div>

<!-- resources/views/bar/empty-mod.blade.php -->
<div class="sidebar-module">
    Empty module
</div>
```

Первый параметр `@each` задает имя составляющей представления, второй — массив или коллекцию для итерации, третий — название переменной, под которым каждый элемент (в нашем случае каждый элемент в массиве `$modules`) будет передан представлению. Четвертый параметр (необязательный) — представление, показы-

вающее, является ли массив или коллекция пустыми (здесь можно передать, строку, которая будет использоваться в качестве шаблона).

## 23.5. Директива `@csrf`

В каждую форму своего приложения вы должны включать скрытое поле токена CSRF, чтобы посредник защиты от CSRF мог провалидировать запрос. Для генерации токена служит директива `@csrf`:

```
<form method="POST" action="/action">  
@csrf
```

```
...
```

```
</form>
```

\* \* \*

Мы рассмотрели далеко не все возможности Blade. Заинтересовавшимся читателям рекомендуем ссылку на документацию по Blade: <https://laravel.ru/docs/8.x/blade>.



## ГЛАВА 24

# Объектно-ориентированное программирование

### 24.1. Основы ООП

Четкого определения объектно-ориентированного программирования (далее — ООП) нет даже в Википедии. Кто-то говорит, что это философия программирования. Можно соглашаться с этим, можно не соглашаться. На мой же взгляд, ООП — это способ разработки программ. Может, вы помните из истории вычислительной техники, что первые программы вводились на перфокартах. Так вот, это тоже способ написания программ. Затем перфокарты канули в Лету, обучаться программированию стало проще, поскольку исчезла необходимость привязки к аппаратным средствам, а сам процесс разработки программ стал существенно быстрее.

Но очень скоро программы на языках высокого уровня разрослись так, что стали совсем нечитабельными из-за отсутствия в них четко выделенной структуры кода. Нужно было что-то делать. И появилось так называемое *процедурное программирование* — это тоже способ написания программ, когда отдельные части программы объединяются в подпрограммы: процедуры и функции. В чем преимущество процедурного программирования? Предположим, что нам нужно написать программу, которая начисляет зарплату (и все отчисления) для трех сотрудников маленькой фирмы. Ориентируясь на процедурное программирование, расчет зарплаты можно организовать в виде функции (или процедуры) и вызывать ее при необходимости. Если бы не было процедур и функций, то код расчета заработной платы пришлось бы повторять три раза — для каждого сотрудника. В этом тривиальном случае, если быть внимательным, ошибку допустить практически невозможно, но ведь на практике задачи ставятся намного сложнее, а чем сложнее программа, тем больше вероятность логической ошибки.

Процедурное программирование существенно облегчило жизнь программистам, но опять-таки только на определенное время. Со временем снова проявилась та же проблема — длина исходного кода. И новым способом разработки программ, учитывающим это обстоятельство, стало объектно-ориентированное программирование (ООП).

Базовым понятием ООП является *класс*. Класс объединяет в себе код — *методы* класса и данные — *свойства* класса. Класс — это как бы тип переменной. Переменная этого типа класса называется *экземпляром класса* (или, проще, *объектом*).

И свойства класса, и методы класса называются *членами класса*. Механизм объединения в единое целое данных и методов для их обработки называется *инкапсуляцией*.

В ОПП кроме инкапсуляции часто используется *полиморфизм*. Возможно, вам знакомы функции `abs()`, `fabs()` и `labs()`. Все эти три функции — различные варианты функции `abs`, просто предназначены они для работы с разными типами данных. Алгоритм функций одинаков, но различны типы входящих и выходящих данных. Так вот, ООП позволяет создать три различных варианта функции, но объединить их под одним именем. В результате у нас будет одна функция: `abs()`, которую можно использовать для всех трех типов данных. Такой полиморфизм значительно упрощает написание сложных программ.

В ООП объект может наследовать свойства другого объекта. Не нужно думать, что *наследование* — это копирование объекта: при копировании создается точная копия исходного объекта, а при наследовании один объект дополняется членами другого объекта.

## 24.2. Классы и объекты

Итак, класс — это тип переменной, а экземпляр класса (объект) — переменная этого типа. Создать класс просто:

```
class имя_класса {  
    // члены класса  
}
```

Напишем небольшой класс `Message`, который будет использоваться для отправки сообщений:

```
<?php  
  
class Message {  
  
    // свойства класса  
    var $text;  
    var $subject;  
    var $to;  
  
    // методы класса  
    function load_text($filename) {  
  
        $this->text = join('',file($filename));  
    }  
  
    function send() {  
        mail($this->$to, $this->subject, $this->text);  
    }  
}
```

Наш класс довольно прост, но вполне работоспособен. Свойство `$text` — это текст сообщения, `$subject` — тема, `$to` — получатель. Вы можете присвоить текст явно, а можете загрузить его из файла с помощью метода `load_text()`. Для отправки используется метод `send()`. Рассмотрим пример кода, реализующего отправку сообщения с помощью созданного нами класса:

```
// создаем экземпляр класса
$mes = new Message;

$mes->to = 'user@example.com';           // получатель
$mes->subject = 'Hi!';                   // тема
$mes->load_text('message.txt');          // загружаем текст
// текст можно указать явно:
// $mes->text = 'Привет! Как дела?';

$mes->send();                            // отправляем сообщение
```

Полный пример кода приведен в листинге 24.1.

#### Листинг 24.1. Класс и пример кода (полный листинг)

```
<?php

class Message {

    // свойства класса
    var $text;
    var $subject;
    var $to;

    // методы класса
    function load_text($filename) {
        $this->text = join('',file($filename));
    }

    function send() {
        mail($this->$to, $this->subject, $this->text);
    }
}

$mes = new Message;

$mes->to = 'user@example.com';
$mes->subject = 'Hi!';
$mes->load_text('message.txt');
$mes->send();

?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *24-1.php*), представленный в каталоге *Glava\_24* сопровождающего книгу электронного архива (см. приложение 4).

Обратите внимание на то, как осуществляется доступ к свойствам класса — с помощью оператора `->`. Доступ к члену класса внутри класса осуществляется с помощью оператора `->` и указателя `$this`. Указатель `$this` служит для доступа к членам класса, т. е. может использоваться как для свойств, так и для методов класса.

Ключевое слово `class` позволяет получить имя класса. Например:

```
// объявление пространства имен должно быть первым оператором
// в программе (см. разд. 24.5.10)
namespace A;

...
class B {
}

echo B::class;           // выводит A\B
```

## 24.3. Конструкторы и деструкторы класса

*Конструктор* — это специальный метод, выполняющий инициализацию объекта. Он, например, устанавливает нужные соединения, открывает файлы, присваивает свойствам объекта начальные значения. Одним словом, подготавливает объект к нормальной работе. Имя метода конструктора совпадает с именем класса. Для нашего класса `Message` конструктор будет выглядеть так:

```
function Message() {
// имя пользователя по умолчанию
$this->to = 'user@example.com';
// тема по умолчанию
$this->subject = 'Привет!';
// текст по умолчанию
$this->text = 'Привет! Как дела?';
}
```

Конструктор вызывается сразу при объявлении объекта класса:

```
$mes = new Message;
```

Поскольку конструктор установил свойства объекта по умолчанию, то для отправки письма пользователю по умолчанию (`user@example.com`) нужно только вызвать метод `send()`:

```
$mes->send();
```

В PHP можно указывать для конструкторов единое имя: `__construct()`:

```
function __construct() {
// имя пользователя по умолчанию
$this->to = 'user@example.com';
}
```

```
// тема по умолчанию
$this->subject = 'Привет!';
// текст по умолчанию
$this->text = 'Привет! Как дела?';
}
```

Кроме конструкторов в ООП существуют еще и *деструкторы*. Деструктор — это метод класса, который выполняет деинициализацию объекта, — например, освобождение используемых ресурсов, закрытие соединений и т. д. В РНР 4 деструкторы не поддерживаются, а все используемые объектом ресурсы освобождаются автоматически. С одной стороны, удобно, с другой — не очень правильно по отношению к ООП, поэтому в РНР поддержка деструкторов включена. Деструктор класса называется `__destruct()`:

```
function __destruct() {
    print "Destroying " . $this->name . "\n";
}
```

Свойство `name` мы не объявляли, оно объявляется по умолчанию (автоматически) и должно содержать имя объекта, но это имя мы должны присвоить сами:

```
$mes = new Message;
$mes->name = 'Message object';
```

В РНР 8 можно объявлять свойства в конструкторе. Например, если в РНР 7 у вас был следующий код:

```
class Point {
    public float $x;
    public float $y;
    public float $z;

    public function __construct(
        float $x = 0.0,
        float $y = 0.0,
        float $z = 0.0
    ) {
        $this->x = $x;
        $this->y = $y;
        $this->z = $z;
    }
}
```

то в РНР 8 его можно переписать так:

```
class Point {
    public function __construct(
        public float $x = 0.0,
        public float $y = 0.0,
        public float $z = 0.0,
    ) {}
}
```

Объявление свойств в конструкторе позволяет сделать ваш код компактнее, поскольку вы можете не только объявить, но и сразу инициализировать свойства.

## 24.4. Наследование классов. Полиморфизм

Как уже отмечалось, наследование — это операция расширения класса путем создания нового класса на базе существующего и добавления к нему новых свойств и методов. Наследование определяется с помощью служебного слова `extends`, после которого следует имя *родительского* класса (того класса, который будет взят за основу при создании нового — *дочернего* класса):

```
class Имя_дочернего_класса extends Имя_родительского_класса {  
  
}
```

Рассмотрим небольшой пример наследования:

```
class ParentClass {  
function F1() {  
    echo "Родительский класс";  
}  
}  
  
class ChildClass extends ParentClass {  
  
function F1() {  
    echo "Дочерний класс";  
}  
function F2() {  
    echo "Дочерний класс. Функция F2";  
}  
}  
  
$p = new ParentClass;  
$c = new ChildClass;  
  
$p->F1();    // выведет "Родительский класс"  
$c->F1();    // выведет "Дочерний класс"  
$c->F2();    // выведет "Дочерний класс. Функция F2"
```

Дочерний класс `ChildClass` расширяет родительский класс `ParentClass`, переопределяя метод `F1()` и добавляя новый метод `F2()`. Метода `F2()` в родительском классе не было — он появился только в дочернем классе `ChildClass`.

Обратите внимание на то, что дочерний класс переопределил метод `F1()` родительского класса. Это и есть пример полиморфизма, т. е. перезагрузки методов и обычных функций.

## 24.5. Область видимости членов класса

В PHP вы можете определить область для переменных — членов класса. Допускается использовать как частные (`private`), так и защищенные (`protected`) переменные. Защищенные свойства доступны только методам своего класса и производных (дочерних) классов — установить значение защищенного свойства непосредственно нельзя. Частные (`private`) свойства доступны только методам своего класса — установить значения этих свойств ни непосредственно, ни через дочерний класс невозможно.

Вспомним наш класс `Message`. Объявим его свойства так (заодно можно установить значения по умолчанию):

```
private $to = "user@example.com";
protected $subject = "Привет!";
protected $text = "Как дела?";
```

После этого мы не сможем уже обратиться к этим свойствам так, как мы это делали раньше:

```
$mes->to = "user@example.com";
```

Для установки приватных (частных) и защищенных свойств нам придется писать собственные методы. Напишем метод `set_attr()`, который устанавливает все наши свойства:

```
function set_attr($to, $subj, $text) {

    $this->to = $to;
    $this->subject = $subject;
    $this->text = $text;

}
```

Свойства `$subject` и `$text` будут доступны дочерним классам, а вот свойство `$to` — частное, оно не будет доступно производным классам.

Аналогично вы можете указать область видимости и для методов класса. При этом перед служебным словом `function` нужно поставить модификаторы `private` и `protected`:

```
private function send() { };
```

или

```
protected function send() { };
```

Если модификаторы `private` и `protected` не указаны, члены класса считаются публичными (`public`) — т. е. общедоступными.

## 24.6. Абстрактные классы и методы

PHP поддерживает абстрактные классы и методы. *Абстрактные методы* — это всего лишь объявления методов, не содержащие реализации метода. *Абстрактный класс* — это класс, поддерживающий абстрактные методы. Если вы хотите использовать абстрактные методы, вы должны объявить абстрактный класс. В обычном классе вы не можете объявить абстрактные методы. Абстрактные методы и классы объявляются с помощью служебного слова `abstract`:

```
abstract class AbstractClass {  
    abstract public function test();  
}
```

Вы не можете создать экземпляр (объект) абстрактного класса. Можно создать обычный класс на базе абстрактного, объявив его дочерним, а потом уже создать объект класса. Но вы должны реализовать каждый метод абстрактного класса в дочернем классе.

```
class MyClass extends AbstractClass {  
    // реализация абстрактного метода test  
    public function test() {  
        echo "проверка";  
    }  
}  
  
$o = new MyClass;  
$o->test();
```

## 24.7. Служебное слово *final*

Служебное слово `final` используется для описания методов класса, которые не могут быть переопределены в классе-потомке (в дочернем классе):

```
class MyClass {  
    final function test() {  
        echo "проверка";  
    }  
}
```

Служебное слово `final` можно использовать и при описании класса. Тогда этот класс нельзя будет наследовать:

```
final class MyClass {  
    // определение класса  
}
```

Если вы попытаетесь наследовать класс `MyClass`, то получите ошибку времени выполнения.

## 24.8. Клонирование объектов

Мы уже говорили о наследовании классов. Теперь поговорим о *клонировании* объектов, а именно о создании копии объекта. РНР 4 просто осуществлял побайтовое копирование объекта, создавая точные копии всех его свойств. Начиная с РНР 5, появилась возможность управления клонированием объекта.

Управляет клонированием объекта специальный метод — `__clone()`. Этот метод есть у каждого класса по умолчанию, он просто побайтно копирует все свойства и методы объекта. Клонирование осуществляется с помощью оператора `clone`:

```
$obj2 = clone $obj1;
```

В руководстве по РНР 5 упоминается, что можно клонировать объект вызовом метода `__clone()`:

```
$obj2 = obj1->__clone();
```

Однако последние версии РНР завершают выполнение такого сценария с рекомендацией использовать оператор `clone`:

```
Fatal error:  Cannot call __clone() method on objects — use 'clone $obj'
instead in D:\PHP\class.php on line 23
```

Рассмотрим листинг 24.2, в котором приведен пример клонирования класса.

### Листинг 24.2. Пример клонирования класса

```
<?php
class MyClass {
    public $id = 0;           // значение по умолчанию

    function MyClass() {
        // конструктор присваивает свойству id значение 10
        $this->id = 10;
    }
    function __clone() {
        // при клонировании будет скопировано имя объекта
        // и присвоено свойству id значение 0
        $this->name = $that->name;
        $this->id = 0;
    }
}

$obj = new MyClass();        // создаем класс

print $obj->id . "\n";        // выведет 10

$obj = clone $obj;           // клонируем класс
```

```
// выведет 0, потому что __clone() присвоит свойству id ноль
print $obj->id . "\n";
?>
```

### ПРИМЕЧАНИЕ

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *24-2.php*), представленный в каталоге *Glava\_24* сопровождающего книгу электронного архива (см. *приложение 4*).

## 24.9. Константы — члены класса

В PHP 5/8 вы можете использовать константы — члены класса, причем обратиться к константе допускается без объявления объекта класса. Доступ к константе можно получить так:

```
имя_класса::имя_константы
```

Вам не нужно указывать знак доллара (\$) ни для имени класса, ни для имени константы. Вот небольшой пример:

```
<?php
class Foo {
    const constant = "constant";
    public $id = 0;
}

echo "Константа = " . Foo::constant . "\n";
?>
```

## 24.10. Статические члены класса

Статические члены класса можно вызывать за пределами контекста объекта — без объявления объекта с помощью оператора ::, как мы поступали с константой. Для объявления статических членов класса используется служебное слово `static`:

```
class Stat {
    static $static_member = 5;
    public static function aStaticMethod() {
        // ...
    }
}

// обращаемся к статическим членам класса без объявления объекта
Stat::aStaticMethod();
echo Stat::$static_member;
```

Из PHP 7 была удалена поддержка статических вызовов нестатических методов. Для примера рассмотрим такой фрагмент кода:

```
class A {
    public function test() { var_dump($this); }
}

// Класс B НЕ расширяет A
class B {
    public function callNonStaticMethodOfA() { A::test(); }
}

(new B)->callNonStaticMethodOfA();
```

Этот код в PHP 7/8 работать не будет, поскольку нестатический метод `A::test()` не может быть вызван статично.

## 24.11. Оператор *instanceof*

Оператор `instanceof` позволяет определить, является ли тот или иной объект экземпляром определенного класса. Использовать `instanceof` нужно так:

```
if (объект instanceof класс) ...
```

Например:

```
$mes = new Message;

if ($mes instanceof Message)
echo "Объект mes является экземпляром класса Message";
```

## 24.12. Итераторы

Оператор цикла `foreach` можно использовать для перебора всех элементов ассоциативного массива примерно так:

```
foreach($assoc_array as $key => $value) { ... };
```

В PHP этот оператор можно применить и для итерации по свойствам объекта. Мы можем перебрать и обработать все свойства объекта в цикле так:

```
foreach (объект as переменная_имя_свойства => переменная_значение_свойства) {};
```

Например:

```
$mes = new Message;
foreach ($mes as $prop_name => $prop_value)
    echo "$prop_name = $prop_value\n";
```

## 24.13. Пространства имен

### 24.13.1. Общая концепция

Одна из самых полезных функций современных версий PHP — пространства имен (namespaces). Впервые пространства имен появились в PHP 5.3, но с той поры прошло уже много лет, а некоторые программисты все еще пренебрегают использованием пространств имен в своих приложениях.

Пространства имен позволяют организовать PHP-код в виртуальную иерархию, сравнимую со структурой каталогов вашей операционной системы. Каждый современный компонент или фреймворк PHP организует свой код в собственное пространство имен, благодаря чему код не конфликтует с другим кодом.

Давайте не будем углубляться теорию, а лучше перейдем сразу к практике и посмотрим, как реальные фреймворки PHP используют пространство имен, — на примере фреймворка Symfony.

Чтобы вам не загружать код Symfony, вы можете просматривать его онлайн на странице: <https://github.com/symfony/http-foundation>. В состав фреймворка входит популярный компонент `symfony/httpfoundation`, который управляет запросами и ответами HTTP. Компонент `symfony/httpfoundation` использует общие имена PHP-классов вроде `Request`, `Response` и `Cookie`. Не сложно предположить, что другие компоненты тоже могут использовать такие же имена классов. Как PHP поймет, какой код нужно выполнить, если имена классов одинаковые?

Для этого как раз и служат пространства имен. Мы можем поместить компонент `symfony/httpfoundation` в «песочницу» под названием пространства имен. Просмотрите исходный код файла `Request.php`:

```
namespace Symfony\Component\HttpFoundation;
use Symfony\Component\HttpFoundation\Exception\ConflictingHeadersException;
use Symfony\Component\HttpFoundation\Exception\SuspiciousOperationException;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
/**
 * Request represents an HTTP request.
```

Первая строка здесь определяет пространство имен `Symfony\Component\HttpFoundation`. Эта строка должна быть всегда указана первой (если не считать комментариев) после открывающегося тега `<?php`. Объявление пространства имен делает следующее:

1. Мы теперь знаем, что класс `Request` «живет» внутри пространства имен разработчика Symfony.
2. Мы знаем, что класс `Request` относится к подпространству имен `Component`.
3. Наконец, есть еще одно подпространство с именем `HttpFoundation`, к которому и относится наш класс `Request`.

#### ПРИМЕЧАНИЕ

Подпространства при объявлении пространства имен разделяются символом «\». Если вы привыкли к Linux, это несколько необычно, но придется привыкнуть.

Теперь, надеюсь, вы понимаете, почему важны пространства имен. Они изолируют код одного разработчика от кода другого разработчика. Это краеугольный камень современной экосистемы компонентов PHP.

## 24.13.2. Объявление пространства имен

Рассмотрим пример использования пространства имен. Пространство имен объявляется инструкцией `namespace`:

```
namespace MySpace;
```

Как уже отмечалось, инструкция `namespace` должна быть первым оператором в вашей программе, т. е. ее нужно вызывать сразу после `<?php`:

```
<?php
namespace MySpace;
```

Затем можно объявить какой-нибудь класс. Я объявил класс `XMLWriter`, который уже существует в PHP, — налицо коллизия имен, PHP должен сообщить об этом и прервать выполнение сценария:

```
class XMLWriter
{
    // члены и метода класса
}
```

То есть старые версии PHP сообщат:

```
Cannot redeclare class xmlwriter
```

Однако, начиная с версии 5.3, все проходит гладко, — только при использовании нашего класса нужно указывать наше же пространство имен:

```
$xml = new MySpace::XMLWriter();
```

При желании или необходимости можно объявить подпространство, как и в предыдущем примере:

```
namespace MySpace\SubSpace;
```

Не нужно объявлять все классы пространства имен в одном PHP-файле. Вы можете объявлять каждый класс в отдельном файле, только в начало этого файла не забудьте добавить объявление пространства имен. Это позволяет объявлять члены одного пространства имен в разных файлах.

В то же время, если нужно объявить несколько небольших пространств имен, это можно сделать в одном PHP-файле, — без необходимости «плодить» лишние сценарии:

```
namespace Foo {
    // Здесь объявляем константы, интерфейсы, функции, классы
}

namespace Bar {
    // Здесь объявляем константы, интерфейсы, функции, классы
}
```

### 24.13.3. Псевдонимы

До появления пространств имен PHP-разработчики решали проблему коллизии имен с помощью имен классов в стиле Zend. Такая схема именования имен была популярна в Zend Framework: вместо символа «\» использовался символ «\_» — например, имя `My_Engine_DocumentService_Adapter_PDF_Query` соответствовало PHP-файлу `My/Engine/DocumentService/Adapter/PDF/Query.php`. Побочный эффект этого стиля был в том, что программистам приходилось использовать очень длинные имена классов.

В современных пространствах имен присутствует та же проблема. Например, посмотрите на полное имя класса `Request` в компоненте `symfony/httpfoundation`. Оно будет таким: `\Symfony\Component\HttpFoundation\Request`. К счастью, PHP позволяет нам создавать псевдонимы и импортировать пространства имен.

Под «импортом» подразумевается то, что вы говорите PHP, какие пространства имен, классы, интерфейсы, функции и константы вы будете использовать в каждом PHP-файле.

Псевдонимы означают, что вы можете ссылаться на импортированные классы, интерфейсы, функции или константы по другому, более короткому, имени. Приведенный далее код создает и отправляет ответ 404 без использования псевдонимов и импорта:

```
<?php
$response = new \Symfony\Component\HttpFoundation\Response('Not found', 404);
$response->send();
```

Не то чтобы это выглядело очень ужасно. Но просто представьте, что вам нужно использовать `Response` несколько раз в своем коде — в одном PHP-файле. Такой код будет смотреться не очень красиво. Посмотрим, как можно сделать то же самое с помощью импорта:

```
<?php
use Symfony\Component\HttpFoundation\Response;
$response = new Response('Not found', 404);
$response->send();
```

Здесь мы говорим PHP, что будем использовать (ключевое слово `use`) класс `Symfony\Component\HttpFoundation\Response`. В итоге длинное имя нам нужно будет указать всего лишь один раз. Далее мы можем использовать просто имя `Response`, что делает наш код гораздо читабельнее, а его написание — проще.

Но можно сделать имена еще короче с помощью псевдонимов. Для объявления псевдонима надо использовать ключевое слово `as`:

```
<?php
use Symfony\Component\HttpFoundation\Response as Res;
$res = new Res('Not found', 404);
$res->send();
```

Начиная с PHP 5.6, возможно импортировать функции и константы. Для этого надо использовать небольшой трюк синтаксиса. Так, для импорта функции нужно изменить `use` на `use func`:

```
<?php
use func Mypspace\functionName;
functionName();
```

Для импорта константы необходимо использовать `use constant` вместо просто `use`:

```
<?php
use constant Mypspace\CONST_NAME;
echo CONST_NAME;
```

Импорт может быть множественным. Для этого просто в операторе `use` можно через запятую указать все, что вы хотите импортировать, — например:

```
use Symfony\Component\HttpFoundation\Request,
    Symfony\Component\HttpFoundation\Response;
```

Но с точки зрения читабельности кода лучше использовать `use` несколько раз — по одному для каждого импортируемого субъекта:

```
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
```

Наконец, пространства имен, согласно стандарту PSR4, поддерживают автоматическую загрузку. Автозагрузка используется многими современными компонентами PHP.

## 24.14. Типажи

*Типажи* (traits), или, как еще их называют, *примеси* — служат заменой мультинаследования. Каждый типаж задается отдельным классом с помощью ключевого слова `trait`.

Внутри типажа вы можете определять методы, которые будут доступны в классе, к которому подключите типажи. Подключение типажей происходит с помощью ключевого слова `use`. Имена типажей задаются через запятую.

Если совпадают имена методов класса и типажей, то будет вызван метод класса, а не типажа, поэтому будьте внимательны, поскольку PHP даже не сообщит вам об этом — не будет ни ошибки, ни даже уведомления (notice).

Чтобы типаж всегда имел доступ к подключенному классу, нужно использовать конструкцию `parent`.

Теперь рассмотрим все сказанное на примере:

```
// определяем класс A
class A {
    public function foo() {
        return 'foo';
    }
}
```

```
// определяем типаж B
trait B {
    public function bar() {
        // вызываем родительский метод foo, добавляем к его результату
        // строку 'bar'
        return parent::foo() . ' bar';
    }
}

// класс C расширяет A
class C extends A {
    // чтобы не описывать члены и методы, мы просто
    // сообщаем, что будем использовать типаж B
    use B;
}

// создаем объект класса C
$c = new C();
echo $c->foo(); // выведет "foo"
echo $c->bar(); // выведет "foo bar"
```

## 24.15. Вызов метода или свойства класса выражением

В PHP 5.4 в свое время появилось еще одно очень приятное нововведение — чтобы вызвать свойство или метод класса, не нужно его запоминать в отдельную переменную.

Рассмотрим пример. Пусть у нас есть класс A, в нем — метод `foo_bar`:

```
class A {
    public static function foo_bar() {
        return '123';
    }
}
```

Вызвать этот метод можно так:

```
echo A::{"foo_var"}();
```

Или даже так:

```
$foo = 'foo';
$bar = 'bar';
echo A::{ $foo . '_' . $bar }();
```

## 24.16. Генераторы

Впервые генераторы появились в PHP 5.5.0. Многие PHP-разработчики до сих пор не понимают назначения генераторов, поскольку их цель не так очевидна, как тех же пространств имен. Генераторы являются простыми итераторами.

В отличие от стандартного итератора, генераторы не требуют реализации интерфейса `Iterator`. Вместо этого генераторы вычисляют значения по требованию.

Давайте создадим простой генератор и посмотрим, как он работает. Генератор — это обычная РНР-функция, которая использует ключевое слово `yield` один или более раз. В отличие от обычной функции, генераторы никогда не возвращают значение (ключевое слово `return`). Они только приводят (`yield`) к значениям. Вот код простого генератора:

```
<?php
function simpleGenerator() {
yield 'value1';
yield 'value2';
yield 'value3';
}
```

Просто, да? Когда вызывается функция-генератор, РНР возвращает объект, принадлежащий классу `Generator`. По этому объекту можно «пройтись» с помощью цикла `foreach`. Во время каждой итерации РНР запрашивает экземпляр `Generator` для вычисления и предоставления следующего значения итерации. Следующий пример:

```
<?php
foreach (simpleGenerator() as $yieldedValue) {
echo $yieldedValue, PHP_EOL;
}
```

выведет:

```
value1
value2
value3
```

С одной стороны, всё кажется довольно простым. Но теперь давайте напишем реальный генератор. Не будете же вы прописывать константы в нескольких `yield`? Именно поэтому мы напишем генератор `getRow`, возвращающий строку из текстового CSV-файла. А затем — программу, которая бы выводила весь текстовый файл построчно с помощью этого генератора. Генератор будет таким:

```
function getRow($file) {
    $handle = fopen($file, 'rb');
    if ($handle === false) {
        throw new Exception();
    }
    while (feof($handle) === false) {
        yield fgetcsv($handle);
    }
    fclose($handle);
}
```

А вот пример программы:

```
foreach (getRow('report.csv') as $row) {  
    print_r($row);  
}
```

Этот пример выделяет память только для одной CSV-строки при каждой итерации вместо того, чтобы прочитать огромный (скажем, 1 Гбайт) CSV-файл в память сразу. Как видите, генераторы — не только косметическое нововведение, но еще и удобная возможность, предоставляемая новыми версиями PHP.

## 24.17. Атрибуты

В PHP 7 стали довольно популярными аннотации PHPDoc — обычные комментарии, позволяющие документировать методы классов, например:

```
class PostsController  
{  
    /**  
     * @Route("/api/posts/{id}", methods={"GET"})  
     */  
    public function get($id) { /* ... */ }  
}
```

В PHP 8 для этого используются атрибуты:

```
class PostsController  
{  
    #[Route("/api/posts/{id}", methods: ["GET"])]  
    public function get($id) { /* ... */ }  
}
```



## ГЛАВА 25

# Хранение данных в Cookies и сессиях

## 25.1. Зачем нужны Cookies и сессии?

Часто бывает нужно где-то хранить несущественные данные о пользователе — например, выбранную тему оформления или данные, которые по тем или иным причинам нельзя хранить в базе данных, — ту же корзину пользователя.

Почему корзину нельзя хранить в базе данных? Хотя бы даже по той причине, что если пользователь еще не зарегистрировался, а уже начал добавлять товары в корзину (большинство интернет-магазинов допускают такое поведение), то мы не знаем, кому принадлежит эта корзина. После регистрации у пользователя будет собственный идентификатор, и можно было бы хранить корзину в базе данных, но зачем это делать, создавая дополнительную нагрузку на базу данных, если можно хранить эти данные в браузере пользователя?

В таких случаях на помощь приходят Cookies и сессии, примеры работы с которыми будут рассмотрены в этой главе книги.

## 25.2. Cookies или хранение данных на стороне клиента

### 25.2.1. Что такое Cookies?

*Cookies* — это небольшие данные, хранящиеся на стороне клиента, т. е. в браузере. Cookies можно использовать, например, для хранения выбранной пользователем цветовой гаммы сайта, а также для хранения элементов корзины заказов.

Поскольку Cookies хранятся на стороне клиента, они могут быть легко подделаны. Следовательно, не нужно помещать в Cookies важную (конфиденциальную) информацию.

Как у всего на свете, у Cookies есть преимущества и недостатки. Начнем с преимуществ:

- Cookies могут храниться годами — пока пользователь их не удалит, не переустановит операционную систему или пока не истечет срок их жизни, установленный программистом, т. е. вами;

□ поскольку Cookies хранятся на стороне клиента, они не создают дополнительную нагрузку на сервер. Вы только представьте, что у нас есть большой сервер, у которого 100 тыс. пользователей (имеются в виду не зарегистрированные пользователи, а только те, которые посетили наш сайт и не поленились выбрать тему дизайна). Нам нужно хранить в Cookies всего одну настройку сайта — цветовую тему сайта. Название темы — это строка длиной до 20 символов. А теперь подсчитаем:  $20 \times 100\,000 = 2\,000\,000$  байтов = 1,95 Мбайт. Вроде бы немного для большого сервера, но, как правило, серьезные сайты не ограничиваются хранением в Cookies только одной переменной, а могут сохранять там довольно внушительные объемы данных.

А вот и недостатки Cookies:

- браузер может не принимать Cookies (некоторые пользователи отключают их прием), и вы даже об этом не узнаете. Максимум, что можно сделать, — это проверить существование того или иного Cookie с помощью функции `isset()`;
- размер хранимой в Cookies информации не должен превышать 512 Кбайт (для каждой переменной), поэтому вы не можете хранить в Cookies большие объемы данных;
- Cookies привязаны к конкретному браузеру. Если пользователь зайдет на сайт с другого браузера или вообще с другого компьютера, вы не сможете использовать ранее установленные пользователем параметры.

## 25.2.2. Установка Cookies

Для установки Cookies используется функция `setcookie()`, которой нужно передать следующие параметры:

- имя Cookie — потом вы будете обращаться к сохраненным данным под этим именем;
- значение переменной Cookie;
- срок жизни — задает время жизни Cookie в секундах. Если этот параметр не указан, Cookie будет существовать до закрытия окна браузера;
- путь Cookie. Если вы хотите, чтобы устанавливаемая переменная относилась только к страницам из каталога `/site/`, то установите `/site/` в качестве значения этой переменной;
- домен Cookie. Предположим, у вас есть два сайта: `orders.example.com` и `www.example.com` — и вы хотите, чтобы устанавливаемая Cookie была доступна на обоих сайтах. Тогда в качестве значения этого параметра нужно указать `.example.com`;
- защита — если этот параметр равен `true`, то Cookie должна передаваться по защищенному соединению (SSL).

Все указанные параметры, кроме первого, необязательны. Рассмотрим пример установки Cookie:

```
setcookie("username", "admin", time()+36000);
```

Здесь мы устанавливаем Cookie с именем `username`, значением `admin` и временем жизни — 10 часов (36 000 секунд).

Функция `setcookie` устанавливает HTTP-заголовки, поэтому до вызова этой функции ваш сценарий не должен производить никакого вывода в браузер. Если вы не можете установить Cookies, тому есть две причины:

1. Браузер их не поддерживает (что маловероятно — ведь ваш браузер их точно поддерживает, и они включены, раз вы надумали их использовать).
2. Сценарий производит вывод до вызова функции `setcookie()` — именно поэтому функцию `setcookie()` нужно вызывать в начале сценария.

Давайте поговорим о второй причине. Вы можете возразить, мол, какой вывод, если в сценарии вообще нет ни одного оператора `echo`?! Обязательно убедитесь, чтобы до и после тегов `<?php` и `?>` не было пробелов и других пробельных символов. Пробельные символы интерпретируются как HTML-код, и PHP выводит их перед выполнением кода сценария. То есть получается, что перед выполнением `setcookie()` уже был какой-то вывод.

После установки Cookie к нему можно обратиться через массив `$_COOKIE`:

```
echo $_COOKIE['username'];
```

### 25.2.3. Удаление Cookies

Для удаления Cookies не нужно использовать функцию `unset()`. Когда вы делаете вызов вроде `unset($_COOKIE['username'])`, то вы просто удаляете элемент массива, но сама Cookie живет на стороне пользователя — т. е. в его браузере, и продолжает жить.

В принципе, удалять Cookies не обязательно — нужно просто подождать, пока истечет их срок жизни, и браузер сам удалит их. Но вот незадача. Представим, что мы разработали корзину для интернет-магазина. Пользователь добавил в корзину товары, оформил заказ, и если не удалить Cookies, то корзина будет содержать ранее добавленные в нее товары, пока не выйдет срок жизни Cookies.

Во-первых, так неудобно. Если пользователь опять захочет сделать заказ, он обнаружит ранее добавленные в корзину товары. Во-вторых, так неправильно с точки зрения конфиденциальности. Если кто-то сядет за компьютер пользователя и зайдет на сайт магазина, он обнаружит, какие товары заказывал пользователь (или добавлял в корзину, что уже достаточно, чтобы понять, чем он интересовался).

Поэтому Cookies нужно удалять, когда они уже не нужны — в конце вашего бизнес-процесса. Для интернет-магазина таким концом служит оформление заказа.

Для удаления Cookies нужно указать отрицательное время, например:

```
setcookie("username", null, -1, '/');
```

Здесь мы не только устанавливаем время `-1`, но и уничтожаем предыдущее значение Cookies (`null`).

## 25.2.4. Организация корзины с помощью Cookies

Рассмотрим практический пример, а именно — добавление товара в корзину, реализованную с помощью Cookies.

Создадим реальный сценарий добавления товара в корзину. Наша корзина будет храниться в виде массива:

```
$cart[ID_товара] = количество_в_корзине;
```

Другими словами, `$cart[5] = 10` означает, что пользователь добавил в корзину 10 единиц товара с ID = 5.

Код сценария приведен в листинге 25.1.

### Листинг 25.1. Добавление товара в корзину

```
<?php
// Проверяем, установлена ли корзина, если да, то "разворачиваем" ее
if (isset($_COOKIE['cart'])) { $cart = unserialize($_COOKIE['cart']); }

$bid = $_GET['id'];           // получаем ID товара
$qty = $_GET['qty'];          // получаем количество
// Проверяем, являются ли числами ID и количество
if (!is_numeric($bid)) die('Error!');
if (!is_numeric($qty)) die('Error qty!');

// добавляем товар в корзину
$cart[$bid] = $qty;
$cnt = count($cart);          // обновляем количество позиций в корзине

// Устанавливаем Cookies заново
setcookie("cart",serialize($cart),time()+60*60*24*30);
setcookie("cnt",$cnt,time()+60*60*24*30);

// Перенаправление на страничку магазина, можно
// перенаправлять на http_referer (откуда пришел пользователь)
Header('Location: shop.php', TRUE, 301);

?>
```

Что мы здесь делаем:

1. Первым делом проверяем, установлена ли Cookie, хранящая корзину. Если да, мы десериализуем строку в массив `$cart`.
2. Получаем параметры (номер товара и количество).
3. Добавляем новый элемент в массив.
4. Устанавливаем Cookie. В качестве значения мы сериализуем наш массив (Cookie может хранить только строку).
5. Выполняем перенаправление на страничку магазина.

Использовать сценарий нужно так:

```
add_cart.php?id=ID-товара&qty=Количество
```

Теперь рассмотрим еще один пример — удаление товара из корзины. Здесь алгоритм будет другим:

1. Выполнить десериализацию корзины.
2. Удалить элемент массива с заданным ID.
3. Сериализовать корзину и установить Cookies.

Код этого сценария приведен в листинге 25.2.

#### Листинг 25.2. Удаление из корзины

```
<?php

$ref = $_SERVER['HTTP_REFERER'];

if (isset($_COOKIE['cart'])) {
    $cart = unserialize($_COOKIE['cart']);
    $cnt = $_COOKIE['cnt'];
}
else {
    Header("Location: $ref");
    die();
}

$bid = $_GET['id'];
if (!is_numeric($bid)) die('Error!');

// проверяем, есть ли в корзине элемент с таким ID
if (isset($cart[$bid])) {

    unset($cart[$bid]);          // удаляем из корзины элемент
    $cnt = $cnt - 1;             // уменьшаем количество товаров в корзине

    // обновляем данные в куках
    setcookie("cart", serialize($cart), time()+60*60*24*30);
    setcookie("cnt", $cnt, time()+60*60*24*30);

}

// Перенаправляем туда, откуда пришли
Header("Location: $ref");

?>
```

Использовать этот сценарий нужно так:

```
cart_delete.php?id=5
```

Понятное дело, что товар с заданным ID должен быть в корзине. Впрочем, сценарий выполняет проверку, так что даже если вы передадите ID товара, которого нет в корзине, ничего страшного не случится.

## 25.3. Механизм сессий

### 25.3.1. Для чего нужны сессии?

Наряду со своей простотой у Cookies есть огромные недостатки. Пусть вы установили переменную с именем А. Другой сайт может тоже установить переменную с таким же именем. В итоге значения переменных перемешаются. Правда, этот недостаток уже решен на уровне браузера — современные браузеры запоминают, какие переменные установлены тем или иным сайтом, и передают только те переменные, которые относятся к вашему сайту. Второй недостаток заключается в самом хранении данных на компьютере пользователя. Представьте, чтобы после успешной проверки имени пользователя и пароля на сайте создали две Cookie-переменных: `auth` (признак успешной авторизации) и `login` (содержит логин пользователя). Поскольку данные хранятся на компьютере пользователя, никто не мешает злоумышленнику создать две такие переменные в Cookies браузера и зайти под именем, указанным переменной `login` без ввода пароля на сайт. Если вы не знаете, как это сделать, то это совсем не означает, что это невозможно. Я, к примеру, могу написать собственный браузер, который будет подделывать переменные Cookies так, как мне нужно. Да и писать собственный браузер не придется — в том же Opera есть функция управления Cookies, позволяющая, помимо всего прочего, еще и изменять значения Cookies-переменных. Так что обойти защиту, построенную на базе Cookies, не составит особого труда даже школьнику.

Значения переменных *сессии*, в отличие от Cookies, хранятся на сервере, а туда злоумышленник не доберется. Существуют более простые способы взлома сайта, чем получение доступа к каталогу, в котором хранятся переменные сессии, и их подделка.

Сессия обычно существует до закрытия окна браузера, поэтому вы можете особо не беспокоиться за пользователя, который забыл нажать кнопку **Выйти**, работая в интернет-кафе. А вот если бы защита была построена на Cookies, было бы обидно — тогда любой другой пользователь смог бы зайти под его именем на сайт.

Работать с сессией очень просто. Это гораздо проще, чем вы можете себе представить. В начало сценария, работающего с сессиями, нужно поместить два вызова функций:

```
session_name("mysession");  
session_start();
```

Первая функция задает имя сессии, а вторая — запускает сессию. Тут есть два важных момента:

- имя сессии можно и не задавать — оно будет сгенерировано PHP автоматически. Однако, если ваш сайт состоит не из одного файла `index.php`, а из нескольких PHP-сценариев — например: `login.php` (проверяет имя пользователя и пароль), `index.php` (основной файл), `profile.php` (позволяет изменять параметры учетной записи пользователя), тогда нужно задать имя сессии. Иначе, если в каждом файле выполнить просто `session_start()`, то у каждого будет своя сессия. Поэтому желательно задать имя сессии и указывать его в каждом сценарии, который должен «подключиться» к созданной сессии;
- второй момент заключается в самом вызове этих функций. Все, что связано с Cookies и сессиями, должно устанавливаться/создаваться до первого вывода из сценария в браузер, т. к. после первого вывода формирование заголовков сервера будет завершено, и вы не сможете установить Cookies или создать сессию.

### 25.3.2. Автоматическое создание сессии

Интерпретатор PHP позволяет инициализировать сессию автоматически — сразу при запуске сценария без вызова функции `session_start()`. Для этого нужно было бы установить параметр `session.auto_start=1` в файле конфигурации `php.ini`. Но так делать не рекомендуется, потому что в этом случае всегда будет создаваться «безымянная сессия», и вы не сможете установить имя сессии с помощью функции `session_name()`.

### 25.3.3. Хранение данных в сессии

Чтобы поместить в сессию данные, достаточно поместить их в массив `$_SESSION`, — например:

```
$_SESSION['username'] = 'admin';
```

Больше ничего делать не нужно. Если вы перед этим создали сессию, или она была создана автоматически, данные будут автоматически помещены в сессию.

Для удаления данных из сессии используйте функцию `unset()`:

```
unset($_SESSION['username']);
```

Также можно использовать функцию `session_unset()`, которая удаляет все переменные сессии, которые были зарегистрированы на текущий момент. Функция не принимает каких-либо параметров и освобождает все переменные сессии.

\* \* \*

Нужно отметить, что способы работы с Cookies и сессиями, рассмотренные в этой главе, будут работать, только если вы пишете код без использования фреймворков. Фреймворк предоставляет свои способы работы с Cookies и сессиями, и использовать нужно именно их, поскольку стандартные способы работать не будут.



## ГЛАВА 26

# Обработка исключений

### 26.1. Введение в обработку исключений

В PHP реализована полноценная модель исключений, как и в других современных языках программирования, — вроде C#. Исключение может быть выброшено с помощью ключевого слова `throw` и поймано с помощью блока `catch`. Блок `catch` обычно используется внутри блока `try..catch..finally`.

Если вы не программировали на C#, тогда поясню, что к чему. В блок `try` заключается потенциально «опасный» код, во время выполнения которого может возникнуть исключение — например, произойти ошибка при подключении к БД, при открытии файла и т. д. После блока `catch` вы можете определить один или несколько блоков исключений. Исключения генерируются, если в случае выполнения кода, заключенного в `try`, возникла ошибка. В блок `finally` помещаются операторы, которые будут выполнены в любом случае.

Начиная с PHP 8.0, ключевое слово `throw` является *выражением* и может быть использовано в любом контексте выражения. Для сравнения: в предыдущих версиях PHP оно было *утверждением* и должно было располагаться в отдельной строке.

Выброшенный объект должен наследовать (`instanceof`) интерфейс `Throwable`. Попытка выбросить объект, который таковым не является, приведет к фатальной ошибке PHP (PHP Fatal Error).

Если выброшено исключение, а в текущей области видимости функции нет блока `catch`, исключение будет «подниматься» по стеку вызовов к вызывающей функции, пока не найдет подходящий блок `catch`. Все блоки `finally`, которые встретятся на этом пути, будут выполнены. Когда стек вызовов разворачивается до глобальной области видимости, не встречая подходящего блока `catch`, сценарий завершается с неисправимой ошибкой, если не был установлен глобальный обработчик исключений (об этом мы поговорим отдельно).

Пример обработки исключений приведен в листинге 26.1.

**Листинг 26.1. Пример обработки исключений**

```
<?php
try {
    throw new Exception('Порождаем не PDO исключение');
    $pdo = new PDO('mysql://host=wrong_host;dbname=wrong_name');
} catch (PDOException $e) {
    // Обработка исключений PDO
    echo "Ошибка PDO";
} catch (Exception $e) {
    // Обработка всех других исключений
    echo "Другая ошибка";
} finally {
    // Всегда выполняем это
    echo "Будет выведено всегда";
}
```

Здесь мы порождаем исключение из соображений демонстрации возможностей `try..catch..finally`.

Так, в блоках `catch` мы «улавливаем» исключения типов `PDOException` и `Exception`. Первое исключение относится непосредственно к расширению PDO, а вот `Exception` — это любая другая исключительная ситуация.

## 26.2. Блок *catch*

Этот блок устанавливает, как нужно реагировать на выброшенное исключение. В блоке `catch` определяется один или несколько типов ошибок, которые он может обработать, и переменная (хотя это не обязательно), которой можно присвоить исключение (указание переменной было обязательно до PHP 8). Первый блок `catch`, с которым столкнется выброшенное исключение или ошибка, соответствующий типу выброшенного объекта, обработает объект.

Несколько блоков `catch` можно использовать для перехвата различных классов исключений. Нормальное выполнение (когда исключение не выброшено в блоке `try`) будет продолжаться после последнего блока `catch`, определенного в последовательности. Исключения могут быть выброшены (с помощью `throw`) внутри блока `catch` — иначе выполнение будет продолжено после блока `catch`, который был вызван.

Когда возникнет исключение, код, следующий за утверждением, не будет выполнен, а интерпретатор попытается найти первый подходящий блок `catch`. Если исключение не поймано, будет выдана фатальная ошибка PHP с сообщением **Uncaught Exception ...**, если только обработчик не был определен с помощью функции `set_exception_handler()`.

Рассмотрим еще один пример использования исключений (листинг 26.2).

**Листинг 26.2. Самый простой пример использования исключений**

```
try {
    print "try\n";
    throw new Exception();
} catch (Exception $e) {
    print "some error\n";
} finally {
    print "finally\n";
}
```

Вывод будет таким:

```
try
some error
finally
```

Начиная с версии PHP 7.1, в блоке `catch` можно указывать несколько исключений, используя символ «`|`». Это полезно, когда разные исключения из разных иерархий классов обрабатываются одинаково.

Начиная с версии PHP 8.0, имя переменной для пойманного исключения является необязательным. Если оно не указано, блок `catch` будет выполнен, но он не будет иметь доступа к выброшенному объекту.

## 26.3. Блок *finally*

Этот блок также может быть указан после или вместо блоков `catch`. Код в блоке `finally` всегда будет выполняться после блоков `try` и `catch` — независимо от того, было ли выброшено исключение, и до возобновления нормального выполнения.

Одно из заметных взаимодействий происходит между блоком `finally` и оператором `return`. Если оператор `return` встречается внутри блоков `try` или `catch`, блок `finally` все равно будет выполнен. Более того, оператор `return` выполнится, когда встретится, но результат будет возвращен после выполнения блока `finally`. Кроме того, если блок `finally` также содержит оператор `return`, возвращается значение из блока `finally`.

В версии PHP 5.5 вы можете столкнуться с ошибочным поведением `finally`, о котором вы должны знать. Рассмотрим пример такого поведения (листинг 26.3).

**Листинг 26.3. Неожиданное поведение `finally` в PHP 5.5**

```
<?php
try {
    try {
        echo "First\n";
        throw new Exception("Error 1");
    }
```

```
    } finally {  
        try {  
            echo "second\n";  
            throw new Exception("Error 2");  
        } catch (\Exception $e) {  
            echo "Third\n";  
        }  
    }  
    echo "Here\n";  
} catch (\Exception $e) {  
    echo "Exception ". $e->getMessage(). "\n";  
}  
echo "Done\n";
```

Если внимательно прочитать код, то вывод должен быть таким (рис. 26.1):

```
First  
second  
Third  
Exception Error 1  
Done
```

В версиях 5.6–8.x он такой и есть. А вот в версии 5.5 вывод довольно неожиданный:

```
First  
second  
Third  
Done
```

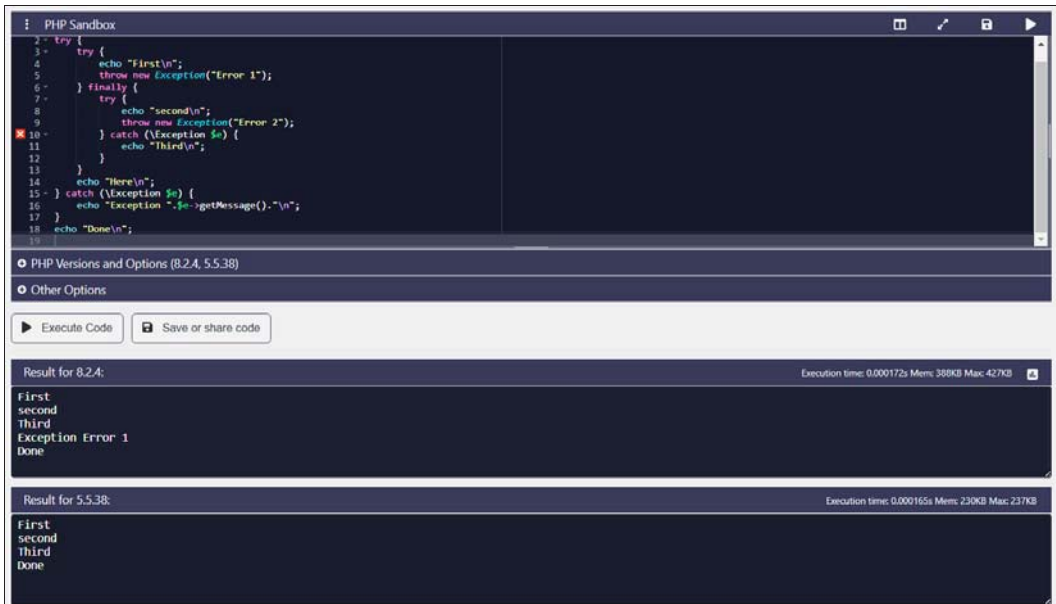


Рис. 26.1. Результат выполнения кода в разных версиях PHP

## 26.4. Глобальный обработчик исключений

Если исключению разрешено распространяться на глобальную область видимости, оно может быть перехвачено *глобальным обработчиком исключений*, если он установлен. Функция `set_exception_handler()` может задать функцию, которая будет вызвана вместо блока `catch`, если не будет вызван никакой другой блок. Эффект по сути такой же, как если бы вся программа была обернута в блок `try-catch` с этой функцией в качестве `catch` (листинг 26.4).

**Листинг 26.4. Пример установки глобального обработчика исключений**

```
<?php
function exceptions_error_handler($severity, $message, $filename, $lineno) {
    throw new Exception($message, 0, $severity, $filename, $lineno);
}

set_error_handler('exceptions_error_handler');
?>
```



## ГЛАВА 27

# Контроль версий

### 27.1. Выбор системы контроля версий

Если вы работаете над крупным проектом, вам просто необходима система контроля версий (далее — просто система). Система будет полезна как программисту-одиночке (фрилансеру), так и группе программистов, работающих над одним проектом. В последнем случае такая система не просто приятное дополнение к набору инструментов, а необходимость. Итак, разберемся, зачем вам нужна система контроля версий. По меньшей мере есть три причины использовать контроль версий в вашем проекте:

- ❑ система позволяет сэкономить время и сберечь ваши нервы, если вы случайно испортите какие-то файлы или внесете изменения, после которых ваш проект вообще откажется работать. Да, можно закомментировать старый код перед тем, как писать новый. И новый код всегда можно будет потом удалить и раскомментировать старый, но зачем себя так утруждать, если есть система контроля версий, которая поможет восстановить предыдущее состояние файла? В Windows встроен механизм теневых копий файла, позволяющий восстановить предыдущее содержимое файла, но Windows используют не все, да и с оболочкой системы контроля версий работать удобнее;
- ❑ при групповой работе над проектом всегда видно, кто, когда и какие изменения вносил в файлы проекта. Вася Пупкин уже не сможет избежать ответственности за код с ошибками, внесенный в проект;
- ❑ вы имеете возможность безболезненно и с наименьшими затратами времени объединить фрагменты разных проектов в новый проект.

Существуют различные системы контроля версий. В настоящее время стандартом де-факто является система контроля версий Git. Именно Git принято использовать во множестве IT-компаний мира — от небольших до самых крупных.

Далее мы рассмотрим основы использования Git, но прежде мы должны поговорить о том, как установить Git. Если вы работаете под Linux (либо разработка ведется на VDS под управлением Linux, к которому вы подключаетесь по SSH), то делать это вам уже не нужно, поскольку в большинстве случаев Git уже установлен.

В операционной системе Windows можно использовать или Git for Windows, (<https://gitforwindows.org/>) или графическую оболочку GitHub Desktop (<https://desktop.github.com/>). Особенно удобно эту оболочку будет использовать новичкам и небольшим проектам, которые хранят свой код на GitHub.

## 27.2. Первоначальная настройка

Начнем с настройки информации о пользователе для всех локальных репозиториев. Вот команда, устанавливающая имя, которое будет отображаться в поле автора у выполняемых вами коммитов:

```
$ git config --global user.name "[ИМЯ]"
```

Следующая команда устанавливает адрес электронной почты, который будет отображаться в информации о выполняемых вами коммитах:

```
$ git config --global user.email "[адрес электронной почты]"
```

## 27.3. Создание нового репозитория или получение его по существующему URL-адресу

*Репозиторий Git* — просто база данных, содержащая всю информацию, необходимую для управления версиями и историей проекта. В Git, как и в большинстве других систем управления версиями, репозиторий хранит полную копию всего проекта на протяжении всей его жизни. Однако, в отличие от большинства других VCS, репозиторий Git не только содержит полную рабочую копию всех файлов, но также и копию самого репозитория, с которым работает.

Следующая команда создает новый локальный репозиторий с заданным именем:

```
$ git init [название проекта]
```

Команда `clone` скачивает репозиторий вместе со всей его историей изменений:

```
$ git clone [url-адрес]
```

Дабы смоделировать типичную ситуацию, сейчас мы создадим репозиторий для вашего персонального сайта в каталоге `~/public_html`.

Если у вас нет своего личного сайта в `~/public_html`, создайте этот каталог и поместите в него файл `index.html` с любой строкой:

```
$ mkdir ~/public_html
$ cd ~/public_html
$ echo 'Мой сайт' > index.html
```

Чтобы превратить `~/public_html` или любой другой каталог в репозиторий Git, просто запустите `git init`:

```
$ git init
Initialized empty Git repository in .git/
```

Команде `git` все равно, начинаете ли вы с полностью пустого каталога или с каталога, полного файлов. Процесс преобразования каталога в репозиторий Git будет одинаковым.

Чтобы отметить, что ваш каталог является репозиторием Git, команда `git init` создает скрытый каталог с названием `.git`, находящийся на верхнем уровне вашего проекта.

Все остальное в каталоге `~/public_html` останется нетронутым. Git считает этот каталог рабочим каталогом вашего проекта, в котором вы изменяете свои файлы. Git интересуют только файлы из скрытого каталога `.git`.

## 27.4. Операции с файлами. Перемещение и удаление версий файлов репозитория

Итак, команда `git init` создает новый репозиторий Git. В самом начале каждый репозиторий Git будет пустым. Чтобы управлять контентом, вы должны явно добавить его в репозиторий. Такой осознанный файл позволяет отделить рабочие файлы от важных файлов.

Команда `git add <файл>` служит для добавления файла `<файл>` в репозиторий:

```
$ git add index.html
```

### ПРИМЕЧАНИЕ

Если в вашем каталоге есть несколько файлов, не нужно добавлять все их вручную, пусть за вас это сделает `git`. Чтобы добавить в репозиторий все файлы каталога и всех подкаталогов, используйте команду `git add .` (одна точка в UNIX означает текущий каталог).

После добавления файла Git знает, что файл `index.html` принадлежит репозиторию. Но Git просто подготовил файл, это еще не все. В Git разделены операции добавления и фиксации. Это сделано для лучшей производительности — вы только представьте, сколько времени займет обновление репозитория при каждом добавлении или удалении файла. Вместо этого Git предлагает отдельные операции, что особенно удобно в пакетном режиме.

Команда `git status` покажет промежуточное состояние файла `index.html`:

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
# new file:   index.html
```

Эта команда сообщает, что при следующей фиксации в репозиторий будет добавлен новый файл `index.html`.

Команда `rm` удаляет конкретный файл из рабочего каталога и индексирует его удаление:

```
$ git rm [файл]
```

Параметр `-cached` убирает конкретный файл из контроля версий, но физически оставляет его на своем месте:

```
$ git rm --cached [файл]
```

Команда `mv` перемещает и переименовывает указанный файл, сразу индексируя его для последующего коммита:

```
$ git mv [оригинальный файл] [новое имя]
```

Git будет игнорировать файлы и каталоги, указанные в файле `.gitignore` с помощью wildcard-синтаксиса:

```
*.log  
build/  
temp-*
```

```
$ git ls-files --others --ignored --exclude-standard
```

Это список всех игнорируемых файлов в текущем проекте.

## 27.5. Сохранение и восстановление незавершенных изменений

Команда `stash`:

- временно сохраняет все незафиксированные изменения отслеживаемых файлов:

```
$ git stash
```

- восстанавливает состояние ранее сохраненных версий файлов:

```
$ git stash pop
```

- выводит список всех временных сохранений:

```
$ git stash list
```

Команда `stash drop` сбрасывает последние временно сохраненные изменения:

```
$ git stash drop
```

## 27.6. Просмотр изменений и создание коммитов (фиксация изменений)

- Команда `$ git status`

Перечисляет все новые или измененные файлы, которые нуждаются в фиксации.

❑ Команда `$ git diff`

Показывает различия по внесенным изменениям в еще не проиндексированных файлах.

❑ Команда `$ git add [файл]`

Индексирует указанный файл для последующего коммита.

❑ Команда `$ git diff --staged`

Показывает различия между проиндексированной и последней зафиксированной версиями файлов.

❑ Команда `$ git reset [файл]`

Отменяет индексацию указанного файла, при этом сохраняет его содержимое.

❑ Команда `$ git commit -m "[сообщение с описанием]"`

Фиксирует проиндексированные изменения и сохраняет их в историю версий.

## 27.7. Коллективная работа

Команда `branch`, если имя ветки не указано, выводит список именованных веток коммитов с указанием выбранной ветки:

```
$ git branch
```

Создать новую ветку можно так:

```
$ git branch [ИМЯ ветки]
```

Переключиться на выбранную ветку и обновить рабочий каталог до ее состояния:

```
$ git switch -c [ИМЯ ветки]
```

Внести изменения указанной ветки в текущую ветку:

```
$ git merge [ИМЯ ветки]
```

Удалить выбранную ветку:

```
$ git branch -d [ИМЯ ветки]
```

## 27.8. Просмотр и изучение истории изменений файлов проекта

История коммитов (зафиксированных изменений) для текущей ветки может быть получена так:

```
$ git log
```

История изменений конкретного файла, включая его переименование, выводится следующей командой:

```
$ git log --follow [файл]
```

Показать разницу между содержанием коммитов двух веток:

```
$ git diff [первая ветка]...[вторая ветка]
```

Вывести информацию и показать изменения в выбранном коммите:

```
$ git show [КОММИТ]
```

## 27.9. Откат изменений. Удаление ошибок и корректировка созданной истории

Следующая команда отменяет все коммиты после заданного, оставляя все изменения в рабочем каталоге:

```
$ git reset [КОММИТ]
```

Параметр `-hard` сбрасывает всю историю вместе с состоянием рабочего каталога до указанного коммита:

```
$ git reset --hard [КОММИТ]
```

## 27.10. Синхронизация с удаленным репозиторием. Регистрация удаленного репозитория и обмен изменениями

❑ Команда `fetch` скачивает всю историю из удаленного (`remote`) репозитория:

```
$ git fetch [удаленный репозиторий]
```

❑ Команда `$ git merge [удаленный репозиторий] / [ветка]`

Вносит изменения из ветки удаленного репозитория в текущую ветку локального репозитория.

❑ Команда `$ git push [удаленный репозиторий] [ветка]`

Загружает все изменения локальной ветки в удаленный репозиторий.

❑ Команда `$ git pull`

Загружает историю из удаленного репозитория и объединяет ее с локальной (то есть операция `pull = fetch + merge`).



## ГЛАВА 28

# Тестирование РНР-сценариев

### 28.1. Программа работает, но не так, как нам нужно

Знакома ли вам такая ситуация? Вроде бы все правильно, программа запускается, ошибок нет, что-то она делает, но явно не то, чего мы от нее ожидаем. Возникает вопрос: в чем же причина? Возможно, какая-то функция вернула неправильный результат, возможно, мы передали функции вследствие логической ошибки неверное значение. Да, именно логические ошибки самые коварные. Ведь с точки зрения синтаксиса все написано правильно. Синтаксические ошибки выявить просто, поскольку их помогает устранить интерпретатор. А вот ошибки в логике программы интерпретатор устранить не в состоянии, поскольку он пока еще не умеет читать наши мысли.

Если вы ранее программировали на C++, Object Pascal или Java, то вам будет особенно сложно. Ведь вы привыкли к строгому коду, да и компиляторы этих языков программирования выявляют очень много различных ошибок, причем не только синтаксических, но иногда помогают устранять и логические ошибки. Еще раз повторюсь, что работать с РНР будет намного труднее. Во-первых, РНР не требует обязательной инициализации переменных, как в случае с другими языками программирования. Во-вторых, само понятие «тип переменной» здесь довольно размыто. Вот небольшой пример:

```
$a = 'test string';  
$a = 10-5;  
$b = '5';  
$c = $a + $b;
```

Сначала переменная `$a` у нас является строкой (в том же Object Pascal это тип `string`). Затем `$a` присваивается целое значение (5) — выходит, она теперь стала переменной целого типа. Переменная `$b` должна быть строкой, поскольку ее значение заключено в апострофы, но, как видите, компилятор «ни слова не скажет», когда вы попытаетесь сложить переменные двух разных типов: целого (переменная `$a`) и строкового (`$b`). Переменная `$c`, как ни в чем не бывало, будет содержать целое значение 10.

Попробуйте реализовать такое в C++. Без функций преобразования типов у вас ничего не выйдет, а в PHP — пожалуйста, типы переменных преобразуются «на лету». С одной стороны, это хорошо, с другой — не очень, поскольку строгий код C++ (Pascal, Java) дисциплинирует программиста. Да и такое преобразование может стать источником проблем. Допустим, у нас есть функция, которая ожидает параметр определенного типа — например, строку. А вы по ошибке передаете ей переменную, содержащую целое значение. Интерпретатор автоматически преобразует тип переменной, но понятно, что функция будет работать неправильно. Скажем, функция ожидала имя файла, а вы передали ей какое-то число.

Кроме того, наряду с преобразованием типов переменных «на лету» и отсутствием объявлений переменных непривычным является символ доллара, с которого в PHP начинается имя каждой переменной. Почему это так, думаю, понятно. В том же C++ или Pascal нужно сначала объявить переменные, т. е. указать имя переменной и ее тип (можно сразу задать и ее значение). А в PHP, как уже было отмечено, тип не является обязательным атрибутом переменной. Конечно, можно было бы в начале каждого сценария указывать имена переменных и их значения, но это неудобно. Поэтому, чтобы выделить переменные из прочей «серой массы» кода, приходится использовать знак доллара. Программист, не привыкший к значку \$, будет делать много ошибок (особенно поначалу). Вот небольшой пример:

```
for ($i = 0; $i < N; $i++) echo $x[$i];
```

А вот еще один:

```
for ($i = 0; $i < 100; $i++) echo $x[i];
```

Почему вывод пуст? Ведь массив содержит данные. А потому что в первом случае забыли указать знак доллара для переменной \$N, а во втором — для \$i. А ведь интерпретатор при этом не сказал ни слова, он просто молча «проглотил» код.

За свободу, как видите, приходится расплачиваться большим количеством ошибок в коде, которые очень непросто выявить. К тому же некоторые наиболее коварные ошибки проявляются только при определенных условиях, а в остальных случаях все работает отлично.

## 28.2. «Самодельные» точки останова

Если вы когда-нибудь программировали на C++ или Pascal, то знаете, что в любом отладчике есть возможность установки *точки останова* (breakpoint). Вы выбирали произвольную (нужную вам) строчку кода и устанавливали точку останова. Отладчик выполнял программу до заданной точки останова, и выполнение программы приостанавливалось. А вы во время «рекламной паузы» могли просмотреть значения любой переменной (предварительно, переменные нужно было внести в Watch-список).

Для PHP отладчика, подобного описанному, пока нет (точнее, есть, но за него нужно выложить приличную сумму — Zend Studio хотя и значительно подешевел в персональной редакции, но все равно новички могут отказаться за него платить).

Можно, конечно, использовать среду разработки Eclipse, но это, во-первых, не всегда возможно, а, во-вторых, зачастую не дает желаемых результатов. Поэтому попробуем организовать отладчик своими руками.

Роль точки останова будет выполнять функция `die()`, завершающая работу сценария. В нужном месте вы вызываете эту функцию, а в качестве ее параметра передаете все переменные, которые хотите вывести. Если выводить значения переменных через функцию `die()` вам неудобно, можно сначала вывести значения переменных, а затем вызвать `die()`.

Вот небольшой пример:

```
$a = 10; $b= 'test';  
...  
die('a = '.$a. ' b = '.$b);
```

У этого решения всего один недостаток — точка останова может быть только одна, и мы не сможем продолжить выполнение после вызова функции `die()`. Но это все же лучше, чем вообще ничего.

Также можно использовать функции `print_r()` и `var_dump()`, чтобы «заглянуть» в переменную. Особенно они полезны в сложных случаях — при работе с объектами и вложенными массивами:

```
print_r($obj);  
var_dump($arr);  
die();
```

Прототипы этих функций выглядят так:

```
var_dump(mixed $value, mixed ...$values): void  
print_r(mixed $value, bool $return = false): string|bool
```

Функция `print_r()` позволяет не выводить содержимое значения на экран, а возвращать в виде строки. Для этого используется второй параметр, который нужно установить в `true`:

```
$output = print_r($obj, true);
```

Затем содержимое `$output` можно вывести в той же `die()` или же записать в файл с помощью `file_put_contents()`. Запись в файл особенно полезна при отладке сложных сценариев, чтобы можно было в спокойной обстановке проанализировать возникшую проблему. Пример:

```
try {  
    // что-то делаем  
}  
except {  
    die('Что-то пошло не так\n');  
    $output = print_r($obj, true);  
    file_put_contents('debug.log', $output);  
}
```

## 28.3. Система автоматического тестирования

Немного облегчить отладку PHP-приложений поможет система автоматического тестирования SimpleTest, являющаяся развитием систем JUnit для Java, точнее, ее аналогом для PHP. Скачать систему можно по адресу: <https://sourceforge.net/projects/simpletest/files/>. Здесь доступна как сама система, так и модуль для системы Eclipse, если вы ею пользуетесь. В каталоге Glava\_28 сопровождающего книгу электронного архива (см. *приложение 4*) вы найдете файл `simpletest_1.0.1.tar.gz` с версией 1.0.1 этого пакета.

Распакуйте архив системы в каталог `simpletest`. В каталоге `simpletest/docs` содержится подробное описание системы, а в каталоге `simpletest/test` — много различных примеров.

Разберемся, как работает система на простейшем примере ее использования. Создайте файл `ex.php` и поместите его в каталог `simpletest`. Код этого файла приведен в листинге 28.1. Внимательно читайте комментарии, чтобы понять, что и как происходит. Можно, конечно, было бы описать методы отдельно, но вместе с реальным кодом комментарии воспринимаются проще.

### Листинг 28.1. Пример использования системы автоматического тестирования

```
<?php

// подключаем необходимые файлы
require_once ("simpletest/unit_tester.php");
require_once ("simpletest/reporter.php");
// пример тестирования кода PHP с помощью SimpleTest

// создаем класс PHPSimpleTest, который расширяет стандартный
// класс UnitTestCase
class PHPSimpleTest extends UnitTestCase {

function setUp() {

// здесь нужно задать действия, которые должны быть выполнены
// до вызова функции тестирования testCode1()

}

function tearDown() {
// здесь нужно задать действия, которые должны быть выполнены
// после вызова функции тестирования testCode1()

}

// а вот сама функция тестирования PHP-кода
function testCode1()
{
```

```
// определяем переменную $a
$a=2*5;

// проверка на равенство
$this->assertEqual($a,10);

// определяем массив
$ARR=array($a,10);

// вывод значения $ARR и $a
$this->dump($ARR);
$this->dump($a);

// производим проверку на равенство значения и соответствие типа
$this->assertIdentical($ARR,array(10,10));
// проверяем значение на истинность
$this->assertTrue($a==10);

// проверяем значение на ложность
$this->assertFalse($a!=10);
}
}
// вызываем тестирование
$test = new PHPSimpleTest();
$reporter = new HTMLReporter();

$test->run($reporter);

?>
```

### **ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл *ex.php*), представленный в каталоге *Glava\_28* сопровождающего книгу электронного архива (см. приложение 4).

Запустите сценарий *ex.php* в окне браузера. Результат его выполнения представлен на рис. 28.1.

Как здесь можно видеть, сначала выведены значения массива *\$ARR* и переменной *\$a*, а затем выводится общий отчет о тестировании — проверена одна тест-функция, которая прошла все четыре теста:

```
// проверка на равенство
$this->assertEqual($a,10);
// производим проверку на равенство значения и соответствие типа
$this->assertIdentical($arr,array(10,10));
// проверяем значение на истинность
$this->assertTrue($a==10);
```

```
// проверяем значение на ложность  
$this->assertFalse($a!=10);
```

Для лучшего понимания сделайте следующий эксперимент — перед строкой

```
$this->assertTrue($a==10);
```

добавьте строку

```
$a = 9;
```

Этим мы сделали все возможное, чтобы наша функция «провалила» тест. Запустите еще раз сценарий и посмотрите на результат (рис. 28.2).

Поскольку мы присвоили переменной `$a` значение 9, то два последних теста оказались невыполненными:

- ☐ проверка на истинность условия, что переменная `$a` равна 10;
- ☐ проверка на ложность того, что `$a` не равна 10 (поскольку `$a = 9`, то результатом логического выражения `$a!=10` будет истина, а мы проверяли выражение на ложность).

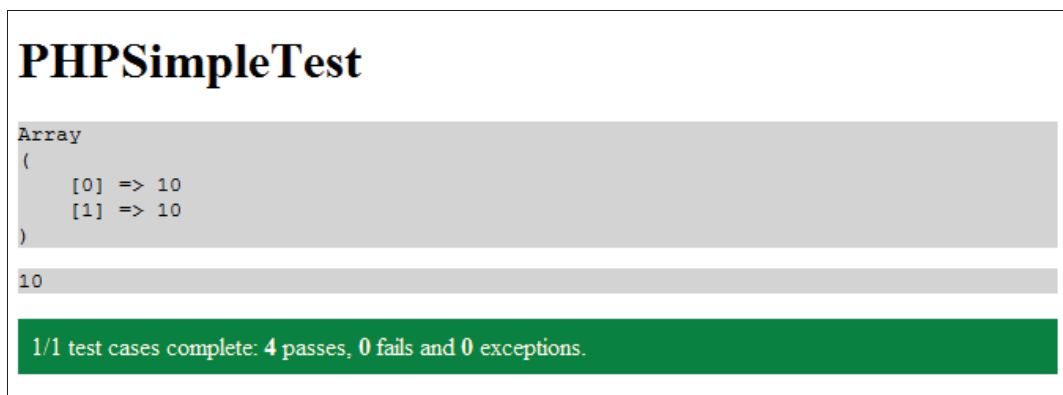


Рис. 28.1. Результат проверки кода

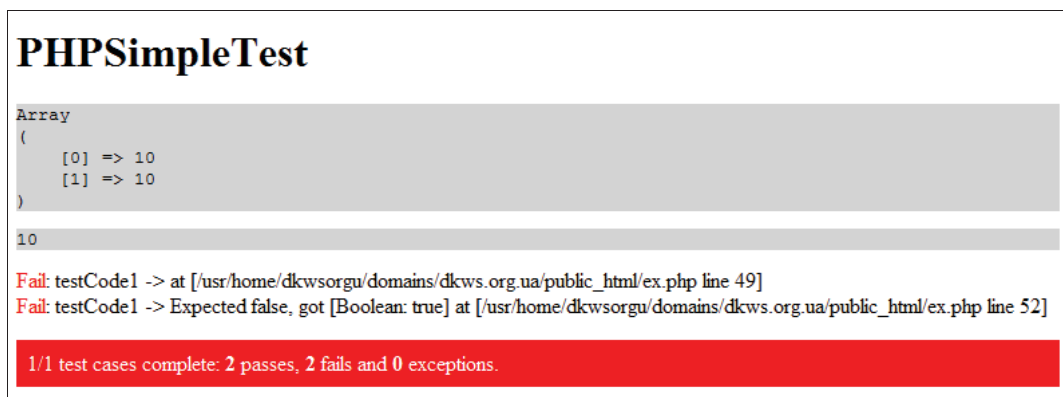


Рис. 28.2. Функция «провалила» два теста

Если одной функции для проверки кода вам мало, можно добавить еще одну. Функция должна называться `testCodeN()`, где `N` — номер функции. Например:

```
function testCode2()  
{  
    $B=12*15;  
    $this->dump($B);  
}
```

## 28.4. Директива *error\_reporting*

При отладке сценариев не нужно забывать об очень полезной директиве: `error_reporting`. Откройте ваш файл `php.ini`. В Windows он находится или в основном каталоге Windows (`C:\Windows`), или в одном с PHP каталоге. В Linux этот файл может находиться в каталоге `/etc` или в `/etc/php/<версия>/cli` (зависит от вашего дистрибутива). В FreeBSD файл `php.ini` может находиться в каталоге `/usr/local/lib` или любом другом каталоге, указанном при сборке PHP.

Найдите директиву `error_reporting` и установите для нее значение `E_ALL`:

```
error_reporting = E_ALL
```

Можно также установить и другие значения (они описаны в файле `php.ini`), но на этапе отладки лучше, чтобы PHP выводил все предупреждения и сообщения об ошибках.

Установить значение этой директивы можно и во время работы сценария так:

```
error_reporting( 'E_ALL' );
```



**ЧАСТЬ II**

---

**ПРАКТИКА**



## **II. РАЗДЕЛ 7**

### **Разработка основных элементов сайта**

<b>Глава 29.</b>	Загрузка файлов на сервер
<b>Глава 30.</b>	Использование FTP-функций
<b>Глава 31.</b>	Отправка и прием почты
<b>Глава 32.</b>	Введение в PEAR
<b>Глава 33.</b>	Импорт и экспорт данных
<b>Глава 34.</b>	Работаем с MP3
<b>Глава 35.</b>	Расширение cURL: практические примеры



## ГЛАВА 29

# Загрузка файлов на сервер

### 29.1. Что нужно знать о загрузке файлов на сервер?

Прежде всего следует выяснить, зачем нам это нужно. Оказывается, многие серьезные проекты требуют загрузки файлов на сервер (в англоязычной литературе этот процесс называется *upload*). Вот несколько примеров:

- ☐ форумы и чаты — загрузка аватаров пользователей;
- ☐ фотогалерея — загрузка фотографий;
- ☐ интернет-магазин — загрузка фотографий товаров;
- ☐ веб-интерфейс почтовой службы — сначала нужно загрузить файл на сервер, а только после этого прикрепить его к письму;
- ☐ веб-интерфейс системы управления контентом сайта — в этом случае без загрузки файлов на сервер тоже никак не обойтись.

Как видите, загрузка файлов на сервер встречается очень часто. На самом деле эта операция довольно простая. И здесь мы поговорим о том, как реализовать загрузку своими силами, не прибегая для этого к использованию посторонних классов.

Для загрузки файлов на сервер существуют специальные формы загрузки — так называемые *multipart*-формы (листинг 29.1).

#### Листинг 29.1. Multipart-форма для загрузки файлов на сервер

```
<html>
<body>
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
Выберите файл: <INPUT NAME="upfile" TYPE="file">

<INPUT TYPE="submit" VALUE="Загрузить">
```

```
</FORM>
</body>
</html>
```

### ПРИМЕЧАНИЕ

Коды формы загрузки (файл *upload.html*) и сценария загрузки (файл *upload.php*), приведенного в листинге 29.3, вы найдете в каталоге *Glava\_29* сопровождающего книгу электронного архива (см. *приложение 4*).

### ПРИМЕЧАНИЕ

Если вы делаете форму загрузки, которая будет использоваться в представлениях Laravel, не забудьте поместить директиву Blade `@csrf` для передачи токена в приложение. Эта техника позволяет защитить ваши приложения от CSRF-атаки. Директиву `@csrf` нужно поместить в любом месте формы, но лучше сразу после тега `<form>`, чтобы не забыть о ней.

Multipart-форма отличается от обычной прежде всего наличием параметра `ENCTYPE="multipart/form-data"`. Но это еще не все. Второй обязательный атрибут такой формы — наличие поля для выбора файла:

```
<INPUT NAME="upfile" TYPE="file">
```

Все остальное реализуется как обычно: задается сценарий, который будет обрабатывать форму (параметр `ACTION`), метод передачи файла (`METHOD`) и кнопка передачи информации (параметр `submit`). В окне браузера наша multipart-форма выглядит так, как показано на рис. 29.1.

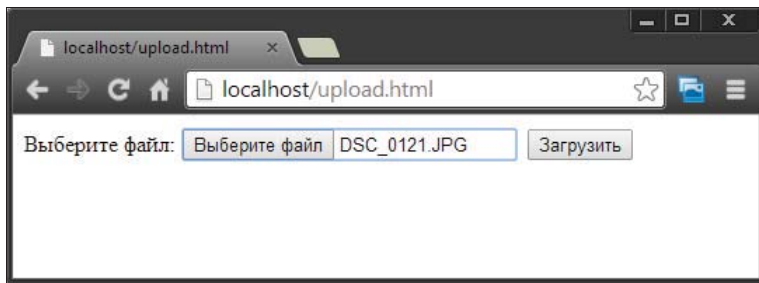


Рис. 29.1. Multipart-форма в окне браузера

Multipart-форма, безусловно, необходима. Но ее наличие еще совсем не гарантирует саму загрузку файлов на сервер. Для включения загрузки файлов нужно отредактировать файл конфигурации `php.ini`. Найдите в нем секцию `file uploads` и измените в ней следующие параметры:

- ☐ `file_uploads = On` — разрешает загрузку файлов на сервер;
- ☐ `upload_max_filesize = 2M` — устанавливает максимальный объем загружаемого файла;
- ☐ `upload_tmp_dir = /tmp` — создает временный каталог для загрузки файлов.

Для того чтобы изменения вступили в силу, нужно перезагрузить веб-сервер Apache.

Следует отметить, что редактирование файла конфигурации может понадобиться только на вашем компьютере для тестирования сценария, поскольку на сервере хостинг-провайдера обычно эти параметры уже установлены (это понятно, ведь хостинг-провайдер старается предоставить максимальное качество обслуживания). Если вы хотите убедиться в том, что ваш хостинг-провайдер разрешил загрузку файлов на сервер, создайте файл `test.php` (листинг 29.2), загрузите его на сервер и запустите в окне браузера. Найдите параметр `file_uploads`. Если он установлен в значение `On`, значит, загрузка файлов на сервер разрешена (рис. 29.2). Там же вы найдете имя временного каталога и максимальный размер загружаемого файла.

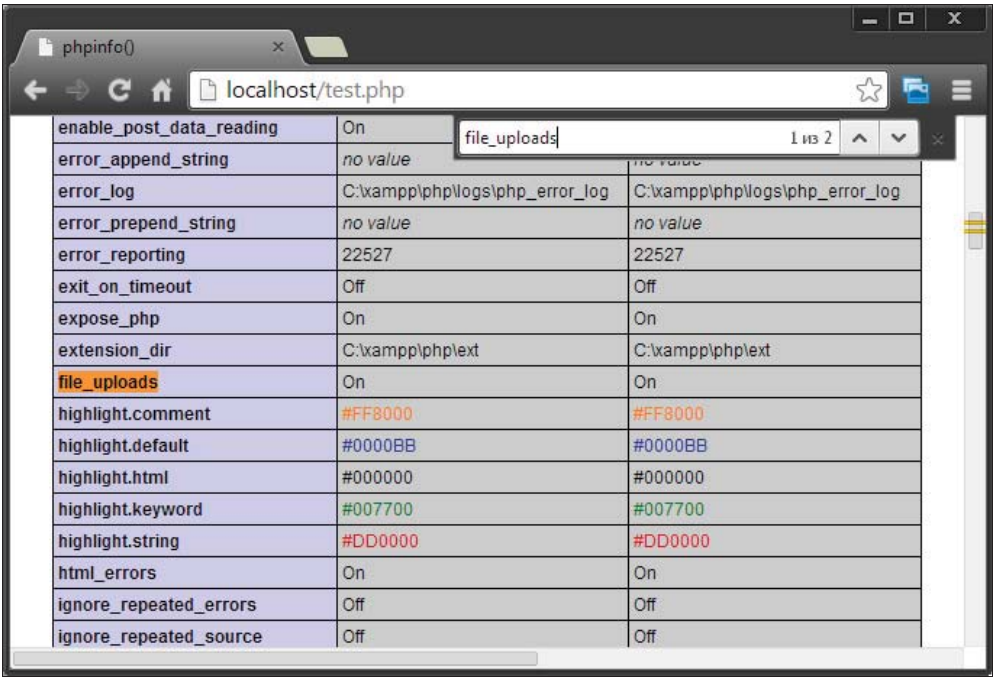


Рис. 29.2. Загрузка файлов включена

Листинг 29.2. Файл test.php

```
<?php
phpinfo();

?>
```

**ПРИМЕЧАНИЕ**

Код этого теста вы найдете в файле `29-2.php` каталога `Glava_29` сопровождающего книгу электронного архива (см. приложение 4).

Теперь, когда мы знаем, что загрузка файлов на сервер разрешена, и у нас есть multipart-форма, можно приступить непосредственно к самой загрузке файла.

## 29.2. Реализация загрузки файла

Задача multipart-формы заключается в передаче содержимого файла нашему сценарию. Должен вас обрадовать — PHP все сделает за вас, вам необходимо только обеспечить копирование файла в требуемый каталог.

В суперглобальном массиве `$_FILES` содержится информация о загружаемых файлах. Структура этого массива следующая:

- ❑ `$_FILES[имя_поля_input][tmp_name]` — PHP сохраняет принятые файлы во временном каталоге, в этом поле массива хранится имя временного файла;
- ❑ `$_FILES[имя_поля_input][name]` — имя файла на компьютере пользователя;
- ❑ `$_FILES[имя_поля_input][size]` — размер файла;
- ❑ `$_FILES[имя_поля_input][type]` — MIME-тип файла;
- ❑ `$_FILES[имя_поля_input][error]` — код ошибки (табл. 29.1).

Таблица 29.1. Коды ошибок при загрузке файлов

Код	Ошибка
0	Ошибок нет, файл загружен
1	Размер файла превышает максимальное значение, указанное с помощью параметра <code>upload_max_filesize</code> в <code>php.ini</code>
2	Размер файла превышает максимальное значение, указанное с помощью параметра <code>MAX_FILE_SIZE</code> в multipart-форме
3	Файл загружен не полностью (например, оборвалось соединение между сервером и клиентом)
4	Файл не загружен

Рассмотрим сценарий, реализующий загрузку файлов на сервер (листинг 29.3).

### Листинг 29.3. Сценарий загрузки файла на сервер `upload.php`

```
<?php

// каталог для загрузки файлов
$dir = './upload/';

// в multipart-форме мы определили имя input-поля upfile
// это имя нужно использовать при работе с массивом $_FILES
if(isset($_FILES["upfile"]))
{
    $upfile      = $_FILES["upfile"]["tmp_name"];
```

```
$upfile_name = $_FILES["upfile"]["name"];
$upfile_size = $_FILES["upfile"]["size"];
$upfile_type = $_FILES["upfile"]["type"];
$error_code = $_FILES["upfile"]["error"];

// если ошибок нет
if($error_code == 0)
{
    // выводим информацию о принятом файле
    echo "Имя файла на сервере: ".$upfile."<br>";
    echo "Имя файла на компьютере пользователя: ".$upfile_name."<br>";
    echo "MIME-тип файла: ".$upfile_type."<br>";
    echo "Размер файла: ".$upfile_size."<br><br>";

    // дополняем имя файла
    $upfile_name = $dir . $upfile_name;

    // копируем временный файл в каталог $dir, имя файла будет исходное,
    // т. е. как на компьютере у пользователя
    // первый параметр — источник
    // второй параметр — получатель
    copy($upfile,$upfile_name);

    // можно использовать функцию move_uploaded_file()
    //move_uploaded_file($upfile, $upfile_name);

}
}

?>
```

### ПРИМЕЧАНИЕ

Напомню, что этот код представлен в файле *upload.php* каталога *Glava\_29* сопровождающего книгу электронного архива (см. *приложение 4*).

Теперь проанализируем сценарий. Он будет копировать с помощью функции `copy()` принятый файл из временного каталога в каталог, заданный переменной `$dir`. Нужно позаботиться о том, чтобы этот каталог существовал, и установить для него права доступа 777:

```
chmod 777 ./upload/
```

Функция копирования `copy()`, в случае если файл-получатель существует, перезапишет его. Поэтому будьте осторожны — в реальном сценарии придется еще добавить функцию проверки существования файла-получателя и, если файл с заданным именем существует, к его имени дописывать произвольный символ.

Вместо `copy()` можно указать функцию `move_uploaded_file()`, которая не копирует, а перемещает файл. Особого смысла в этом я не вижу, поскольку серверы провайдера обычно настроены на автоматическую очистку временных каталогов.

## 29.3. Загрузка нескольких файлов

Возможно, вам потребуется реализовать автоматическую загрузку нескольких файлов. Особых проблем с этим нет. Просто измените multipart-форму так, как показано в листинге 29.4.

### Листинг 29.4. Форма для одновременной загрузки трех файлов

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
```

Выберите файлы:

```
<p><INPUT NAME="upfile1" TYPE="file">
```

```
<p><INPUT NAME="upfile2" TYPE="file">
```

```
<p><INPUT NAME="upfile3" TYPE="file">
```

```
<INPUT TYPE="submit" VALUE="Загрузить">
```

```
</FORM>
```

#### ПРИМЕЧАНИЕ

Код этой формы вы найдете в файле *29-4.html* каталога *Glava\_29* сопровождающего книгу электронного архива (см. приложение 4).

При этом массив `$_FILES` будет выглядеть так:

- ☐ `$_FILES["upfile1"]["tmp_name"]` — имя первого временного файла;
- ☐ `$_FILES["upfile1"]["name"]` — исходное имя первого файла;
- ☐ `$_FILES["upfile1"]["size"]` — размер первого файла;
- ☐ `$_FILES["upfile1"]["type"]` — MIME-тип первого файла;
- ☐ `$_FILES["upfile1"]["error"]` — код ошибки для первого файла;
- ☐ `$_FILES["upfile2"]["tmp_name"]` — имя второго временного файла;
- ☐ `$_FILES["upfile2"]["name"]` — исходное имя второго файла;
- ☐ `$_FILES["upfile2"]["size"]` — размер второго файла;
- ☐ `$_FILES["upfile2"]["type"]` — MIME-тип второго файла;
- ☐ `$_FILES["upfile2"]["error"]` — код ошибки для второго файла;
- ☐ `$_FILES["upfile3"]["tmp_name"]` — имя третьего временного файла;
- ☐ `$_FILES["upfile3"]["name"]` — исходное имя второго файла;
- ☐ `$_FILES["upfile3"]["size"]` — размер третьего файла;
- ☐ `$_FILES["upfile3"]["type"]` — MIME-тип третьего файла;
- ☐ `$_FILES["upfile3"]["error"]` — код ошибки для третьего файла.

Конечно, в программе придется анализировать, передал пользователь файл или нет. Ведь форма рассчитана на передачу трех файлов, а пользователь мог передать

только два или вообще один файл. Проверить факт передачи файла можно или по имени временного файла, или по коду ошибки (листинг 29.5).

**Листинг 29.5. Проверка факта передачи файла**

```
...
if(isset($_FILES["upfile1"]))
{

    if ($_FILES["upfile1"]["error"] == 0) {

        // первый файл загружен, копируем
    }

}

...
```

**ПРИМЕЧАНИЕ**

Код этого фрагмента вы найдете в файле *29-5.txt* каталога *Glava\_29* сопровождающего книгу электронного архива (см. приложение 4).

Также нужно решить, под каким именем сохранять файл. Можно использовать имя временного файла, которое автоматически присваивает загруженному файлу интерпретатор PHP. В этом случае имя файла будет иметь примерно такой вид: *php05hhGi*. Первые три символа — это *php*, а остальные будут сгенерированы случайным образом. Вот пример кода, реализующего такую схему именования файлов (пример загрузки фотографий в галерею):

```
if(isset($_FILES["upfile"]))
{
    // имя временного файла
    $upfile = $_FILES["upfile"]["tmp_name"];

    // если ошибок нет
    if($error_code == 0)
    {
        // каталог для загрузки фото
        $d = './fotos/';
        // дополняем имя файла
        $upfile_name = $d . basename($upfile) . ".jpg";

        // перемещаем файл
        move_uploaded_file($upfile, $upfile_name);

        echo "<p>Загруженное фото:";
        echo "<img border=0 width=640
src=$upfile_name>";
    }
}
```

Но, как показывает практика, когда количество картинок большое, имена файлов начинают повторяться намного раньше, чем нам бы этого хотелось. В итоге одна картинка перезаписывает другую, чего делать нельзя. Можно проверять существование временного файла и, если такой файл уже существует, пытаться добавить к имени нового файла какой-то символ, сгенерированный случайным образом, — например, число от 1 до 99. Но это не очень хороший вариант — дополнительные проверки, операторы, все это увеличит время выполнения РНР-сценария, которое и так ограничено. Нужно сразу сгенерировать случайное имя, которое будет уникальным с вероятностью 99 %.

Я предлагаю добавлять к имени временного РНР-файла временную метку (timestamp). В результате получится, что в имени файла скомбинируются случайно сгенерированная последовательность символов (ее генерирует РНР) и временная метка. Вероятность того, что будут сгенерированы две одинаковые последовательности символов в один и тот же момент времени, практически исключена. Вот пример генерирования уникального имени файла:

```
$tm = time();  
$upfile_name = $dir . basename($upfile) . $tm . ".jpg";
```

## 29.4. Индикатор загрузки файла

### 29.4.1. Некоторые теоретические предпосылки

В РНР 5.4 появился индикатор загрузки файла, и здесь будет показано, как его использовать. Ранее такой возможности не было и приходилось обращаться к сторонним решениям — например, к модулям `uploadprogress`<sup>1</sup> или `APM`<sup>2</sup>. Задача была не из легких и требовала даже модификации конфигурационного файла Apache (нужно было прописать в нем дополнительные модули). На своем сервере это сделать несложно, а вот уговорить хостинг-провайдера подключить дополнительные модули к своему веб-серверу удавалось не всегда.

С выходом РНР 5.4 ситуация с индикатором загрузки стала лучше. Да, в РНР наконец-то появился механизм, способный контролировать процесс загрузки файлов, и он стал частью механизма сессий. Однако создание рабочего индикатора загрузки (прогресс-бара) все равно остается далеко не легкой задачей, и одного только РНР здесь мало — нужно как минимум подключать к работе и AJAX. Но хоть не нужно изменять конфигурацию веб-сервера — и на том спасибо!

Прежде чем приступить к реализации индикатора загрузки, рассмотрим табл. 29.2, в которой приведены конфигурационные параметры механизма контроля процесса загрузки.

---

<sup>1</sup> <http://pecl.php.net/package/uploadprogress>.

<sup>2</sup> <http://habrahabr.ru/post/17620/>.

Таблица 29.2. Необходимые параметры

Параметр	Описание
session.upload_progress.enabled	Включает/выключает механизм контроля процесса загрузки. Значение по умолчанию: 1 (механизм включен)
session.upload_progress.cleanup	Включает/выключает очистку данных в сессии после загрузки файла. Значение по умолчанию: 1 (очистка включена)
session.upload_progress.prefix	Задаёт префикс переменной в сессии. Значение по умолчанию: upload_progress_
session.upload_progress.name	Название элемента массива \$_POST, в котором передается название переменной сессии, использующейся для хранения информации о загружаемых файлах. Если такой переменной в \$_POST не будет, механизм контроля процесса загрузки окажется недоступен! Значение по умолчанию: PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.freq	Частота обновления данных в сессии. Можно задать значения в байтах, а можно в процентах от общего размера загружаемого файла. По умолчанию частота обновления: 1 %
session.upload_progress.min_freq	Минимальная задержка в секундах между обновлениями данных. По умолчанию: 1 секунда

Если вы внимательно изучили табл. 29.2, то вам будет понятно, что механизм контроля процесса загрузки файла станет доступен, когда выполняются следующие условия:

- включен параметр session.upload\_progress.enabled;
- массив \$\_POST содержит элемент с названием session.upload\_progress.name;
- значение только что упомянутого элемента массива \$\_POST не пустое.

Для выполнения этих условий в текущей сессии создается переменная, название которой формируется из значения параметра session.upload\_progress.prefix и значения элемента с названием session.upload\_progress.name массива \$\_POST, а именно:

```
$SESSION[ini_get(session.upload_progress.prefix).  
$_POST[ini_get(session.upload_progress.name)]]
```

В эту переменную в виде ассоциативного массива сохраняется информация о процессе загрузки и о самих загружаемых файлах. Структура массива приведена в листинге 29.6.

Листинг 29.6. Структура массива переменной с данными о загружаемых файлах

```
$SESSION[ini_get(session.upload_progress.prefix).$_POST[ini_get(session.  
upload_progress.name)]] = array(  
    'start_time'           => дата начала загрузки,  
    'content_length'       => размер загружаемых данных,
```

```

'bytes_processed'      => кол-во принятых и обработанных байтов,
'done'                 => флаг завершения загрузки (true/false)
// Информация о загружаемых файлах
'files' => array(
    // Информация о первом файле
    0 => array(
        'field_name'      => название поля в форме,
        'name'            => имя файла,
        'tmp_name'         => имя файла во временном каталоге,
        'error'            => код ошибки,
        'done'             => загружен?(true/false),
        'start_time'       => время начала загрузки этого файла,
        'bytes_processed'  => кол-во принятых и обработанных байтов,
    ),
    // Информация о втором файле и т.д.
    1 => array(
        ...
    ),
    n => array(
        ...
    ),
)
);

```

#### ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-6.txt* каталога *Glava\_29* сопровождающего книгу электронного архива (см. *приложение 4*).

Представим, что у нас есть форма загрузки двух файлов, код которой приведен в листинге 29.7.

#### Листинг 29.7. Форма загрузки двух файлов

```

<form action="" method="POST" enctype="multipart/form-data">
    <input type="hidden" name="<?php echo
ini_get("session.upload_progress.name"); ?>" value="pbar" />
    <label for="file1">Файл 1: </label><input type="file" name="file1" />
    <label for="file2">Файл 2: </label><input type="file" name="file2" />
    <input type="submit" value="Загрузить" />
</form>

```

#### ПРИМЕЧАНИЕ

Код этого листинга вы найдете в файле *29-7.html* каталога *Glava\_29* сопровождающего книгу электронного архива (см. *приложение 4*).

Форма как форма. Очень похожа на обычную форму загрузки файлов, но обратите внимание на скрытое поле, имя которого формирует PHP-оператор `echo ini_get("session.upload_progress.name")`.

Согласно нашей форме при загрузке файлов в сессии будет сохранена следующая информация:

```
$_SESSION['upload_progress_test'] = array(5) {
  "start_time" => 1391929280,
  "content_length" => 4294304,
  "bytes_processed" => 4294304,
  "done" => true,
  "files" => array(2) {
    [0] => array(7) {
      "field_name" => "file1",
      "name" => "DSC_0101.JPG",
      "tmp_name" => "/tmp/phpky7Mqc",
      "error" => 0,
      "done" => true,
      "start_time" => 1391929280,
      "bytes_processed" => 2097152
    },
    [1] => array(7) {
      "field_name" => "file2",
      "name" => "DSC_0102.JPG",
      "tmp_name" => "/tmp/phpDZvS31",
      "error" => 0,
      "done" => true,
      "start_time" => 1391929280,
      "bytes_processed" => 2197152
    },
  },
}
```

Из этого массива становится понятным, что загружалось два (количество элементов в массиве `files`) файла (судя по именам — фотографии) общим размером чуть больше 4 Мбайт, оба файла успешно загружены, о чем свидетельствует общий флаг `done` и флаги `done` каждого из файлов. Прежде чем перейти к практической части, нельзя не упомянуть следующую переменную:

```
$_SESSION['upload_progress_test']['cancel_upload']
```

Если ей присвоить `true`, процесс загрузки файлов будет прерван. Эту возможность можно использовать для отмены загрузки файлов.

## 29.4.2. Пример практической реализации

Как уже было упомянуто мною ранее, к сожалению, реализовать индикатор загрузки только средствами PHP не получится, поэтому мы воспользуемся плагином `form`<sup>1</sup>

---

<sup>1</sup> <http://jquery.malsup.com/form>.

для библиотеки jQuery<sup>1</sup>. Да, без AJAX и JavaScript не обойтись, поэтому я надеюсь, что у читателя есть хотя бы небольшой опыт программирования на JavaScript.

Кроме плагина form и библиотеки jQuery нам также понадобится библиотека jQuery-UI<sup>2</sup>.

Создайте на своем сервере каталог upload (в корневом каталоге документов, в случае с XAMPP — это каталог htdocs). В какой-нибудь другой каталог распакуйте jQuery, form и jQuery-UI. В каталог upload нужно поместить следующие файлы:

- ☐ jquery-версия.min.js — библиотека jQuery;
- ☐ jquery-ui-версия.custom.min.js — библиотека jQuery UI;
- ☐ jquery-ui-версия.custom.css — таблица стилей jQuery-UI;
- ☐ jquery.form.js — плагин form.

Далее в каталоге upload создайте подкаталог images. В него поместите GIF-файл, который будет использоваться для заполнения индикатора загрузки. Для тестирования можете взять файл progress.gif из каталога Glava\_29 прилагаемого диска.

Наш сценарий состоит из трех следующих файлов. Основной файл — index.php — выводит форму загрузки и отображает процесс загрузки. Ход загрузки файла будет получен от сценария progress.php. Третий файл — cancel.php — отменяет загрузку файлов.

Этот сценарий будет загружать файлы только во временный каталог и отображать ход загрузки. Вам нужно будет доработать его и скопировать все загруженные файлы в нужный каталог. Это можно сделать с помощью функции copy(), как было показано ранее в этой главе.

Большая часть файла index.php (листинг 29.8) состоит из JavaScript и HTML-кода. PHP-кода там совсем мало. Перед использованием сценария убедитесь, что вы прописали ваши номера версий jQuery и jQuery-UI.

#### Листинг 29.8. Файл index.php

```
<?php
// Вот и весь PHP-код. Просто запускаем сессию
session_start();
?>
<html>
<head>
<script type="text/javascript" src="jquery-1.x.x.min.js"></script>
<script type="text/javascript" src="jquery-ui-1.x.x.custom.min.js"></script>
<script type="text/javascript" src="jquery.form.js"></script>
<link href="jquery-ui-1.8.14.custom.css" rel="stylesheet" type="text/css" />
<link href="pbar.css" rel="stylesheet" type="text/css" />
<script>
```

---

<sup>1</sup> <http://jquery.com/>.

<sup>2</sup> <https://jqueryui.com/>.

```

var t;
progress = function(){
    $.ajax({
        url: 'progress.php',
        dataType: 'json',
        success: function(data) {
            if(data.percent) {
                $("#bar").progressbar({
                    value: Math.ceil(data.percent),
                });
                $('#ui-progressbar-value').text(data.percent+'%');
            }
        }
    });
}

$(document).ready(function() {
    $('#form').ajaxForm(
        {
            type: 'POST',
            success: function() {
                clearTimeout(t);
                $('#progress').html('<b>Готово!</b>');
            },
            beforeSubmit: function() {
                $('#upload_form').hide();
                $('#progress').show();
                // Интервал 100 мс
                t = setInterval("progress()", 100);
            }
        }
    );
    $('#cancel-form').ajaxForm(
        {
            success: function() {
                clearTimeout(t);
                $('#progress').html('<b>Вы отменили загрузку!</b>');
            }
        }
    );
});
</script>
</head>
<body>
<div id="upload">
<div id="upload_form">
<form id="form" action="progress.php" method="POST" enctype="multipart/
form-data">
<input type="hidden" name="<?php echo ini_get("session.upload_progress.name");
?>" value="pbar" />

```

```

<label for="file1">Файл 1:&nbsp;  </label>
<input type="file" name="file1" /><br /><br />
<label for="file2">Файл 2:&nbsp;  </label>
<input type="file" name="file2" /><br /><br />
<input type="submit" value="Загрузить" />
</form>
</div>

<div id="progress">
Загрузка файлов<br /><br /><div id="bar"></div><br />
<form id="cancel-form" action="cancel.php" method="POST"
enctype="multipart/form-data">
<input type="submit" value="Отмена" />
</form>
</div>
<div id="result"></div>
</div>
</body>
</html>

```

### ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *29-8.php* каталога *Glava\_29* сопровождающего книгу электронного архива (см. *приложение 4*).

Теперь разберемся, что здесь что. Самая главная функция — функция `progress`. Она вызывает файл `progress.php` и получает от него информацию о ходе загрузки файла. Данные отправляются в формате JSON.

После JavaScript-кода начинается HTML-код, в котором вы видите две формы: одна форма загрузки двух файлов, вторая — форма отмены загрузки. Также есть три компонента `<DIV>`: `upload_form` (форма загрузки), `progress` (будет содержать индикатор загрузки) и `result` (результат загрузки). Код файла `progress.php` представлен в листинге 29.9.

### Листинг 29.9. Файл `progress.php`

```

<?php
session_start();
$p = 0;                // Процент загрузки файлов
$data = array();

if(isset($_SESSION['upload_progress_pbar']) and
is_array($_SESSION['upload_progress_pbar'])) {
    // Вычисляем процент загрузки
    $p = ( $_SESSION['upload_progress_pbar']['bytes_processed'] * 100 ) /
    $_SESSION['upload_progress_pbar']['content_length'];

    $p = round($p,2);    // Округляем процент
}

```

```
// Формируем массив данных
$data = array(
    'percent' => $p,
    'content_length' => $_SESSION['upload_progress_pbar']['content_length'],
    'bytes_processed' => $_SESSION['upload_progress_pbar']['bytes_processed']);
}

// Отправляем данные в JSON-формате
echo json_encode($data);
?>
```

**ПРИМЕЧАНИЕ**

Код этого листинга вы найдете в файле *29-9.php* каталога *Glava\_29* сопровождающего книгу электронного архива (см. приложение 4).

Думаю, тут все понятно. Сценарий просто получает данные о загрузке из сессии, вычисляет процент загрузки и передает эту информацию в JSON-формате вызывающему JavaScript-сценарию.

Код файла *cancel.php* приведен в листинге 29.10. Этот файл — самый простой.

**Листинг 29.10. Файл *cancel.php***

```
<?php
session_start();
$_SESSION['upload_progress_pbar']['cancel_upload'] = true;
?>
```

**ПРИМЕЧАНИЕ**

Код этого листинга вы найдете в файле *29-10.php* каталога *Glava\_29* сопровождающего книгу электронного архива (см. приложение 4).

Осталось рассмотреть еще один файл — таблицу стилей *pbar.css*, определяющую внешний вид индикатора загрузки (листинг 29.11).

**Листинг 29.11. Таблица стилей *pbar.css***

```
.ui-progressbar-value {
    background-image:
        url(images/progress.gif);
    padding-left:10px;
    font-weight:normal;
}

#upload_form {
    display:block;
}
```

```
#progress {  
    display: none;  
}  
  
#progress #bar {  
    height: 25px;  
    width: 350px;  
}
```

### ПРИМЕЧАНИЕ

Код этой таблицы вы найдете в файле 29-11.css каталога *Glava\_29* сопровождающего книгу электронного архива (см. приложение 4).

Теперь посмотрим, как все это работает. Откройте браузер, запустите сценарий `index.php`, выберите файлы и нажмите кнопку **Загрузить** (рис. 29.3) — вы увидите индикатор загрузки файла, вот только приготовьтесь к тому, что при загрузке небольших файлов на локальный сервер все происходит достаточно быстро, настолько быстро, что этот индикатор вы можете и не заметить. На рис. 29.4 изображен индикатор загрузки файла в процессе работы.

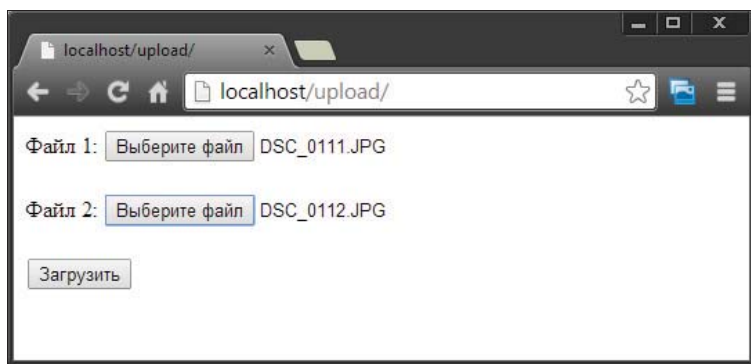


Рис. 29.3. Форма загрузки файлов

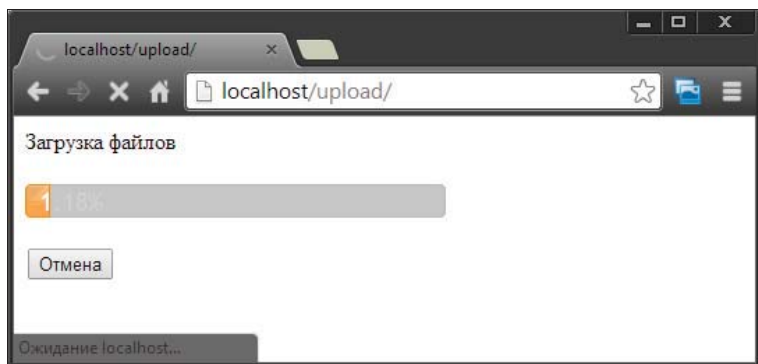


Рис. 29.4. Индикатор процесса загрузки файлов

Для удобства отладки давайте произведем некоторые изменения в нашей программе (чтобы вы убедились, что все работает). Для начала в файле `pbar.css` изменим описание стиля `#progress` так:

```
#progress {  
}
```

Другими словами, мы не будем скрывать форму загрузки. Далее в файле `index.php` закомментируйте строчку `$('#upload_form').hide();`:

```
//$('#upload_form').hide();
```

Это даст возможность не скрывать форму загрузки в процессе. Далее заменим строку:

```
$('#progress').html('<b>Готово!</b>');
```

следующей строкой:

```
$('#result').html('<b>Готово!</b>');
```

Исходная строка перезаписывала область индикатора процесса загрузки по окончании процесса загрузки, новая строка перенаправляет вывод строки **Готово** в область результата, не перезаписывая индикатора загрузки. В результате страничка загрузки файла будет выглядеть так, как показано на рис. 29.5. На ней видны все три области: форма загрузки, индикатор загрузки и область результата.

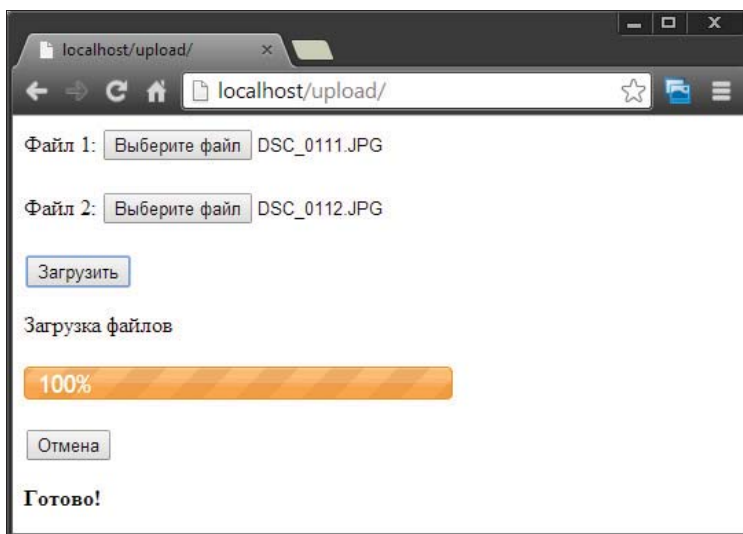


Рис. 29.5. Страница загрузки файлов (режим отладки)

Теперь расскажу, как организовать копирование файлов в нужный вам каталог (сейчас они загружаются во временный каталог и удаляются по завершении работы сценария).

Файл `progress.php` должен использоваться только для отображения процесса загрузки и должен быть определен только в JavaScript-функции:

```
progress = function(){  
    $.ajax({  
        url: 'progress.php',
```

В форме загрузки вместо файла `progress.php` следует использовать другой файл — например, `upload.php`:

```
<form id="form" action="upload.php" method="POST" enctype="multipart/  
form-data">  
<input type="hidden" name="<?php echo ini_get("session.upload_progress.name");  
?>" value="pbar" />
```

Далее — дело техники. Вам нужно написать сценарий `upload.php`, копирующий загруженные файлы из временного каталога в нужный. Пример такого файла уже был приведен ранее в этой главе (см. листинг 29.3).

## 29.5. Проблемы при загрузке файлов

Если вы сделали все, как здесь было описано, трудностей с загрузкой файлов у вас быть не должно. Но все же могут возникнуть две проблемы:

- ❑ файл загружается на сервер, его можно прочитать (ведь у нас есть имя временного файла) и вывести в браузер, но невозможно скопировать в заданный каталог. Скорее всего, это проблема с правами доступа — нужно просто установить права доступа `777` для каталога, в который вы хотите скопировать файл. Это можно сделать с помощью любого FTP-клиента или файлового менеджера панели управления хостингом. Правильнее, конечно, выяснить пользователя, от имени которого работает веб-сервер, и предоставить ему соответствующие права, но права `777` выручают в случаях, когда нужно исправить ошибку с загрузкой немедленно, хотя и являются не очень безопасным вариантом;
- ❑ файл не загружается по причине превышения установленного размера файла для загрузки. Проверьте значение директивы `upload_max_filesize` в файле `php.ini` используемой конфигурации (обычно он находится по пути: `/etc/php/<номер версии>/apache2/php.ini`. Если у вас хостинг, возможность изменения этой настройки, как правило, есть в панели управления хостингом в разделе **Настройки PHP** или аналогичном.
- ❑ файл загружается поврежденным — в этом случае ошибка связана с тем, что Apache пытается перекодировать бинарные файлы, — например, символ с кодом `0x00` он заменяет на символ `0x20` (пробел). Для решения этой проблемы нужно выключить параметр `CharsetRecodeMultipartForms` в файле конфигурации Apache `apache2.conf`:

```
CharsetRecodeMultipartForms Off
```



## ГЛАВА 30

# Использование FTP-функций

### 30.1. Функции для работы с FTP

Непонятно почему, но большинство программистов предпочитают использовать для работы с FTP-сервером сокеты, хотя PHP обладает встроенными функциями, позволяющими работать с FTP. Может быть, причина в том, что немногие знают об этих функциях, поскольку во многих самоучителях они не упомянуты, зато описаны сокеты. Возможно, есть и какая-то другая причина такого положения. Если вы также никогда не слышали о FTP-функциях, это поправимо.

Для подключения к FTP-серверу предусмотрена функция `ftp_connect()`:

```
resource ftp_connect (string host [, int port [, int timeout]])
```

Параметр `host` задает имя сервера, остальные два параметра можно не указывать. Второй параметр — это номер порта, по умолчанию установлено значение 21, а третий — тайм-аут в секундах (по умолчанию — 90).

После подключения к FTP-серверу на нем необходимо зарегистрироваться, т. е. передать имя пользователя и пароль. Для этого служит функция `ftp_login()`:

```
bool ftp_login (resource ftp_stream,  
               string username,  
               string password)
```

Этой функции нужно передать три параметра: идентификатор соединения (его возвращает функция `ftp_connect()`), имя пользователя и пароль.

После успешной регистрации можно осуществлять операции над файлами и каталогами.

Начнем с каталогов. Для изменения каталога предназначена функция `ftp_chdir()`:

```
bool ftp_chdir (resource ftp_stream, string directory)
```

Первый параметр задает идентификатор соединения, а второй — нужный нам каталог. Функция возвращает `true`, если каталог изменен успешно, `false` — если перейти в каталог не удалось. Помните, что не в каждый каталог можно перейти. Для перехода в каталог вы должны обладать правами доступа к нему.

Имя текущего каталога можно узнать с помощью функции `ftp_pwd()`:

```
string ftp_pwd (resource ftp_stream)
```

Для создания каталога служит функция `ftp_mkdir()`, которой передается идентификатор соединения и имя каталога:

```
string ftp_mkdir (resource ftp_stream, string directory)
```

Для создания каталога вы должны обладать необходимыми правами доступа. В случае успеха функция возвращает `true`.

Удалить каталог можно функцией `ftp_rmdir()` с такими же параметрами:

```
bool ftp_rmdir (resource ftp_stream, string directory)
```

Для просмотра содержимого каталога предусмотрена функция `ftp_nlist()`:

```
array ftp_nlist (resource ftp_stream, string directory)
```

Эта функция возвращает массив, содержащий имена файлов и каталогов в указанном каталоге.

Теперь перейдем к функциям для работы с файлами удаленного FTP-сервера. Чаще всего вам понадобятся функции `ftp_fget()` и `ftp_fput()`:

```
bool ftp_fget (resource ftp_stream,  
               resource fp,  
               string remote_file,  
               int mode)  
  
bool ftp_fput (resource ftp_stream,  
               string remote_file,  
               resource fp,  
               int mode)
```

Первая функция загружает файл с сервера, вторая закачивает файл на сервер. Первый параметр обеих функций — это идентификатор соединения.

Последний параметр — это режим передачи файла:

☐ `FTP_ASCII` — текстовый;

☐ `FTP_BINARY` — двоичный.

Двоичный режим рекомендуется устанавливать даже для передачи текстовых файлов.

Параметр `fp` функции `ftp_fget()` задает указатель файла, в который должен закачаться принимаемый файл. Файловый указатель возвращается функцией `fopen()` при открытии/создании файла. Параметр `remote_file` задает имя файла на удаленном сервере.

Аналогично параметр `ftp` функции `ftp_fput()` задает указатель файла, который будет прочитан и передан на сервер. Имя файла, в который будут записаны прочитанные из `fp` данные, определяется параметром `remote_file` функции `ftp_fput()`.

Функции `ftp_fget()` и `ftp_fput()`, конечно, хороши, но несколько неудобны. Ведь нужно открывать файл, а перед этим проверять его существование, передавать

файловый указатель... Намного проще указать имена локального и удаленного файлов. Это позволяют создать функции `ftp_get()` и `ftp_put()`, существенно упрощающие обмен файлами между компьютерами по FTP.

```
bool ftp_get (resource ftp_stream,
             string local_file,
             string remote_file,
             int mode)

bool ftp_put (resource ftp_stream,
             string remote_file,
             string local_file,
             int mode)
```

Тут все просто: первая функция получает файл с FTP-сервера, а вторая — закачивает файл на FTP-сервер. Параметр `local_file` задает имя локального файла, а `remote_file` — удаленного. Параметр `mode` определяет режим передачи файла. Как видите, все намного проще — нам не нужны больше файловые указатели.

Для переименования файла на FTP-сервере служит функция `ftp_rename()`:

```
bool ftp_rename (resource ftp_stream, string from, string to)
```

Параметр `from` задает старое имя файла, а `to` — новое.

Чтобы удалить файл, нужно вызвать функцию `ftp_delete()`, которой следует передать идентификатор FTP-соединения и имя файла:

```
bool ftp_delete (resource ftp_stream, string path)
```

Функция `file_size()` возвращает размер файла на FTP-сервере (полезно узнать размер перед скачиванием):

```
int ftp_size (resource ftp_stream, string remote_file)
```

Мы рассмотрели практически все необходимые FTP-функции. Упомянем еще три. Довольно полезной является функция `ftp_exec()`, позволяющая выполнять команды на FTP-сервере:

```
bool ftp_exec (resource ftp_stream, string command)
```

Функция возвращает `true`, если команду удалось выполнить:

```
if (ftp_exec($conn, 'chmod 777 images'))
    echo "Права доступа изменены успешно";
else
    echo "Ошибка при изменении прав доступа";
```

Протокол FTP предусматривает два режима работы: активный и пассивный. Разница между ними заключается в том, что в пассивном режиме клиент всегда иницирует соединение, а в активном инициатором соединения может быть FTP-сервер.

В активном режиме FTP-соединение выглядит так: клиент подключается к серверу и передает ему имя пользователя, пароль и номер порта (из динамического диапазона 1024–65 535). FTP-сервер подключается к указанному клиентом порту, а со своей стороны он использует TCP-порт 20.

В пассивном режиме сервер назначает номер порта (из диапазона 1024–65 535), к которому должен подключиться клиент для передачи данных. Таким образом, не сервер подключается к клиенту, а клиент к серверу. Пассивный режим полезен, когда брандмауэр блокирует входящие соединения.

Для перехода в пассивный режим служит функция `ftp_pasv()`:

```
bool ftp_pasv (resource ftp_stream, bool pasv)
```

Этой функции нужно передать идентификатор соединения и параметр `pasv`, определяющий режим работы FTP:

❑ `true` — пассивный;

❑ `false` — активный.

Осталось рассмотреть еще одну функцию — `ftp_close()`, которая закрывает FTP-соединение. Вообще, FTP-соединение будет закрыто автоматически при завершении работы сценария, но вызова этой функции требует культура программирования на PHP:

```
void ftp_close (resource ftp_stream)
```

При работе с FTP помните о максимальном времени выполнения сценария, которое задается в файле `php.ini`. Предположим, что это время равно 30 с (по умолчанию), а файл, который нужно передать, «весит» 1 Мбайт. Следовательно, между вами и FTP-сервером должно быть соединение со скоростью не менее 35 Кбайт/с, чтобы уложиться в отведенные 30 с. А учитывая то, что на установку соединения с сервером также требуется время, то скорость передачи данных должна быть не менее 36–40 Кбайт/с.

## 30.2. Примеры использования FTP-функций

Рассмотрим два примера. Первый пример (листинг 30.1) будет подключаться к FTP-серверу, регистрироваться и выводить содержимое текущего каталога (рис. 30.1). Второй пример будет загружать файл с сервера (листинг 30.2).

### Листинг 30.1. Вывод содержимого каталога

```
<?php

// устанавливаем соединение с сервером ftp.example.com
$conn = ftp_connect("ftp.example.com");

// передаем имя пользователя и пароль
$login = ftp_login($conn, "user", "dfh45h2");

// подключаемся к серверу
if ((!$conn) || (!$login))
    die("Ошибка подключения к FTP-серверу");
```

```
else
    echo "Успешно подключились к серверу!";

echo "<p>Выводим содержимое текущего каталога";
// получаем список файлов и каталогов FTP-сервера
$A = ftp_nlist($conn, ftp_pwd($conn));

// выводим список файлов и каталогов
foreach($A as $a) echo "<br>$a";

// отключаемся
ftp_close($conn);

?>
```



```
Успешно подключились к серверу!

Выводим содержимое текущего каталога
./
../
/logs
/login
/mail_aliases
/login_conf
.rhosts
/mailrc
/profile
/.shrc
.shadow
/domains
/tmp
/backups
/mailboxlist
.spamassassin
/Maildir
/upload
/imap
/public_html
```

Рис. 30.1. Результат работы сценария (листинг 30.1)

Листинг 30.2 иллюстрирует пример загрузки файла с сервера.

#### Листинг 30.2. Пример загрузки файла с FTP-сервера

```
<?php

// устанавливаем соединение с сервером ftp.example.com
$conn = ftp_connect("ftp.example.com");
```

```
// передаем имя пользователя и пароль
$login = ftp_login($conn, "anonymous", "user@mail.ru");

// подключаемся к серверу
if (!$conn || !$login)
    die("Ошибка подключения к FTP-серверу");
else
    echo "Успешно подключились к серверу!";

// загружаем удаленный файл ftp://ftp.example.com/pub/soft.rar
if (ftp_get($conn, "soft.rar", "/pub/soft.rar", FTP_BINARY))
    echo "Файл загружен успешно";
else
    echo "Произошла ошибка";

// отключаемся
ftp_close($conn);

?>
```

### **ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода листингов 30.1 и 30.2, настоятельно рекомендую использовать уже готовые варианты (файлы *30-1.php* и *30-1.php* соответственно), представленные в каталоге *Glava\_30* сопровождающего книгу электронного архива (см. приложение 4).



## ГЛАВА 31

# Отправка и прием почты

## 31.1. Отправка почты средствами PHP: функция *mail()*

### 31.1.1. Использование функции

Отослать подтверждение заказа, подтвердить забытый пароль, задать вопрос администрации сайта или поздравить пользователя с днем рождения — для всего этого (и не только) потребуется функция отправки электронных сообщений. В PHP для этой цели есть простая и удобная функция `mail()`:

```
mail(string email, string subject, string message [,string headers]);
```

Первый параметр функции — это электронный адрес получателя, второй — тема сообщения, третий — само сообщение, а четвертый — заголовки сообщения. По умолчанию заголовки указывать не обязательно, но иногда без них не обойтись. Вот самые полезные заголовки:

- ☐ `From` — содержит адрес отправителя (нужно же получателю знать, от кого он получил письмо?);
- ☐ `Reply-To` — позволяет указать адрес для ответа на сообщение;
- ☐ `Content-Type` — позволяет указать тип содержимого, например `text/plain`;
- ☐ `Content-Transfer-Encoding` — указывает, сколько битов используется для передачи символа. Для отправки сообщений на русском языке необходимо 8 битов.

Наиболее важны заголовки `Content-Type` и `Content-Transfer-Encoding`. Как уже было отмечено, первый позволяет определить тип содержимого, а второй — определить, как будут перекодироваться символы.

Для нормальной отправки сообщений на русском языке в кодировке UTF-8 нужно передать два следующих заголовка:

```
Content-Type: text/plain; charset=utf-8  
Content-Transfer-Encoding: 8bit
```

Можно также использовать механизм конвертации `Quoted-Printable`, гарантирующий целостность вашего содержимого при прохождении через различные почтовые шлюзы, производящие языковую перекодировку символов:

```
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: quoted-printable
```

**Пример:**

```
mail('user@host.ru', 'Привет', 'Привет! Как дела!' ,
'Content-Type: text/plain; charset=utf-8\n
Content-Transfer-Encoding: quoted-printable\n');
```

После каждого заголовка не забываем указывать символ новой строки — `\n`.

Если вы тестируете свой сценарий под Windows, то функция `mail()`, скорее всего, работать у вас не будет. Для того чтобы она все-таки заработала, следует отредактировать файл `php.ini`. Откройте его и найдите секцию `mail function`. Параметру `SMTP` нужно присвоить имя (или IP-адрес) SMTP-сервера, а параметру `smtp_port` — номер порта почтового сервера (обычно 25):

```
[mail function]
; For Win32 only.
SMTP = localhost
smtp_port = 25
```

С помощью функции `mail()` можно отправлять сообщения в формате HTML, для этого просто нужно указать тип содержимого `Content-Type: text/html`:

```
$mailto = 'user@host.ru';
$message = '<html><body><h1>Привет, Вася!</h1></body></html>';
$header = 'Content-Type: text/html; charset=Windows-1251\n';
$header .= 'Content-Transfer-Encoding: quoted-printable\n';
```

```
mail($mailto, $message, $header);
```

Обратите внимание, как присваивается значение второго заголовка! Не потеряйте точку!

Если вникнуть в принципы MIME-кодирования, то с помощью функции `mail()` даже можно отправлять электронные письма с вложениями. Но тратить время и силы на это не хочется, особенно когда существуют готовые решения. Далее мы рассмотрим класс `PHPMailer`, позволяющий отправлять сообщения в формате HTML с вложениями, закодированными с помощью MIME.

### 31.1.2. Подробно о настройке сервера

На арендуемых хостингах, как правило, настройка отправки почты уже выполнена службой поддержки, и вам остается только использовать функцию `mail()`. Когда же вы арендуете VDS под управлением Linux, необходимо произвести настройку сервера, чтобы функция `mail()` заработала.

Первым делом откройте ваш файл `php.ini` и отредактируйте значение директивы `sendmail_path`:

```
sendmail_path = /usr/sbin/ssmtp -t
```

Сохраните файл `php.ini`, перезапустите сервис `apache2` (или `php-fpm`, если у вас `nginx`). Больше к настройкам PHP мы возвращаться не будем. Теперь установим приложение `ssmtp`:

```
sudo apt install ssmtp
```

Откройте файл конфигурации `/etc/ssmtp/ssmtp.conf` и отредактируйте его так:

```
mailhub=smtp_сервер:порт
AuthUser=имя_пользователя
AuthPass=пароль
UseSTARTTLS=Yes
```

Эти параметры подойдут для большинства современных SMTP-серверов. Осталось только раздобыть сами параметры доступа. Здесь несколько вариантов:

- ☐ обратиться к вашему провайдеру VDS и запросить параметры SMTP — самый простой вариант;
- ☐ использовать условно-бесплатные сервисы вроде **mailjet.com**, где предоставляется 6000 сообщений в месяц бесплатно (но не более 200 сообщений в день)
- ☐ использовать бесплатные сервисы отправки почты вроде «Яндекс».

## 31.2. Класс *PHPMailer*. Разработка сценария автоматической рассылки прайс-листа

В PHP для отправки писем служит, как уже отмечалось, функция `mail()`, но использовать ее можно разве что для отправки самых простых сообщений. Да, она позволяет задать заголовки письма и даже отправлять письма в формате HTML (при указании типа письма `text/html`). Но что делать, когда вам нужно отправить вложение — например, прайс-лист?

Чтобы не изобретать велосипед, проще использовать класс `PHPMailer` (<https://github.com/PHPMailer/PHPMailer>). Он содержит все необходимое для отправки электронных писем.

По умолчанию `PHPMailer` тоже использует функцию `mail()`, но его прелесть в том, что этот класс может задействовать любой SMTP-сервер, который вы укажете. Даже если вы воспользуетесь системными настройками, т. е. отправкой почты через `mail()`, класс `PHPMailer` сделает работу с исходящей корреспонденцией очень простой.

Класс `PHPMailer` поддерживает SMTP-аутентификацию и отправку сообщений с вложениями. Перед началом работы с `PHPMailer` нужно отредактировать файл `class.phpmailer.php`. Как минимум следует указать имя SMTP-сервера и его порт:

```
var $Host          = 'smtp.example.com';
var $Port          = 25;
```

В некоторых случаях (когда передача данных осуществляется с шифрованием SSL) вместо порта 25 используется порт 465.

Тип шифрования задается переменной `$SMTPSecure` — для SSL и TLS соответственно нужно установить ее значение так:

```
var $SMTPSecure = "ssl";  
var $SMTPSecure = "tls";
```

Чтобы включить SMTP-аутентификацию, надо установить следующие переменные так:

```
var $SMTPAuth      = true;  
var $Username      = 'имя_пользователя';  
var $Password      = 'пароль';
```

Кодировка писем устанавливается так:

```
public $CharSet      = 'utf-8';
```

Теперь приступим к написанию сценария, использующего `PHPMailer`. Подключаем класс:

```
require_once("class.phpmailer.php");  
$mailer = new PHPMailer;
```

Следующий код устанавливает получателя, добавляет вложение и отправляет сообщение:

```
// очищаем список получателей  
$mailer->ClearAddresses();  
// добавляем получателя, вы можете указать  
// несколько получателей  
$mailer->AddAddress('test@example.com');  
// адрес отправителя  
$mailer->From = 'mail@example.com';  
// имя отправителя  
$mailer->FromName = 'Home';  
// тема сообщения  
$mailer->Subject = 'Привет';  
// текст сообщения  
$mailer->Body = "Пример отправки сообщения с вложением";  
// удаляем все вложения  
$mailer->ClearAttachments();  
// добавляем вложение. Первый параметр -  
// имя файла на диске, второй - имя файла во вложении  
$mailer->AddAttachment('mailer/test.zip', 'test.zip');  
// отправляем письмо  
if ($mailer->Send()) {  
    echo "<p>Сообщение успешно отправлено";  
} else {  
    // выводим сообщение об ошибке  
    echo $mailer->ErrorInfo;  
}
```

Рассмотрим небольшой практический проект. Представим, что отделу сбыта нужно рассылать прайс-лист клиентам вашей фирмы. Нам нужно организовать автоматическую рассылку прайс-листа по указанному списку адресов. Адреса клиентов, как и многое другую информацию, которую можно представить в виде таблицы, отечественные менеджеры почему-то привыкли хранить в формате Excel. Мы не станем создавать систему управления таблицей клиентов на базе MySQL и переучивать менеджеров — пусть себе ведут свой список в Excel. Нужно только экспортировать его в формат CSV, затем прочитать этот файл, выделить из него e-mail и отправить каждому клиенту копию прайс-листа.

Чтобы не усложнять наш сценарий, будем считать, что:

- ❑ архив с прайс-листом размещен в каталоге mail подкаталога, в котором находится сценарий, и называется он price.zip. Вы можете добавить форму загрузки файла и сделать так, чтобы менеджер мог каждый раз загружать прайс по протоколу HTTP. Это не сложно — в своем проекте я так и сделал, но чтобы не усложнять код сейчас, возложу добавление такой возможности на ваши плечи. Текущая версия сценария предполагает, что файл с прайсом попал в каталог mail до запуска сценария, — например, вы его туда загрузили по FTP;
- ❑ текст сообщения хранится в файле message\_body.txt. Как правило, текст сообщения всегда один, его достаточно один раз записать в такой файл и закачать по FTP на сервер;
- ❑ все e-mail-адреса в CSV-файле (который, кстати, называется clients.csv) указаны правильно. При желании можете добавить функцию проверки корректности e-mail (см. главу 6);
- ❑ в CSV-файле всего два поля: первое — имя получателя (или название фирмы), второе — e-mail.

Код нашего сценария представлен в листинге 31.1. Вкратце опишу его работу: сначала сценарий получает из CSV-файла адрес получателя, затем передает его функции `phpmailer()`, которая и осуществляет отправку.

#### Листинг 31.1. Сценарий sender.php

```
<?php

// подключаем PHPMailer
require_once("class.phpmailer.php");
$mailer = new PHPMailer;

$from = 'den@host.ru';
$from_name = 'Home';

// функция отправки сообщения с вложением
function phpmailer($email) {

global $mailer, $from, $from_name;
```

```
// получаем текст сообщения
$txt = join(' ', file('message_body.txt'));

$mailer->ClearAddresses();
$mailer->AddAddress($email);
// адрес отправителя
$mailer->From = $from;
// имя отправителя
$mailer->FromName = $from_name;
// тема сообщения
$mailer->Subject = 'Наш новый прайс-лист';
// текст сообщения
$mailer->Body = $txt;
// удаляем все вложения
$mailer->ClearAttachments();
// добавляем вложение
$mailer->AddAttachment('mailer/price.zip', 'price.zip');
// отправление письма
if ($mailer->Send()) {
    echo "<p>Сообщение по адресу $email успешно отправлено <br>";
} else {
    // выводим сообщение об ошибке
    echo $mailer->ErrorInfo;
}

} // phpmailer()

// открываем CSV-файл
$f = fopen('clients.csv', 'r');

while($array=fgetcsv($f, 1024, ';')) {
    // получаем имя получателя и его адрес
    $name= $array[0];
    $email = $array[1];

    // отправляем сообщение
    phpmailer($email);

}

?>
```

Мы создали простой сценарий автоматической отправки прайс-листа, работающий без использования базы данных, что упрощает эксплуатацию сценария.

## 31.3. Получение писем по протоколу POP3

Для приема электронных сообщений используется протокол POP3 (Post Office Protocol v. 3). Письма поступают на сервер и хранятся там, пока мы их не получим (можно при получении сообщения не удалять, тогда они останутся на сервере и после получения). Если при отправке сообщений еще можно было написать свой универсальный класс, используя сокет, то при приеме сообщения по протоколу POP3 это будет сделать накладно — уж очень много нюансов. Гораздо проще применить готовые решения.

Здесь мы рассмотрим класс POP3 (файл pop3.zip из каталога Glava\_31 сопровождающего книгу электронного архива). Как и PHPMailer, класс POP3 берет на себя всю рутинную работу. Нам остается лишь запрограммировать действия POP3-клиента.

Разберемся, как нужно работать с классом POP3. Первым делом следует подключить сам класс:

```
include "pop3.php";
```

Сразу после этого необходимо создать экземпляр класса:

```
$pop3=new pop3_class;
```

Вот теперь можно подключаться к POP3-серверу:

```
// задаем имя POP-сервера
$pop3->hostname="pop.mail.ru";
// открываем соединение
$error_message = $pop3->Open();
```

Метод `Open()`, как и многие другие методы класса POP3, возвращает строку. Если строка пуста, значит, операция выполнена успешно, в противном случае строка содержит сообщение об ошибке. Поэтому сразу после подключения, точнее, вызова метода `Open()`, мы должны проанализировать переменную `$error_message`. Если она не пустая, то выводим ее и завершаем работу сценария:

```
if ($error_message != "") die($error_message);
```

Если мы успешно подключились к серверу, нам потребуется сообщить ему имя пользователя и пароль, а также указать, будет ли использоваться метод аутентификации APOP (Authenticated Post Office Protocol) или нет (последний параметр метода `Login`):

```
$apop = 0; // APOP не используется
$error_message = $pop3->Login($user,$password,$apop);
```

После успешной аутентификации (об этом будет свидетельствовать «пустая» переменная `$error_message`) нужно получить статистику, а именно сколько сообщений в почтовом ящике и какой размер они занимают:

```
$messages = 0; // число сообщений в ящике
$size = 0; // размер
$pop3->Statistics(&$messages,&$size);
```

Если в почтовом ящике есть сообщения, можно с помощью метода `ListMessages()` получить список сообщений или сразу методом `RetrieveMessage()` начать получать сами сообщения. Рассмотрим определение метода `ListMessages()`:

```
Function ListMessages (string message, int id)
```

Первый параметр — это число возвращаемых сообщений. Если задана пустая строка, возвращаются все сообщения (кстати, нумерация сообщений начинается с единицы, а не с нуля). Второй параметр задает, какую информацию о сообщении нужно сообщать: если `id = 1`, то возвращается уникальный идентификатор сообщения, если `id = 0`, то — размер сообщения. Список сообщений возвращается в виде массива. Вот как можно его вывести:

```
$result=$pop3->ListMessages("",0);
for(Reset($result), $m=0; $m<count($result); Next($result), $m++)
echo '<P>Сообщение ' . Key($result). ' '
    . $result[Key($result)] . ' байтов';
```

Метод `RetrieveMessage()` используется для получения сообщений:

```
Function RetrieveMessage($message, &$headers, &$body, $lines)
```

Первый параметр — это номер получаемого сообщения, второй параметр служит для получения заголовков сообщения, третий — для получения тела сообщения. Последний параметр регулирует, сколько строк тела сообщения необходимо получить. Если хотите получить все сообщение, установите отрицательное значение.

Для удаления сообщения с сервера служит метод `DeleteMessage()`:

```
Function DeleteMessage(int $N)
```

где `N` — это номер удаляемого сообщения.

Отключиться от сервера можно с помощью метода `Close()`.

Мы вплотную подошли к созданию собственного сценария получения сообщений, полная версия которого представлена в листинге 31.2.

#### Листинг 31.2. Сценарий получения сообщений

```
<?
include "pop3.php";

$user="test";
$password="12345678";
$apop=0;
$pop3=new pop3_class;

$pop3->hostname="pop.mail.ru";

$error_message = $pop3->Open();
if ($error_message != "") die($error_message);

$error_message = $pop3->Login($user,$password,$apop);
if ($error_message != "") die($error_message);
```

```

$messages = 0; // число сообщений в ящике
$size = 0;      // размер

$pop3->Statistics(&$messages,&$size);

echo "<p>Messages: $messages Size: $size";

if ($messages>0) {

    for($M=1; $M<=$messages; $M++)
    {
        // получаем сообщение с номером $M
        if(($error=$pop3->RetrieveMessage($M,&$headers,&$body,-1)) == "")
        {
            echo "<p><b>Message $M :</b>
                <p><b>Headers</b>";

            // выводим заголовки сообщения
            for($line=0;$line<count($headers);$line++)
                echo "<PRE>", htmlspecialchars($headers[$line]), "</PRE>\n";

            // выводим тело сообщения
            echo "<p><b>Body</b>:<PRE>";
            for($line=0;$line<count($body);$line++)
                echo htmlspecialchars($body[$line]). "\n";
        }
    }

}

$pop3->Close();

?>

```

## 31.4. Получение писем по протоколу IMAP

Для получения писем можно использовать как протокол POP3, так и IMAP. В настоящее время протокол IMAP постепенно вытесняет протокол POP3, поэтому не рассмотреть его мы не можем. Но вы должны знать, что в PHP есть встроенные функции для работы, как с протоколом IMAP, так и с протоколом POP3, — даже не придется использовать какие-либо дополнительные классы, как мы это делали в предыдущем примере.

Сразу перейдем к практике и рассмотрим простейший пример работы с IMAP-сервером (листинг 31.3).

**Листинг 31.3. Подключение к IMAP-серверу и получение последнего письма**

```

<?php
// открываем IMAP-соединение
$mail = imap_open('{mail.example.com:143}', 'username', 'password');

```

```
// можно открыть POP3-соединение
// $mail = imap_open('{mail.example.com:110/pop3}', 'username', 'password');
// берем список всех почтовых заголовков
$headers = imap_headers($mail);
// берем объект заголовка для последнего сообщения в почтовом ящике
$last = imap_num_msg($mail);
$header = imap_header($mail, $last);
// выбираем тело для того же сообщения
$body = imap_body($mail, $last);
// закрываем соединение
imap_close($mail);
?>
```

Первая строка обеспечивает подключение к серверу mail.example.com, порт 143 (IMAP) с заданными именем пользователя и паролем. Аналогично можно открыть POP3-соединение — для этого нужно указать порт 110 и добавить /pop3 к строке.

Чтобы зашифровать ваше соединение с помощью SSL, добавьте /ssl в конце точно так же, как вы поступали с pop3. Необходимо также убедиться в том, что ваша инсталляция PHP собрана с конфигурационным параметром --with-imap-ssl в дополнение к параметру --withimap.

Кроме того, саму системную библиотеку IMAP необходимо собрать с поддержкой SSL. Если вы используете сертификат, подписанный самостоятельно, и хотите предотвратить неудачные попытки проверки его подлинности, добавьте также /novalidate-cert. Пример:

```
$mail = imap_open('{mail.server.com:993/novalidate-cert/pop3/ssl}',
    'username', 'password');
```

После установки соединения с сервером нужно получить заголовки сообщений, находящихся в почтовом ящике. Сделать это можно с помощью функции `imap_headers()`:

```
$headers = imap_headers($mail);
```

Функция `imap_header()` возвращает объект с несколькими полями. Самые полезные из них поля: `subject`, `fromaddress` и `udate`. Функция `imap_body()` позволяет получить тело письма с заданным ID. Мы сначала получаем ID последнего письма функцией `imap_num_msg($mail)`, а потом получаем его тело.

Элемент `body` — это просто строка, но если сообщение состоит из нескольких частей — например, сообщение представлено и в виде простого текста, и в виде HTML-документа, то элемент `$body` содержит обе версии MIME-строки, описывающие их:

```
-----_Part_1046_3914492.1008372096119
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Plain-Text Message
-----_Part_1046_3914492.1008372096119
```

```
Content-Type: text/html
Content-Transfer-Encoding: 7bit
<html>HTML Message</html>
-----_Part_1046_3914492.1008372096119--
```

Чтобы избежать этого, вызывайте функцию `imap_fetchstructure()` в комбинации с функцией `imap_fetchbody()`. Это позволит выяснить, как отформатировано тело, и извлечь только требуемую часть:

```
// выбираем текст для сообщения $n
$st = imap_fetchstructure($mail, $n);
if (!empty($st->parts)) {
    for ($i = 0, $j = count($st->parts); $i < $j; $i++) {
        $part = $st->parts[$i];
        if ($part->subtype == 'PLAIN') {
            $body = imap_fetchbody($mail, $n, $i+1);
        }
    }
} else {
    $body = imap_body($mail, $n);
}
```

Если сообщение состоит из нескольких частей, то переменная `$st->parts` содержит массив объектов, которые их описывают. Свойство `part` содержит целочисленное описание MIME-типа основного тела.

#### **ПРИМЕЧАНИЕ**

Напомню, что коды листингов 31.1–31.3 приведены в файлах *31-1.php–31-3.php* соответственно каталога *Glava\_31* сопровождающего книгу электронного архива (см. приложение 4).

Надеюсь, что материал этой главы поможет вам в разработке собственного почтового интерфейса.



## ГЛАВА 32

# Введение в PEAR

### 32.1. Серьезные проекты и PEAR

PEAR (PHP Extension and Application Repositor) — это база расширений и приложений для PHP. Понять, что представляет собой PEAR, лучше на примере. Предположим, что у вас есть довольно серьезный проект, состоящий, скажем, из 100 PHP-сценариев (на самом деле это не очень много). Ваш проект основан на сервере баз данных MySQL, что неудивительно — MySQL является самым популярным сервером баз данных и часто используется в паре с PHP.

Вот пример фрагмента кода, который будет, вероятнее всего, присутствовать в каждом из ваших 100 сценариев:

```
include "config.php";

$mysqli = new mysqli($dbhost,$dbuser,$dbpass,$db);
if ($mysqli->connect_errno) {
    die($mysqli->connect_error);
}

// запрос
$q = "select * from main_table";
$r = $mysqli->query($q);

while ($row = $r->fetch_array(MYSQLI_ASSOC)) {
    {
        echo $row[field_1];
    }
}
```

Здесь мы видим: установку соединения, выбор базы данных, передачу запроса и обработку результата. Все вроде бы хорошо. До одного прекрасного момента — пока вы (или ваше руководство) не надумаете перейти на другой сервер баз данных — например, на Oracle. Тогда вам придется изменить все 100 сценариев. При чем простая замена имен функций с `mysql_` на `ora_` не пройдет.

Что же делать? Первое, что приходит в голову, — создать класс вида:

```
<?php
class DBClass
{
    public $connection;

    function DBClass()
    {
    }

    function connect($dbhost, $dbname, $login, $password)
    {
        $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $db);
        if ($mysqli->connect_errno) {
            die($mysqli->connect_error);
        }
        $this->connection = $mysqli;
        return $mysqli;
    }

    function query($sql)
    {
        return $this->connection->query($sql);
    }
    ...
}
```

Этот класс нужно подключить ко всем сценариям и использовать только его методы класса при обращении к БД. Тогда для перехода на другой сервер баз данных вам нужно будет изменить только этот файл. Что намного проще, чем редактировать все ваши 100 сценариев. К тому же ваш проект станет намного более универсальным, ведь можно написать несколько разных классов — под разные СУБД, тогда перенос вашего проекта на другой сервер, где используется другая СУБД, не составит никакого труда.

Если вы действительно надумали создавать подобный класс, то могу вас заверить, что вы изобретаете велосипед. Поскольку все уже сделано за вас — вам следует лишь выбрать нужный вам класс из базы PEAR.

PEAR — это больше, чем просто набор классов. PEAR — это библиотека открытого кода для PHP-пользователей, система управления пакетами и распространения этих пакетов среди разработчиков, базовые классы PHP-кода, библиотека дополнительных модулей для PHP (The PHP Extension Code Library, PECL) и многое другое. Настоятельно рекомендую ознакомиться с официальным сайтом PEAR: <https://pear.php.net/manual/ru/introduction.php>.

## 32.2. Пример использования класса *DB*

Прежде всего нужно определиться, какие PEAR-пакеты вам нужны. Для этого посетите следующую страницу: <https://pear.php.net/packages.php>.

На момент написания этих строк в базе PEAR было 603 пакета — этого хватит для большинства проектов. Не спешите качать все подряд. Нам понадобится только пакет PEAR, содержащий классы PEAR и PEAR\_Error, на которые и опирается большинство (однако не все) пакетов из базы PEAR.

Последняя версия пакета PEAR доступна по адресу:

<https://pear.php.net/package/PEAR/download>.

Распакуйте архив PEAR в какой-нибудь каталог, найдите в нем файл PEAR.php. Затем перейдите в каталог веб-сервера, где хранится ваш проект, и создайте в нем каталог pear. В этот каталог нужно скопировать найденный вами файл PEAR.php. Остальные файлы на этом этапе нам не нужны — для работы класса DB вполне хватит одного файла PEAR.php — нечего захламлять сервер ненужными файлами. Необходимо отметить, что это неполная установка PEAR, с полной установкой вы можете ознакомиться по адресу: <https://pear.php.net/manual/en/installation.php>.

Последняя версия пакета DB доступна по адресу: <https://pear.php.net/package/DB/download>. Распакуйте полученный архив. Мы опять не будем копировать на веб-сервер все его файлы, а скопируем только те, которые нам необходимы: подкаталог DB (он появится, когда вы распакуете архив) и файл DB.php.

Далее откройте файл DB.php и в самом начале найдите строчку:

```
require_once 'PEAR.php'
```

Удалите или закомментируйте ее. Эта строка подключала файл PEAR.php, но поскольку мы устанавливали PEAR выборочно, то будем подключать его вручную в основном файле.

Теперь создайте файл test\_pear.php и поместите его в корневой каталог сервера. У вас должна получиться следующая структура файлов и каталогов:

```
/-
|_ test_pear.php
|_ pear
|   |_ PEAR.php
|   |_ DB.php
|   |_ DB
|       |_ common.php
|       |_ dbase.php
|       ...
```

Отредактируем файл test\_pear.php. Текст этого файла с моими комментариями представлен в листинге 32.1.

**Листинг 32.1. Файл test\_pear.php**

```
<?php
require_once('pear/PEAR.php');
require_once('pear/DB.php');
// мы будем использовать MySQL
// oci8 - Oracle
// msql - MS SQL
// sqlite - SQL Lite
// sybase - Sybase
// список других доступных "драйверов" можно найти в каталоге pear/DB
$myDB =& DB::factory('mysql');

// подключаемся к базе данных
$myDB -> connect (DB::parseDSN('mysql://логин:пароль@сервер/база_данных'));

// произвольный запрос
$q = 'SELECT * FROM main_table';

// получаем результат
$result = $myDB -> query($q);

// проверяем, получен ли результат
if(!DB::isError($result) && $result -> numRows() > 0) {
    while($row = $result -> fetchRow(DB_MOD_ASSOC)) {
        // результат получен, выводим его
        print('field_1 = '.$row['field_1']);
    }
}
?>
```

**ПРИМЕЧАНИЕ**

Чтобы не допустить ошибок при наборе кода, настоятельно рекомендую использовать уже готовый вариант (файл 32-2.php), представленный в каталоге *Glava\_32* сопровождающего книгу электронного архива (см. приложение 4).

Рассмотрим листинг 32.1 подробнее. Первым делом мы подключили необходимые нам файлы:

```
require_once('pear/PEAR.php');
require_once('pear/DB.php');
```

Далее мы выбираем тип СУБД:

```
$myDB =& DB::factory('mysql');
```

После этого устанавливаем соединение с базой данных:

```
$myDB -> connect(DB::parseDSN('mysql://логин:пароль@сервер/база_данных'));
```

Затем выполняем произвольный запрос:

```
$q = 'SELECT * FROM main_table';  
$result = $myDB -> query($q);
```

Все остальное — это обработка результата. С другими методами класса `DB` можно ознакомиться в официальной документации: <https://pear.php.net/manual/en/package.database.db.php>. Документация (на русском языке) по другим классам PEAR доступна по адресу: <https://pear.php.net/manual/ru/>.

В *главе 47* мы рассмотрим еще один PEAR-пакет — пакет, использующийся для подключения к базе данных MongoDB.



## ГЛАВА 33

# Импорт и экспорт данных

### 33.1. Импорт прайс-листов из формата CSV в базу данных MySQL

Представим, что у нас есть прайс-лист запчастей в формате CSV. Формат его строк следующий:

```
номер_склада;артикул;наименование;к-во;цена
```

Пример такого файла:

```
1;17 11 7 639 021;Radiator cap;5;14.95
1;17 12 7 593 490;Coolant hose;2;48.11
2;17 12 7 537 101;Coolant hose;3;29.15
2;83 19 2 211 191; Anti-freeze;10;9.0
```

Подразумевается, что пользователь будет загружать файл через браузер, а также указывать множитель, на который нужно умножить цену перед импортом позиции в базу данных.

Первым делом в сценарии мы получаем этот самый множитель, устанавливаем переменные, относящиеся к загрузке файла, и загружаем файл:

```
$mul = $_POST['mul']; // Осторожно! Не фильтруем значение
$uploaddir = 'csv/'; // Каталог, в который будет загружен файл
$uploadfile = $uploaddir."hold3.csv"; // Имя файла в каталоге
```

Загрузка файла осуществляется так:

```
if (copy($_FILES['csv']['tmp_name'], $uploadfile))
{
    // Операции, выполняемые в случае успешной загрузки
}
else
    die('Error due file uploading');
```

Имя поля в форме — csv:

```
<form method=post action=parser_03.php ENCTYPE=multipart/form-data>
...
<p> CSV file <input name=csv type=file>
...
```

Самое интересное, как вы догадались, происходит после комментария: Операции, выполняемые в случае успешной загрузки.

Теперь обсудим алгоритм импорта. Чтение файла будет выполняться построчно функцией `fgetcsv()`. Поскольку размеры прайсов бывают довольно большими, не нужно загружать в память весь файл — при работе с одной строкой память расходуется экономнее. Что же касается производительности, то скажу так: этот сценарий писался для клиента с не очень хорошим хостингом, где стояло ограничение (`memory_limit`) 64 Мбайт, и даже загрузка прайса размером 10 Мбайт вызывала превышение лимита памяти. А производительности этого сценария было достаточно, чтобы за 30 секунд (таково ограничение на время выполнения сценария) построчно прочитать и вывести в браузер более 170 тыс. строк. Думаю, для большинства прайсов этого более чем достаточно.

Итак, с чтением файла мы разобрались. Импортировать прочитанные данные мы будем в таблицу `sklad`, структура которой напоминает структуру прайса, но все же немного отличается, — в ней содержатся некоторые служебные поля, которые использовались ранее (напоминаю, мы рассматриваем реальный проект, а не какой-то абстрактный, где все красиво, и структура прайса повторит структуру таблицы).

В эту таблицу загружаются данные из нескольких источников: из бухгалтерской программы (в нашем случае — «1С») и из разных прайсов — менеджерами. Если бы в таблицу `sklad` производилась загрузка из одного источника, мы могли бы перед началом импорта просто очистить ее:

```
TRUNCATE TABLE sklad;
```

Можно было бы использовать и SQL-оператор `DELETE`, но `TRUNCATE`, как правило, работает быстрее. Если именно ваша таблица использует поля-счетчики, то хорошо было бы сбросить `AUTO_INCREMENT`, установив его в 1:

```
ALTER TABLE sklad AUTO_INCREMENT = 1
```

После этого нужно просто выполнить импорт — серию операций `INSERT`. При этом вам не нужно заботиться, если ли уже такой артикул в таблице или нет. Но в моем случае все было гораздо сложнее — ведь данные импортировались из разных источников. Если удалить все записи, а потом сделать импорт из одного источника, то он не «перекроет» записи, которые были получены из других источников. А это неправильно. Поэтому при импорте мне пришлось проверять существование конкретной записи. Для этого была написана следующая функция (листинг 33.1).

**Листинг 33.1. Проверяем, изменилась ли запись**

```
// Возвращаемые значения
// 0 - элемент не существует
// 1 - элемент существует и не требует изменения
// 2 - элемент требует обновления
function process_item($p, $hold, $desc, $qty, $price) {
    $res = 0;
    global $table;
    global $mysqli;

    $q = "select * from $table where part_number = \"\$p\"
        and hold = $hold limit 1";
    $r = $mysqli->query($q);
    if ($r->num_rows > 0) {
        // Элемент существует, ответ будет или 1, или 2
        $row = $mysqli->fetch_array();
        // Проверяем, изменилось ли описание, к-во и цена
        if (($row['element_name'] == $desc) && ($row['xQty'] == $qty) &&
            ($row['rozn'] == $price))
            $res = 1;
        else
            $res = 2;
    }
    return $res;
}
```

Здесь мы выполняем запрос к таблице `sklad`. Имя таблицы передается через глобальную переменную `$table`, чтобы ее можно было легко изменить. Также мы используем еще одну глобальную переменную: `$mysqli` — для хранения соединения с базой данных.

**ПРИМЕЧАНИЕ**

Все листинги из этой главы вы найдете в каталоге `Glava_33` сопровождающего книгу электронного архива (см. *приложение 4*).

Определение, существует ли запись, происходит по двум критериям: должны совпадать номер склада и артикул. Ведь в наличии могут быть запчасти, находящиеся на разных складах, например:

```
1;17 11 7 639 021;Radiator cap;5;14.95
2;17 11 7 639 021;Radiator cap;1;12.95
```

Эти две записи считаются разными, поскольку вторая относится к складу 2. Давайте проясним: первая запись означает, что на складе 1 есть 5 крышек радиатора с артикулом 17 11 7 639 021, цена каждой — 14.95 условных единиц (долларов или евро — не важно, мы считаем, что все цены указаны в одной валюте). На втором складе есть одна такая крышка и ее цена 12.95 у.е.

Именно поэтому производится выборка по артикулу (17 11 7 639 021 — в нашем случае) и номеру склада. Если такая запись найдена, то мы должны решить, изменилась ли она или нет по сравнению с теми данными, которые есть в CSV-файле. Если запись не изменилась, мы возвращаем значение 1. Получив это значение, наш основной сценарий ничего делать не будет — он просто перейдет к чтению следующей строки из CSV-файла.

Если же запись изменилась (например, изменилась цена, или описание товара, или количество), то мы возвращаем значение 2. Получив это значение, наш основной сценарий выполнит запрос UPDATE.

Если запись не существует, функция вернет значение 0, а наш сценарий выполнит вставку записи — запрос INSERT.

Рассмотрим основной сценарий — код, который выполняется в случае успешной загрузки файла (листинг 33.2).

### Листинг 33.2. Импорт из CSV

```
// Счетчики для статистики
$row = 0; // обработано (прочитано) строк
$skipped = 0; // пропущено строк
$affected = 0; // затронуто (обнов. или вставка) строк

// Построчное чтение
for ($row = 0; $dsa = fgetcsv($f, 1000, ";"); $row++) {
    $hold = $dsa[0]; // номер склада
    $part_num = $dsa[1]; // артикул
    if ($part_num == "") continue; // пропуск пустых строк

    $desc = $dsa[2]; // описание (наименование)
    // экранируем символы для помещения строки в SQL-запрос
    $desc = mysqli_real_escape_string($desc);
    $qty = $dsa[3]; // количество

    // Защита от дурака: заменяем запятую на точку во множителе
    // множитель - передается пользователем
    $mul = str_replace(',', '.', $mul);

    $price = $dsa[4]; // цена
    // заменяем запятую точкой (double) - вдруг будет ошибка в CSV
    $price = str_replace(',', '.', $price);
    $price = $price * $mul; // умножение
    $price = round($price, 2); // округление до 2 знаков

    // Что делать с записью - вставить, обновить или пропустить?
    $res = process_item($part_num, $hold, $desc, $qty, $price);
```

```

if ( $res == 0)
    // SQL-код вставки, у вас будет другим
    $q = "insert into $table values (\\"$part_num\\",
        \\"$desc\\", $hold, \\"$price\\", 0, 10, 0, $qty, 0,
        \\"\\", \\"$price\\")";
else
    if ($res == 1) {    // Запись не изменилась - пропуск
        $skipped++;
        continue;
    }

    else
        if ($res == 2)
            $q = "update $table set element_name = \\"$desc\\",
                xQty = $qty, rozn = \\"$price\\" where
                part_number = \\"$part_num\\" and hold = $hold limit 1";

    $r = mysqli->query($q);
    $error = mysqli->error();
    if ($error != "")
        echo "<B>ERROR:</B> $error";

}

$affected = $row - $skipped;
// Перенаправляем на сценарий, показывающий форму импорта прайса
// и сообщаем собранную статистику для контроля
Header("Location:
http://site.com/import/prices.php?h=$hold&r=$row&s=$skipped&a=$affected");

```

## 33.2. Преобразование файлов Excel в CSV с помощью PHP. Импорт прайсов из Excel

Импортируемые данные могут приходить в разных форматах: CSV, XML и, конечно же, Excel. Вместо того, чтобы загружать менеджеров лишней работой — а именно сведением всех форматов к одному (CSV), лучше автоматизируем этот процесс. Например, если к нам поступил прайс в формате Excel, то его можно преобразовать в формат CSV с помощью библиотеки `PHPExcel` (<https://github.com/PHPOffice/PHPExcel>).

В листинге 33.3 показано, как преобразовать загруженный с помощью браузера Excel-файл в формат CSV. Изначально прайс-лист был в формате Excel 2007 (\*.xlsx), но вы можете указать другой формат. Допустимые значения приводятся в массиве `$excel_readers`. Это сугубо информационный массив — чтобы не забыть названия форматов.

**Листинг 33.3. Загрузка Excel-файла и его преобразование в формат CSV**

```
if (copy($_FILES['xls']['tmp_name'], $uploadfile))
{
    // Конвертирование прайса в формат CSV для последующей обработки
    $excel_readers = array(
        'Excel5' ,
        'Excel2003XML' ,
        'Excel2007'
    );
    // Подключаем библиотеку
    require_once('lib/PHPExcel.php');

    // Устанавливаем формат Excel 2007
    $reader = PHPExcel_IOFactory::createReader('Excel2007');
    $reader->setReadDataOnly(true);

    // Загружаем файл
    $path = $uploadfile;
    $excel = $reader->load($path);

    // Выполняем преобразование в формат CSV
    $writer = PHPExcel_IOFactory::createWriter($excel, 'CSV');
    // Сохраняем результат в файл tech.csv
    $writer->save('tech.csv');
}
```

По окончании выполнения этого кода у вас будет файл tech.csv, который можно обработать, как было показано в предыдущем примере.

## 33.3. Работа с XML-файлами

### 33.3.1. Парсинг XML-файла

Довольно часто перед программистами ставятся задачи импорта/экспорта. Например, одна из программ выгружает информацию о товарах в XML-формате, а нам эту информацию нужно прочитать, выполнить ее *парсинг* (т. е. разбор) и поместить в базу данных. Представим, что у нас есть XML-файл, приведенный в листинге 33.4.

**Листинг 33.4. XML-файл**

```
<?xml version="1.0" encoding="UTF-8"?>
<items>
    <item>
        <sku>3-025 289</sku>
        <priceUSD>17</priceUSD>
```

```

        <priceUAH>880</priceUAH>
        <address>Shop 1</address>
        <qty>1</qty>
    </item>
    <item>
        <sku>1-141 836</sku>
        <priceUSD>247</priceUSD>
        <priceUAH>15560</priceUAH>
        <address>Shop 2</address>
        <qty>2</qty>
    </item>
</items>

```

В разделе `items` есть несколько подразделов `item`, каждый из которых несет информацию о товаре, а именно:

- ❑ артикул — `sku`;
- ❑ цену в долларах — `priceUSD`;
- ❑ цену в местной валюте — `priceUAH`;
- ❑ адрес магазина, в котором находится товар в текущий момент, — `Shop N`;
- ❑ количество товара — `qty`.

Нам нужно прочитать этот файл и выполнить его разбор. Изобретать колесо не будем, а вместо этого воспользуемся стандартным классом `SimpleXMLElement()`. Этот класс позволяет создать массив объектов, прочитанных из XML-файла. Затем в цикле мы можем обратиться к каждому из объектов, чтобы прочитать его свойства. Пример парсинга XML-файла приведен в листинге 33.5.

#### Листинг 33.5. Парсинг XML-файла

```

<?php
// Читаем содержимое XML-файла в одну строку
$xmlstr = file_get_contents('todayImport.xml');
// Строим массив элементов
$items = new SimpleXMLElement($xmlstr);

// Счетчик обработанных элементов
$item_count = 0;

foreach ($items as $item) {
    $priceUSD = str_replace(",", ".", $item->priceUSD);
    $priceUAH = str_replace(",", ".", $item->priceUAH);
    $adr = $item->address;
    $qty = $item->qty;
    $sku = $item->sku;
}

```

```
echo "sku $priceUSD $priceUAH $qty $adr\n";  
$item_count++;  
}  
  
echo "\nTotal items: $item_count";  
  
?>
```

Здесь в цикле `foreach` мы обращаемся к свойствам объекта так: название свойства соответствует названию подраздела в XML-файле. Параллельно ведем счетчик обработанных элементов. Количество элементов массива можно подсчитать и с помощью функции `count()`, однако вы можете в качестве счетчика обработанных элементов использовать переменную `$item_count` — ведь, возможно, не все элементы нужно внести в БД. В нашем случае количество элементов массива равно количеству обработанных, поскольку вся обработка заключается в замене запятой на точку и выводе свойств объекта.

Дополнительные примеры использования класса `SimpleXMLElement` вы найдете в официальной документации: <https://www.php.net/manual/en/simplexml.examples-basic.php>.

### 33.3.2. Генерирование XML-файла

Теперь попробуем выполнить обратную задачу, а именно сгенерировать файл, подобный приведенному в листинге 33.4, с такой же структурой. Первое, что приходит в голову, — сформировать строку и вывести ее в файл функцией `file_put_contents()`. Но это довольно кривое и неправильное решение.

Во-первых, есть стандартные средства для работы с XML-файлом. Во-вторых, лично я довольно ленивый человек, и мне не хочется разбираться с формированием строки и всем прочим. Можно, конечно, записать все в одну строку и вывести ее в файл, но выглядеть это будет не очень хорошо.

Именно поэтому гораздо проще и правильнее использовать средства для работы с XML-файлами. На этот раз мы воспользуемся классом `DomDocument`<sup>1</sup> (листинг 33.6). В конструктор класса мы передаем два параметра: версию документа и кодировку. Затем создаем корневой элемент `$items`. В него можно в цикле добавить множество элементов `$item`, но мы создадим только один такой элемент для простоты примера.

Посмотрите: создав `$items`, мы создаем `$item` путем присоединения дочернего элемента (метод `appendChild()`) к элементу `$items`. Далее аналогичным образом (с помощью `appendChild()`) мы присоединяем элементы `sku`, `priceUSD` и т. д. Внутри каждого такого элемента мы создаем с помощью метода `createTextNode()` текстовое содержимое. То есть мы используем здесь три метода:

---

<sup>1</sup> <https://www.php.net/manual/ru/class.domdocument.php>.

- `appendChild()` — добавляет дочерний элемент;
- `createElement()` — создает элемент вроде `<sku></sku>`;
- `createTextNode()` — создает текстовый узел: `<sku>3-263 743</sku>`.

**Листинг 33.6. Создание XML-файла с помощью DomDocument**

```
<?php

$xml = new DomDocument('1.0', 'utf-8');

$items = $xml->appendChild($xml->createElement('items'));

$item = $items->appendChild($xml->createElement('item'));

$sku = $item->appendChild($xml->createElement('sku'));
$sku->appendChild($xml->createTextNode('3-263 743'));

$priceUSD = $item->appendChild($xml->createElement('priceUSD'));
$priceUSD->appendChild($xml->createTextNode('10'));

$priceUAH = $item->appendChild($xml->createElement('priceUAH'));
$priceUAH->appendChild($xml->createTextNode('280'));

$adr = $item->appendChild($xml->createElement('address'));
$adr->appendChild($xml->createTextNode('280'));

$qty = $item->appendChild($xml->createElement('qty'));
$qty->appendChild($xml->createTextNode('1'));

$xml->formatOutput = true;
$xml->save('goods.xml');

?>
```

Далее мы устанавливаем свойство `formatOutput` в `true`, чтобы `DomDocument` отформатировал за нас XML-код. Наконец, с помощью метода `save()` мы сохраняем XML-код в файл `goods.xml`.

Привожу ссылку на официальную документацию по классу `DomDocument`: <https://www.php.net/manual/en/class.domdocument.php>.

Отмечу только, что с помощью этого класса можно не только создавать, но и читать XML-файлы, а дополнительную информацию вы найдете в официальной документации.



## ГЛАВА 34

# Работаем с MP3

### 34.1. Формат MP3

В настоящее время в Интернете существует очень много сайтов, предлагающих скачать музыку в формате MP3. Для каждой композиции выводится соответствующая информация: название, исполнитель, альбом, год, жанр и др. Как вы думаете, где хранятся эти сведения? Скорее всего, ваш ответ будет: «В базе данных». Вполне вероятно, что это именно так. Но мы рассмотрим иной подход к созданию подобного музыкального сайта.

Конечно, можно хранить всю информацию в базе данных, но ведь формат MP3 поддерживает так называемые ID3-теги, в которых и содержится нужная нам информация! Зачем же дублировать информацию о композиции в базе данных, ведь она уже есть в MP3-файле? Достаточно достать ее оттуда.

Что же такое ID3-тег? Он представляет собой структуру размером 128 байтов, которая дописывается в конец MP3-файла. Структура ID3-тега версии 1.0 следующая:

- ☐ служебное слово TAG;
- ☐ название композиции (30 символов);
- ☐ исполнитель (30 символов);
- ☐ альбом (30 символов);
- ☐ год записи (четырёхзначное число);
- ☐ комментарий (30 символов);
- ☐ жанр (1 байт).

Но ID3-теги версии 1.0 встречаются редко. Значительно чаще используется версия 1.1, которая отличается от 1.0 тем, что в ней сокращена длина комментария — до 28 символов, но зато появилось новое поле, содержащее номер дорожки, т. е. номер композиции в альбоме. На рис. 34.1 показан фрагмент MP3-файла (в шестнадцатеричном виде): видно служебное слово **TAG**, после которого следуют название композиции, имя исполнителя, название альбома, а остальная информация для этого MP3-файла не указана.

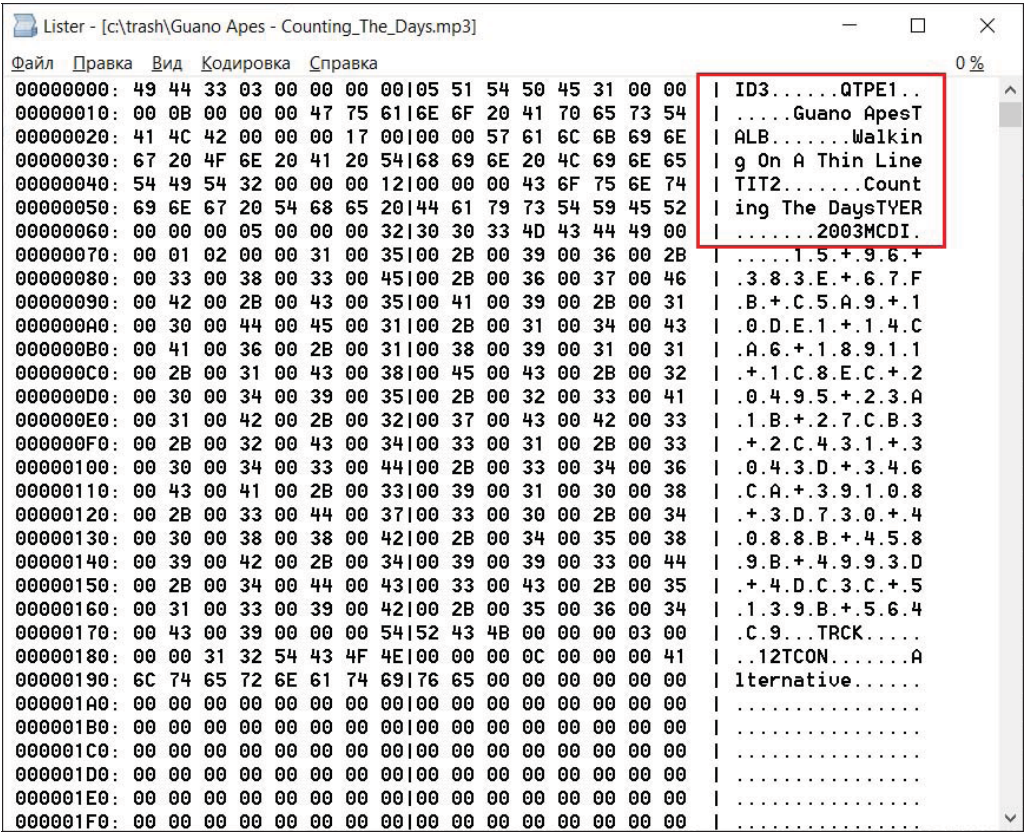


Рис. 34.1. MP3-файл (шестнадцатеричный вид)

## 34.2. Библиотека PEAR

Если честно, то копаться в MP3-файлах в поисках нужной информации как-то не хочется. Нет, с технической стороны это совсем не сложно, дело в отсутствии желания. Причем до такой степени, что можно уговорить себя на создание отдельной таблицы, которая будет содержать всю необходимую нам информацию. К счастью, на помощь приходит библиотека PEAR (подробно она была рассмотрена в главе 32). Ее возможности достаточно велики, но мы будем ее использовать (в паре с пакетом MP3\_Id) для чтения ID3-тегов.

В каталоге Glava\_34 сопровождающего книгу электронного архива (см. приложение 4) вы найдете:

- ☐ файл PEAR-1.10.13.tgz с версией 1.10.13 пакета PEAR<sup>1</sup>. Последнюю версию пакета можно скачать по адресу: <https://pear.php.net/package/PEAR/download>;

<sup>1</sup> Это тот же файл, что и в каталоге Glava\_32, однако удобно, когда все нужное лежит вместе, не так ли?

❑ файл MP3\_Id-1.2.2.tgz с пакетом MP3\_Id. Последняя версия пакета доступна по адресу: [https://pear.php.net/package/MP3\\_Id/download/](https://pear.php.net/package/MP3_Id/download/).

Из первого архива нам понадобится файл PEAR.php, из второго — Id.php. Распакуйте эти файлы в отдельный каталог.

### 34.3. Вывод ID3-тегов

Для вывода информации из ID3-тега потребуется сначала подключить библиотеку MP3\_Id:

```
require_once 'Id.php';
```

Обращение к ID3-тегам осуществляется через объект класса MP3\_Id:

```
$id3 = &new MP3_Id();
```

После того как объект создан, следует прочитать MP3-файл методом read():

```
$result = $id3->read(<имя_файла>);
```

Затем методом getTag() можно прочитать информацию о теге. Методу нужно передать название тега (табл. 34.1).

Таблица 34.1. ID3-теги

Название тега	Описание
<name>	Название композиции
<artists>	Исполнитель
<album>	Альбом
<year>	Год записи
<comment>	Комментарий
<genre>	Жанр
<genreno>	Номер жанра (число)
<track>	Номер композиции в альбоме

Теперь напишем небольшой сценарий, выводящий информацию об MP3-файле (листинг 34.1).

Листинг 34.1. Вывод информации об MP3-файле

```
<?php
// подключаем MP3_Id
require_once 'Id.php';

// создаем объект
$id3 = &new MP3_Id();
```

```
// читаем MP3-файл 1.mp3
$result = $id3->read('1.mp3');

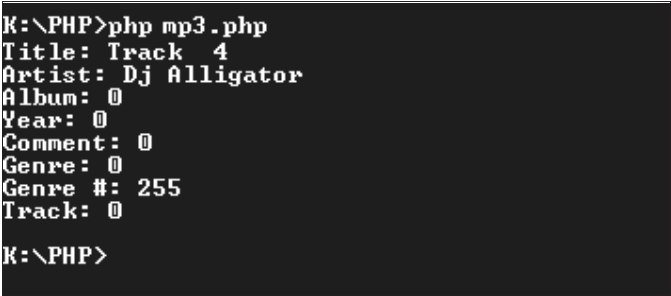
// есть ли ошибка?
if (PEAR::isError($result)) {
    die($result->getMessage() . "\n");
}

// читаем поля и выводим информацию
// название
echo 'Title: ' . $id3->getTag('name') . "\n";
// исполнитель
echo 'Artist: ' . $id3->getTag('artists') . "\n";
// альбом
echo 'Album: ' . $id3->getTag('album') . "\n";
// год
echo 'Year: ' . $id3->getTag('year') . "\n";
// комментарий
echo 'Comment: ' . $id3->getTag('comment') . "\n";
// жанр
echo 'Genre: ' . $id3->getTag('genre') . "\n";
// жанр (номер)
echo 'Genre #: ' . $id3->getTag('genreid') . "\n";
// номер дорожки
echo 'Track: ' . $id3->getTag('track') . "\n";
?>
```

### ПРИМЕЧАНИЕ

Очищенный от комментариев код этого сценария вы найдете в файле *mp3.php* каталога *Глава\_34* сопровождающего книгу электронного архива (см. приложение 4).

Результат работы сценария показан на рис. 34.2.



```
K:\PHP>php mp3.php
Title: Track 4
Artist: Dj Alligator
Album: 0
Year: 0
Comment: 0
Genre: 0
Genre #: 255
Track: 0
K:\PHP>
```

Рис. 34.2. Результат работы сценария (листинг 34.1)

## 34.4. Редактирование ID3-тегов

Библиотека `MP3_id` позволяет не только просматривать ID3-теги, но и редактировать их. Для этого предусмотрен метод `setTag()`, которому требуется передать два параметра (название и значение тега):

```
function setTag($name, $value)
```

Запись изменений в MP3-файл реализует метод `write()`:

```
$id3->write();
```

Рассмотрим небольшой пример установки всех тегов (листинг 34.2).

### Листинг 34.2. Пример установки тегов

```
<?php
require_once 'Id.php';
$id3 = &new MP3_Id();
$id3->read('1.mp3');

$id3->setTag('name', 'Test track');
$id3->setTag('artists', 'Test artists');
$id3->setTag('album', 'Test album');
$id3->setTag('year', 2023);
$id3->setTag('comment', '');
$id3->setTag('genre', 'Rock');
$id3->setTag('track', 8);

// пытаемся записать теги в MP3-файл
$result = $id3->write();

if (PEAR::isError($result)) {
    // произошла ошибка
    die($result->getMessage() . "\n");
}

echo "Теги записаны!";
```

## 34.5. Удаление тега

Для удаления ID3-тега служит метод `remove()` (листинг 34.3). Удаляется весь ID3-тег (все его поля).

### Листинг 34.3. Удаление ID3-тега

```
<?php
require_once 'Id.php';
```

```
$id3 = &new MP3_Id();  
$id3->read('1.mp3');  
$id3->remove();  
if (PEAR::isError($result)) {  
die($result->getMessage() . "\n");  
}
```

```
echo "Тег стерт!";
```

Удаление тега, как и его запись, требуют изменения MP3-файла, поэтому убедитесь, что вы обладаете должными правами для выполнения этой операции.



## ГЛАВА 35

# Расширение cURL: практические примеры

### 35.1. Этот загадочный cURL

Предмет этой главы — мощный инструмент cURL, реализованный в виде библиотеки (расширения) для PHP.

Очень часто в веб-программировании возникают различные задачи взаимодействия по протоколам HTTP и FTP со сторонними серверами. Как раз для решения таких задач, чтобы не изобретать велосипед заново, и предназначен инструмент cURL, а библиотека `libcurl` позволяет получать и отправлять заголовки ответов сервера, программно авторизоваться на разных сайтах, создавать скрипты для автоматизированной отправки сообщений в социальных сетях, загружать файлы и т. п. Да, с помощью cURL можно сделать многое.

Нужно отметить, что библиотека `libcurl` не всегда устанавливается по умолчанию. Скачать библиотеку можно по адресу: <https://curl.haxx.se/download.html>. Убедитесь, что вы загружаете подходящую для вашей операционной системы модификацию.

После загрузки файла библиотеки нужно поместить его в каталог, заданный директивой `extension_dir` вашего файла конфигурации `php.ini`. В моем случае это каталог `/opt/alt/php52/usr/lib64/php/modules` (рис. 35.1).

После этого само расширение нужно «прописать» в файле конфигурации `php.ini`:

```
extension=php_curl.dll
```

Обратите внимание, что указанное здесь название библиотеки `php_curl.dll` соответствует Windows. В других операционных системах файл библиотеки может называться иначе — например, в Linux он носит название `libcurl.so`. Впрочем нужно отметить, что в Linux в большинстве случаев эта библиотека уже установлена.

После внесения изменений в файл `php.ini` следует перезапустить ваш веб-сервер, чтобы изменения вступили в силу. Делать это не придется, если PHP работает в режиме CGI, а не запускается как модуль сервера.

Далее выполните функцию `phpinfo()`. Если вы все сделали правильно, вы увидите информацию о cURL в выводе этой функции (рис. 35.2).

browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	no value	no value
disable_functions	no value	no value
display_errors	Off	Off
display_startup_errors	Off	Off
doc_root	no value	no value
docref_ext	no value	no value
docref_root	no value	no value
enable_dl	On	On
error_append_string	no value	no value
error_log	no value	no value
error_prepend_string	no value	no value
error_reporting	6143	6143
expose_php	On	On
extension_dir	/opt/alt/php52/usr/lib64/php/modules	/opt/alt/php52/usr/lib64/php/modules
file_uploads	On	On
highlight.bg	#FFFFFF	#FFFFFF
highlight.comment	#FF8000	#FF8000
highlight.default	#0000BB	#0000BB
highlight.html	#000000	#000000
highlight.keyword	#007700	#007700
highlight.string	#DD0000	#DD0000
html_errors	On	On
ignore_repeated_errors	Off	Off
ignore_repeated_source	Off	Off
ignore_user_abort	Off	Off
implicit_flush	Off	Off
include_path	./opt/alt/php52/usr/share/pear:/opt/alt/php52/usr/share/php	./opt/alt/php52/usr/share/pear:/opt/alt/php52/usr/share/php
log_errors	On	On
log_errors_max_len	1024	1024
magic_quotes_gpc	Off	Off
magic_quotes_runtime	Off	Off

Рис. 35.1. Каталог расширений

calendar		
Calendar support	enabled	
ctype		
ctype functions	enabled	
curl		
cURL support	enabled	
cURL Information	libcurl/7.36.0 OpenSSL/1.0.1e zlib/1.2.3 libidn/1.18 libssh2/1.4.3	
date		
date/time support	enabled	
"Olson" Timezone Database Version	2010.9	
Timezone Database	internal	
Default timezone	Europe/Kiev	
	Directive	Local Value
	date.default_latitude	31.7667
	date.default_longitude	35.2333
	date.sunrise_zenith	90.583333
	date.sunset_zenith	90.583333
	date.timezone	Europe/Kiev
dba		
DBA support	enabled	
Supported handlers	cdb cdb_make db4 infile flatfile	

Рис. 35.2. Информация о библиотеке

Если у вас нет собственного сервера, но вы используете виртуальный хостинг, выполните на нем функцию `phpinfo()`. Вполне вероятно, что у вас `cURL` уже установлен. Во многих случаях это так и есть.

Нужно отметить, что `cURL` — это невероятно мощный инструмент, заслуживающий отдельной книги, пусть и не очень большой, поэтому изучение всех его возможностей выходит за рамки нашего издания. Далее мы рассмотрим практическое применение `cURL` — именно те примеры, которые представляют ценность для программиста, те, с которыми он может столкнуться на практике. Что же касается официальной документации, то она всегда доступна по адресу: <https://curl.haxx.se/docs/>, и я не вижу смысла переписывать ее сюда.

## 35.2. Авторизация на сайте и загрузка файла после нее

Начнем с довольно серьезного и имеющего практическую ценность примера. Относительно недавно ко мне обратился клиент, которому нужно было доработать существующую систему контроля транспорта (интересующимся могу открыть секрет — СКТ «Глобус»), а именно создать несколько нестандартных отчетов на базе существующих стандартных.

Беда в том, что никакого API у системы нет, но зато есть ссылки, по которым можно получить стандартные отчеты в CSV-формате. Казалось бы, что может быть проще? Взять да и получить нужный CSV-файл! Но не тут-то было! Дело в том, что кому угодно система CSV-файлы не отправляет (это и понятно — тогда бы и конкуренты смогли узнать о перемещениях транспорта клиента), и чтобы получить этот файл с отчетом, нужно сначала зарегистрироваться на сайте.

Поискав в Интернете, можно найти и примеры авторизации на сайте, и примеры загрузки файлов с помощью `cURL`. Но мало какой пример объясняет, как скачать файл именно после авторизации. Проблема в том, что при повторном запросе `cURL` не передаются ранее полученные cookies, и сервер на той стороне «думает», что вы — новый пользователь.

Поэтому при авторизации нужно сохранить cookies в файл, а затем использовать его при установке нового запроса — на получение файла. Написанный мной соответствующий сценарий приводится в листинге 35.1.

### Листинг 35.1. Авторизация на сайте и загрузка файла после авторизации

```
<?php
/*                               Шаг 1: Авторизация                               */

$curl = curl_init();              // Инициализация cURL

/* Далее в произвольном порядке устанавливаем опции соединения */

// URL удаленного сервера, точнее, URL сценария авторизации.
curl_setopt($curl, CURLOPT_URL, 'http://62.149.16.25:8080/login');
```

```
// Устанавливаем имя файла, в котором будем сохранять Cookies
curl_setopt($curl, CURLOPT_COOKIEJAR, 'cook.txt');
// Читать Cookies будем из этого же файла
curl_setopt($curl, CURLOPT_COOKIEFILE, 'cook.txt');
// Представляем как браузер Opera - пусть сервер думает,
// что мы - обычный пользователь
curl_setopt($curl, CURLOPT_USERAGENT, "Opera/10.00 (Windows NT 5.1; U; ru)
Presto/2.2.0");

// Установим эту опцию, чтобы PHP завершал работу в случае ошибки
// Ошибкой считается код ответа HTTP выше 300
curl_setopt($curl, CURLOPT_FAILONERROR, 1);

// Устанавливаем referer - адрес последней активной страницы
curl_setopt($curl, CURLOPT_REFERER, 'http://62.149.16.25:8080/');
// Тайм-аут (в секундах) операции
curl_setopt($curl, CURLOPT_TIMEOUT, 3);
// Метод POST
curl_setopt($curl, CURLOPT_POST, 1);
// Указываем логин и пароль для входа, а также поля формы,
// использующиеся для передачи логина и пароля
curl_setopt($curl, CURLOPT_POSTFIELDS,
'http://62.149.16.25:8080/nmain/login&session-login=логин&session-
password=пароль');

// Установите эту опцию в значение, отличное от 0, чтобы заголовок
// ответа включался в вывод.
// Из сообщений отладки можно включить эту опцию
// В уже готовом сценарии установите значение 0
curl_setopt($curl, CURLOPT_HEADER, 1);

// Мы не используем сертификаты и разрешаем редиректы
curl_setopt ($curl, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt ($curl, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($curl, CURLOPT_FOLLOWLOCATION, 1);
// Данный параметр нужен, чтобы curl возвращал данные, а не
// выводил их в браузер
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
// Выполняем запрос и его результат записываем в переменную
$html = curl_exec($curl);
// Обрабатываем результат. Я просто вывожу его в браузер
echo $html;

/*          Шаг 2: Загрузка файла          */
// Устанавливаем адрес файла (в нашем случае - файла с отчетом)
$report = 'http://62.149.16.25:8080/export/VEHICLES-LIST-
EXPORT?from_date=11.10.2016&to_date=11.10.2016&ids=53234';
```

```
// URL, который будем получать
curl_setopt($curl, CURLOPT_URL, $report);
// Адрес, откуда мы пришли. Обычно нужно указывать страницу, на которую
// вы попадаете после авторизации
curl_setopt($curl, CURLOPT_REFERER, 'http://62.149.16.25:8080/nmain');
// Внимание: указываем Cookie-файл, полученный ранее
curl_setopt($curl, CURLOPT_COOKIEFILE, 'cook.txt');
// Представляемся как браузер
curl_setopt($curl, CURLOPT_USERAGENT, "Opera/10.00 (Windows NT 5.1; U; ru)
Presto/2.2.0");
// Здесь уже используем метод GET
curl_setopt($curl, CURLOPT_POST, 0);
// Отображаем заголовок ответа для отладки
curl_setopt($curl, CURLOPT_HEADER, 1);
// Остальное все так же, как и ранее
curl_setopt ($curl, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt ($curl, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($curl, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
// Выполняем запрос
$data = curl_exec($curl);
// Обрабатываем данные, я просто вывожу в браузер
echo $data;
?>
```

### ПРИМЕЧАНИЕ

Значимые листинги из этой главы вы найдете в каталоге Glava\_35 сопровождающего книту электронного архива (см. *приложение 4*).

Как видите, все здесь довольно несложно. Когда вы станете адаптировать этот сценарий под свои нужды, вам надо будет правильно заполнить данные формы, а именно поля, которые используются для передачи имени пользователя и пароля, а также адрес сценария авторизации. Все эти данные узнать тоже несложно — откройте исходный текст страницы (рис. 35.3). Здесь видно, как называется сценарий

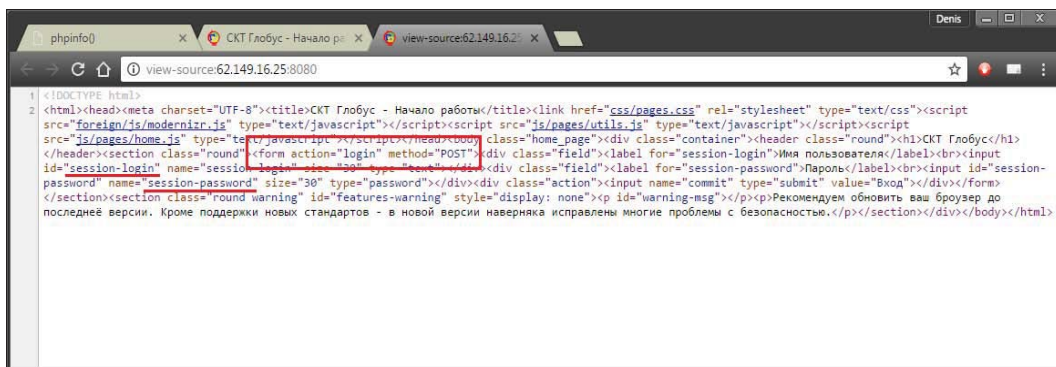


Рис. 35.3. Исследуем форму авторизации

авторизации, какой метод передачи формы используется (POST), как называются поля с именем пользователя и паролем.

### 35.3. Замена функции *file\_get\_contents()* с помощью cURL

На некоторых хостингах функция `file_get_contents()` отключена. Но если на хостинге установлено расширение cURL, реализовать аналог такой функции достаточно просто. Код функции `curl_file_get_contents()`, которая работает точно так же, как и ее исходный аналог, приведен в листинге 35.2.

**Листинг 35.2. Функция `curl_file_get_contents()`**

```
<?php
function curl_file_get_contents($url) {
    $ch = curl_init();

    curl_setopt($ch, CURLOPT_HEADER, 0);
    // Этот параметр нужен, чтобы curl возвращал данные, а не
    // выводил их в браузер
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_URL, $url);

    $data = curl_exec($ch);
    curl_close($ch);

    return $data;
}
?>
```

### 35.4. Загрузка файла через FTP

В PHP есть библиотека FTP `library`, но вы можете использовать cURL для работы с FTP, если вам удобнее. В листинге 35.3 приводится пример загрузки файла на FTP-сервер.

**Листинг 35.3. Загрузка файла на FTP-сервер с помощью cURL**

```
<?php
// Открываем файл
$fname = "some_file.txt";
$file = fopen($fname, "r");

// URL содержит большую часть нужной информации, а именно:
// имя пользователя, пароль, номер порта (21), путь к файлу на сервере
$url = "ftp://username:password@example.com:21/path/to/new/file";
```

```
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

// Устанавливаем опции
curl_setopt($ch, CURLOPT_UPLOAD, 1);
curl_setopt($ch, CURLOPT_INFILE, $fp);
curl_setopt($ch, CURLOPT_INFILESIZE, filesize($fname));

// Устанавливаем режим ASCII (текстовый)
curl_setopt($ch, CURLOPT_FTPASCII, 1);

$output = curl_exec($ch);
curl_close($ch);
?>
```

## 35.5. Проверка доступности сайта

Расширение cURL можно с успехом использовать для автоматизации проверки доступности сайта. Все достаточно просто — нужно обратиться к серверу, и если получен ответ, значит, сайт доступен. В противном случае — сайт «упал». Для проверки доступности разработана функция `isSiteAvailable()`, которая возвращает `true`, если сайт доступен, и `false` — в противном случае. Функции нужно передать URL сайта (листинг 35.4).

### Листинг 35.4. Проверка доступности сайта

```
<?php

// Возвращает true, если сайт доступен
function isSiteAvailable($url)
{
    // Проверка на правильность URL
    // Если URL неправильный, сразу возвращаем false
    if(!filter_var($domain, FILTER_VALIDATE_URL))
    {
        return false;
    }

    // Инициализация curl
    $curl = curl_init($domain);
    curl_setopt($curl, CURLOPT_CONNECTTIMEOUT, 10);
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_NOBODY, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
```

```
// Получаем ответ
$response = curl_exec($curl);

curl_close($curl);

if ($response) return true;

return false;
}

if (isSiteAvailible('http://example.com'))
{
    echo "Сайт работает!";
}
else
{
    echo "Сайт не отвечает!";
}

?>
```

Пример довольно простой — мы просто выводим информацию о доступности сайта. Вы же можете использовать функцию `mail()` для отправки сообщения на электронный адрес администратора сайта, если сценарий обнаружит, что сайт не отвечает.

Надеюсь, что я вас заинтересовал. Дальнейшее изучение библиотеки cURL вы сможете продолжить самостоятельно — благо документации и примеров в Интернете хватает.



## **II. РАЗДЕЛ 8**

### **Введение в Laravel**

<b>Глава 36.</b>	Фреймворк или чистый PHP-код?
<b>Глава 37.</b>	Установка Laravel на VDS с Ubuntu Linux
<b>Глава 38.</b>	Определяем маршруты
<b>Глава 39.</b>	Пишем контроллер
<b>Глава 40.</b>	Создаем представление
<b>Глава 41.</b>	Запрос и ответ
<b>Глава 42.</b>	Работа с данными



## ГЛАВА 36

# Фреймворк или чистый PHP-код?

### 36.1. Что такое фреймворк?

PHP — это очень мощный и популярный среди веб-разработчиков язык программирования. Большая часть современных веб-сайтов реализована именно на PHP.

При разработке сайтов каждый разработчик рано или поздно сталкивается с ситуацией, когда в разных сайтах приходится использовать одни и те же конструкции кода. Кто-то создает собственные библиотеки кода, кто-то для нового сайта пишет новый код (таких мало, но они есть). В общем, программистам приходится изобретать свое колесо повторного использования кода.

Однако зачем изобретать колесо, если можно просто использовать *фреймворк*? К тому же это не только удобно, но безопасно и эффективно. Да, вы можете создать собственную библиотеку кода, но вам придется заниматься обеспечением безопасности кода (не в том смысле, чтобы код никто не украл, а в том, что он может содержать ошибки, которые могут привести к взлому сайта). То же самое можно сказать и об эффективности. Фреймворки содержат уже оптимизированный другими программистами код, который снижает нагрузку на сервер. Можно всем этим заниматься самому, и даже есть вероятность, что вы напишете код безопаснее и эффективнее, но на все это нужно время. Если вы в нем не ограничены, то почему бы и нет? Но обычно задан жесткий лимит времени, а фреймворки позволяют его существенно экономить.

Итак, фреймворк — это программная платформа, состоящая из:

- *инструментария* — набора быстро интегрирующихся компонентов. Это означает, что вам придется писать меньше кода, следовательно, меньше риска допустить ошибку;
- *методологии* — структурный подход к проектированию приложения может показаться довольно сложным, особенно в первое время. Но со временем вы поймете, что он позволяет разработчикам работать над самыми сложными проектами быстро и эффективно. В итоге разработчик может сконцентрироваться на приложении (веб-приложение — это тоже приложение), а не над архитектурой кода.

Что лучше — использовать фреймворк или же чистый PHP-код? На дворе 2023 год, и мало кто пишет на чистом PHP-коде. Для приложений средней сложности и выше разработчики стараются использовать один из предпочитаемых (либо выбранных заказчиком) фреймворков. Код без фреймворка пишут либо новички (самые простые сценарии), либо профессионалы, создающие подобие собственного фреймворка. Обычному разработчику проще взять один из готовых фреймворков и существенно ускорить разработку веб-приложения.

Используя любой из фреймворков, вы получите уже готовый каркас приложения и избавитесь от некоторой головной боли, связанной с безопасностью вашего приложения, — если, конечно, будете использовать средства, предоставляемые фреймворком, а не будете писать свой код, минуя их. Говоря о безопасности, имеется в виду обработка запросов, а также аутентификация пользователей — как правило, такая функциональность уже встроена в любой из фреймворков, причем обычно на выбор предоставляется несколько способов аутентификации, и вы можете выбрать более подходящий для вашего проекта. Также вы получите красивые URL и улучшенную интеграцию с современным фронтендом — благодаря встроенной поддержке JSON-запросов и ответов.

В этой главе мы продолжим разговор о фреймворке Laravel (работа с базой данных в Laravel была описана в *главе 21*). Это один из лучших фреймворков на сегодняшний день. Говорить о том, какой самый лучший не стану, поскольку пальма первенства переходит из года в год от фреймворка к фреймворку и зачастую зависит от личных предпочтений разработчиков. На сегодняшний день существуют три самых значимых фреймворка: Laravel, Symfony и Yii. Если вам этого достаточно, можете смело переходить к следующей главе. А если нет — в следующем разделе мы рассмотрим особенности других фреймворков.

## 36.2. Обзор популярных PHP-фреймворков

### 36.2.1. Zend Framework и The Laminas Project

Zend Framework — один из самых древних фреймворков. Разработчиком его является компания Zend Technologies. Да, это та самая компания, которая разработала ядро PHP. То есть Zend Framework — это фреймворк от разработчиков PHP.

Он подходит для довольно крупных проектов и с успехом справляется с их реализацией. Так, на базе этого фреймворка построена популярная e-commerce платформа Magento.

Интерес к Zend Framework на фоне более молодых его собратьев стал угасать, и разработчики переименовали его в The Laminas Project — в надежде, что небольшой ребрендинг положительно скажется на востребованности их продукта. Получить фреймворк можно на официальном сайте <https://getlaminas.org/> или сразу из репозитория: <https://github.com/laminas/laminas-mvc-skeleton>.

## 36.2.2. Laravel

Очень простой в изучении и надежный каркас для вашего будущего приложения. Автор этой книги перешел на него с фреймворка Symfony. Проще, «легче», понятнее — если вкратце, то именно так можно охарактеризовать впечатления после нескольких лет использования.

Laravel существенно облегчает аутентификацию, управление сессиями, маршрутами и другие задачи. Есть для него хорошая и понятная документация. В следующей главе мы подробно рассмотрим процесс его установки.

## 36.2.3. Symfony

Всемирно известный фреймворк, на базе которого созданы сервисы вроде Sixt, NatGeo Play от National Geographic, BlaBlaCar и др. Symfony распространяется по лицензии MIT, а не GPL. Если не вдаваться в подробности, то он тоже бесплатный.

Первый выпуск Symfony состоялся в 2005 году, т. е. за 18 лет продукт не только не потерял своей популярности, но и уверенно держится на плаву. Так, восьмая версия популярной CMS Drupal была переписана именно на Symfony в 2021 году — т. е. не так уж и давно.

За 18 лет существования Symfony сформировалось огромное сообщество разработчиков, поэтому, если у вас возникнут проблемы с его использованием, вам будет к кому обратиться.

Когда-то Symfony считался более медленным, чем Laravel. Но в последних версиях ситуация с производительностью была исправлена, и сейчас оба фреймворка демонстрируют неплохую производительность. Laravel все еще лучше, но Symfony тоже хорошо подтянули по сравнению с тем, как было раньше. При небольших нагрузках (например, 10 запросов, 1 пользователь, веб-сервер Nginx) производительность примерно одинаковая: 7,5 секунды Symfony против 6,5 у Laravel. Когда же нагрузка существенно выше, то Symfony начинает сдавать: 658 секунд против 211 у Laravel — при нагрузке 1000 запросов и 10 пользователей. В случае с Apache Symfony показывает чуть лучшую производительность: 638 секунд при том же уровне нагрузки. Это обусловлено тем, что PHP встраивается в Apache в виде модуля, а не через сервис PHP-FPM. Опять-таки, говоря о производительности, нужно упомянуть не только условия, при которых она тестировалась, но и версию PHP, поскольку очень сильно производительность PHP-приложения зависит от версии самого интерпретатора. Версии 7.4 и 8.0 примерно одинаковые по производительности, но версия 7.2 значительно хуже, а 8.1 лучше, чем 8.0 (хоть и ненамного). Так что статистикой, как и данными с производительностью, можно манипулировать, что и делают некоторые сайты.

## 36.2.4. Yii

Быстрый, компактный и безопасный фреймворк. «Из коробки» содержит инструменты для управления кешированием, аутентификацией, распределенным управле-

нием учетными записями, инструменты для проверки форм. Также к достоинствам продукта нужно отнести простоту его установки.

У этого фреймворка есть одна особенность — он ориентирован на сайты сообществ и социальные сети. На его базе можно построить и другие приложения, но все же основная его ориентация — на такие проекты.

### 36.2.5. CodeIgniter

Популярный фреймворк с открытым кодом. Старая ветка CodeIgniter распространялась по закрытой лицензии BSD, но текущая его ветка перелицензирована под MIT, что позволяет безо всяких проблем использовать этот продукт не только для себя, но и в коммерческих проектах.

Фреймворк этот довольно минималистичный, и много функций, которые есть в других фреймворках, вам предстоит реализовать самостоятельно. Так, он не имеет поддержки Eloquent ORM — механизма, который позволяет взаимодействовать с базой данных с использованием единой модели. А в Laravel эта возможность встроена. Зато у него есть поддержка MongoDB<sup>1</sup> (не SQL база данных) и Microsoft BI (инструмент для бизнес-анализа). Также в CodeIgniter нет поддержки шаблонизатора. Правда, существует возможность его интеграции со Smarty, однако это нужно делать самому. Сразу после установки в CodeIgniter никаких шаблонизаторов нет.

\* \* \*

Существует множество других фреймворков: Kohana, CakePHP, Phalcon, Aura и др. У каждого есть свои особенности, и выбор конкретного фреймворка зависит от предпочтений разработчика и поставленной задачи. Однако Laravel и Symfony — два универсальных фреймворка «на каждый день», позволяющие покрыть 99% задач современных PHP-разработчиков. Изучив один из этих фреймворков, вам уже больше не захочется писать чистый PHP-код, поскольку использование фреймворков не только удобнее, но еще и существенно экономит время в проектах средней сложности и выше.

---

<sup>1</sup> При необходимости поддержку MongoDB можно интегрировать в Laravel: <https://www.mongodb.com/compatibility/mongodb-laravel-intergration>.



## ГЛАВА 37

# Установка Laravel на VDS с Ubuntu Linux

### 37.1. Выбор места для установки

В серьезных проектах вместо обычного хостинга используются виртуальные выделенные серверы — VDS. В этой главе мы рассмотрим установку фреймворка Laravel как раз на такой сервер. В большинстве случаев VDS поставляются на базе Debian-ориентированных систем вроде Ubuntu, но даже если вам достался сервер с RPM-системой пакетов, в большинстве случаев все, что у вас поменяется, — это команды установки пакетов: вместо `apt` нужно будет использовать `dnf`.

Если вы начинающий разработчик и у вас еще нет собственного VDS, вы можете выбрать один из следующих вариантов:

- ❑ установите Linux — или на свой физический компьютер, или в виртуальную машину, которая будет работать под управлением Windows. Вы потратите немного времени, зато у вас будет полноценная среда для PHP-разработки и совершенно бесплатная;
- ❑ некоторые облачные сервисы поставляют свои услуги бесплатно определенное время. Так, на **cloud.google.com** вы можете получить VDS совершенно бесплатно на срок до 3 месяцев. Вам предоставят 300\$ кредита (возвращать их не придется) и 3 месяца бесплатного использования ресурсов. Важно, чтобы вы за эти три месяца не израсходовали кредитные средства, иначе услуга будет приостановлена раньше чем через 3 месяца.

### 37.2. Установка необходимого ПО

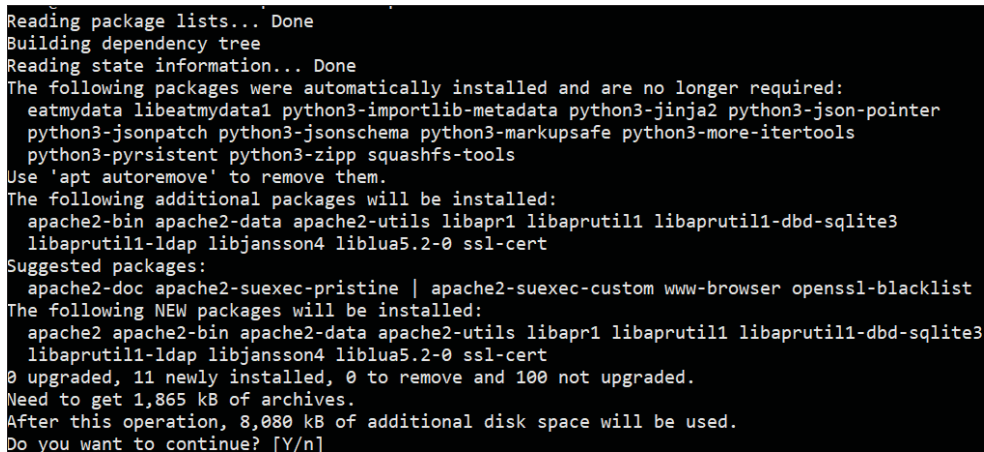
#### 37.2.1. Установка веб-сервера Apache

По последним тестам Laravel быстрее работает именно под управлением Apache. Возможно, сказывается то, что поддержка PHP встраивается в сам сервер в виде модуля.

Для установки веб-сервера в среде Linux<sup>1</sup> введите команды:

```
sudo apt update
sudo apt install apache2
sudo a2enmod rewrite
```

Первая команда обновляет список пакетов. Ее нужно ввести один раз — сразу после получения VDS. Потом вы можете вводить ее раз в несколько месяцев, если возникнет в этом необходимость. Вторая команда, собственно, устанавливает сам веб-сервер (рис. 37.1). А вот третья — включает модуль `rewrite`. Важно сделать это прямо сейчас, иначе потом вы про него забудете, а затем потратите кучу времени на поиск того, почему Laravel не работает так, как нужно.



```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  eatmydata libeatmydata1 python3-importlib-metadata python3-jinja2 python3-json-pointer
  python3-jsonpatch python3-jsonschema python3-markupsafe python3-more-itertools
  python3-pyrsistent python3-zipp squashfs-tools
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libjansson4 liblua5.2-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser openssl-blacklist
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libjansson4 liblua5.2-0 ssl-cert
0 upgraded, 11 newly installed, 0 to remove and 100 not upgraded.
Need to get 1,865 kB of archives.
After this operation, 8,080 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Рис. 37.1. Установка apache2

Чтобы вам было удобнее работать с VDS, а именно перемещаться по его каталогам и редактировать файлы, установите пакет `mc`, содержащий классический двухпанельный файловый менеджер:

```
sudo apt install mc
```

Для управления веб-сервером используются команды:

```
sudo systemctl start apache2
sudo systemctl stop apache2
sudo systemctl restart apache2
sudo systemctl status apache2
```

Назначение этих команд, думаю, понятно: запуск, останов, перезапуск и просмотр состояния сервера. Пока запускать сервер не нужно — мы к нему еще вернемся, а пока оставьте его в покое.

---

<sup>1</sup> Напомню, что в главе 1 мы устанавливали Apache под Windows.

## 37.2.2. Установка PHP и его расширений

Команда для установки PHP:

```
sudo apt install php
```

В процессе установки пакета `php` будут установлены и вспомогательные пакеты. Один из них: `libapache2-mod-php` — именно этот пакет обеспечивает поддержку PHP веб-сервера Apache.

Затем установим необходимые расширения:

```
sudo apt install php-bcmath php-json php-mbstring php-xml php-tokenizer php-zip php-mysql
```

Откройте (в редакторе `mcedit` или в любом другом) конфигурационный файл `/etc/php/<версия>/php.ini` и отредактируйте некоторые параметры (рис. 37.2):

```
memory_limit = 256M
max_execution_time = 120
```

Затем, если у вашего VDS есть доменное имя, его нужно прописать в двух файлах: прежде всего, в конфигурации Apache — в директиве `ServerName` файла `/etc/apache2/sites-enabled/000-default.conf`, а также в файле `/etc/hosts`:

<IP-адрес> <доменное имя>

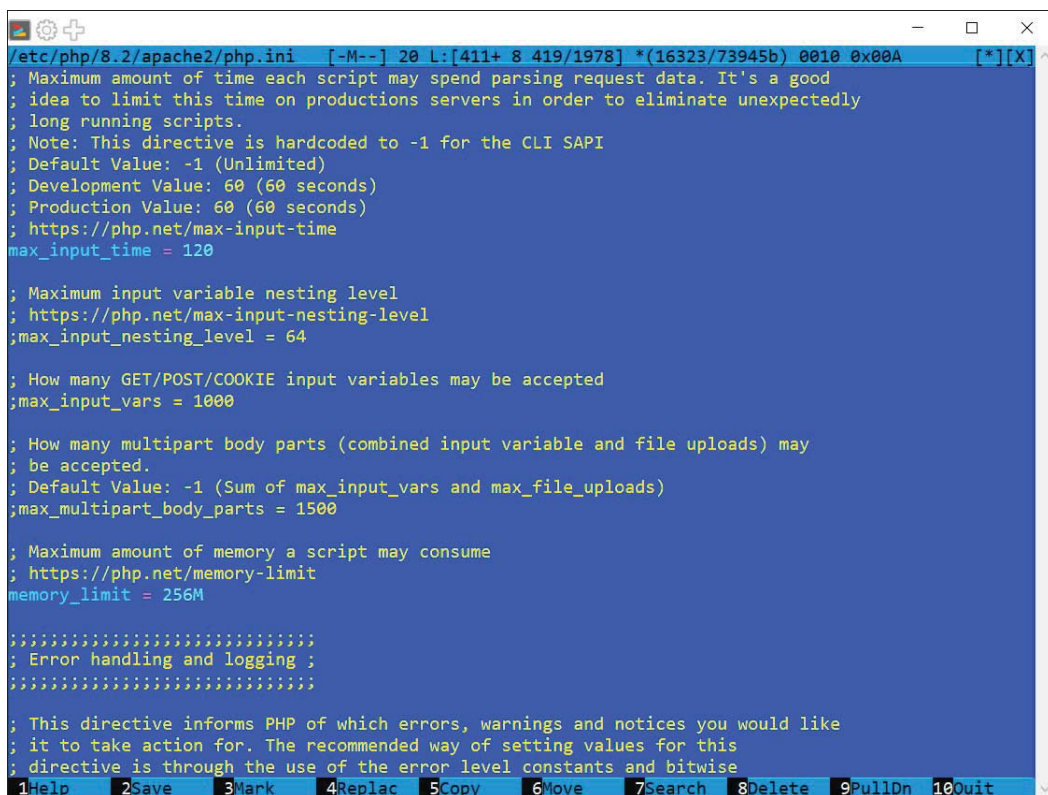


Рис. 37.2. Редактирование файла конфигурации

Доменным именем желательно обзавестись — оно необходимо для корректной работы механизма сессий в Laravel. Обычно каждому VDS назначается бесплатное имя от самого облачного провайдера — совсем необязательно для этого покупать домен. На локальной машине вы можете использовать имя `example.com` — только пропишите в файле `/etc/hosts` строку:

```
127.0.0.1 example.com
```

Далее перейдите в каталог `/var/www/html` и создайте файл `info.php` следующего содержания:

```
<?php
phpinfo();
?>
```

Это простейший сценарий, выводящий информационную страничку. Вот теперь все готово к запуску веб-сервера:

```
sudo systemctl start apache2
```

PHP Version 8.1.2-1ubuntu2.11	
System	Linux instance-1 5.15.0-1030-gcp #37-Ubuntu SMP Tue Feb 14 19:37:08 UTC 2023 x86_64
Build Date	Feb 22 2023 22:56:18
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.1/apache2
Loaded Configuration File	/etc/php/8.1/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.1/apache2/conf.d
Additional .ini files parsed	/etc/php/8.1/apache2/conf.d/10-mysqlnd.ini, /etc/php/8.1/apache2/conf.d/10-opcache.ini, /etc/php/8.1/apache2/conf.d/10-pdo.ini, /etc/php/8.1/apache2/conf.d/15-xml.ini, /etc/php/8.1/apache2/conf.d/20-bcmath.ini, /etc/php/8.1/apache2/conf.d/20-bz2.ini, /etc/php/8.1/apache2/conf.d/20-calendar.ini, /etc/php/8.1/apache2/conf.d/20-ctype.ini, /etc/php/8.1/apache2/conf.d/20-curl.ini, /etc/php/8.1/apache2/conf.d/20-dom.ini, /etc/php/8.1/apache2/conf.d/20-exif.ini, /etc/php/8.1/apache2/conf.d/20-ffi.ini, /etc/php/8.1/apache2/conf.d/20-fileinfo.ini, /etc/php/8.1/apache2/conf.d/20-ftp.ini, /etc/php/8.1/apache2/conf.d/20-gd.ini, /etc/php/8.1/apache2/conf.d/20-gettext.ini, /etc/php/8.1/apache2/conf.d/20-gmp.ini, /etc/php/8.1/apache2/conf.d/20-iconv.ini, /etc/php/8.1/apache2/conf.d/20-mbstring.ini, /etc/php/8.1/apache2/conf.d/20-mysql.ini, /etc/php/8.1/apache2/conf.d/20-pdo_mysql.ini, /etc/php/8.1/apache2/conf.d/20-phar.ini, /etc/php/8.1/apache2/conf.d/20-posix.ini, /etc/php/8.1/apache2/conf.d/20-readline.ini, /etc/php/8.1/apache2/conf.d/20-shmop.ini, /etc/php/8.1/apache2/conf.d/20-simplexml.ini, /etc/php/8.1/apache2/conf.d/20-sockets.ini, /etc/php/8.1/apache2/conf.d/20-sysvmsg.ini, /etc/php/8.1/apache2/conf.d/20-sysvsem.ini, /etc/php/8.1/apache2/conf.d/20-sysvshm.ini, /etc/php/8.1/apache2/conf.d/20-tokenizer.ini, /etc/php/8.1/apache2/conf.d/20-xmlreader.ini, /etc/php/8.1/apache2/conf.d/20-xmlwriter.ini, /etc/php/8.1/apache2/conf.d/20-xsl.ini, /etc/php/8.1/apache2/conf.d/20-zip.ini
PHP API	20210902
PHP Extension	20210902
Zend Extension	420210902
Zend Extension Build	API420210902.NTS
PHP Extension Build	API20210902.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

Рис. 37.3. Все работает!

После запуска сервера откройте браузер и попробуйте обратиться к серверу по доменному имени:

```
http://доменное_имя/info.php
```

или по IP-адресу:

```
http://IP-адрес/info.php
```

Если вы все сделали правильно, то увидите информационную страничку (рис. 37.3).

Теперь все готово к установке Laravel. Если ваше приложение будет использовать базу данных MySQL или любую другую, установите эту базу. Процесс установки MySQL на VDS был описан в *главе 18*.

### 37.2.3. Установка Laravel

Для установки всего необходимого достаточно выполнить одну команду (из каталога `/var/www/`):

```
cd /var/www
sudo composer create-project --prefer-dist laravel/laravel laravel
```

Последний параметр `laravel` означает, что фреймворк будет установлен в каталог `/var/www/laravel`. После завершения установки вам нужно только выставить правильные права на файлы фреймворка. Их владельцем должен быть тот пользователь, от имени которого запускается веб-сервер и PHP. По умолчанию это `www-data`:

```
sudo chown -R www-data:www-data laravel
```

Затем нужно отредактировать конфигурацию Apache (рис. 37.4):

```
<VirtualHost *:80>
    ServerName доменное_имя

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/laravel/public

    <Directory "/var/www/laravel/public">
        Options FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Откройте файл конфигурации Laravel `/var/www/laravel/.env` и отредактируйте следующие параметры:

```
APP_URL=http://доменное_имя
```

```
DB_CONNECTION=mysql
```

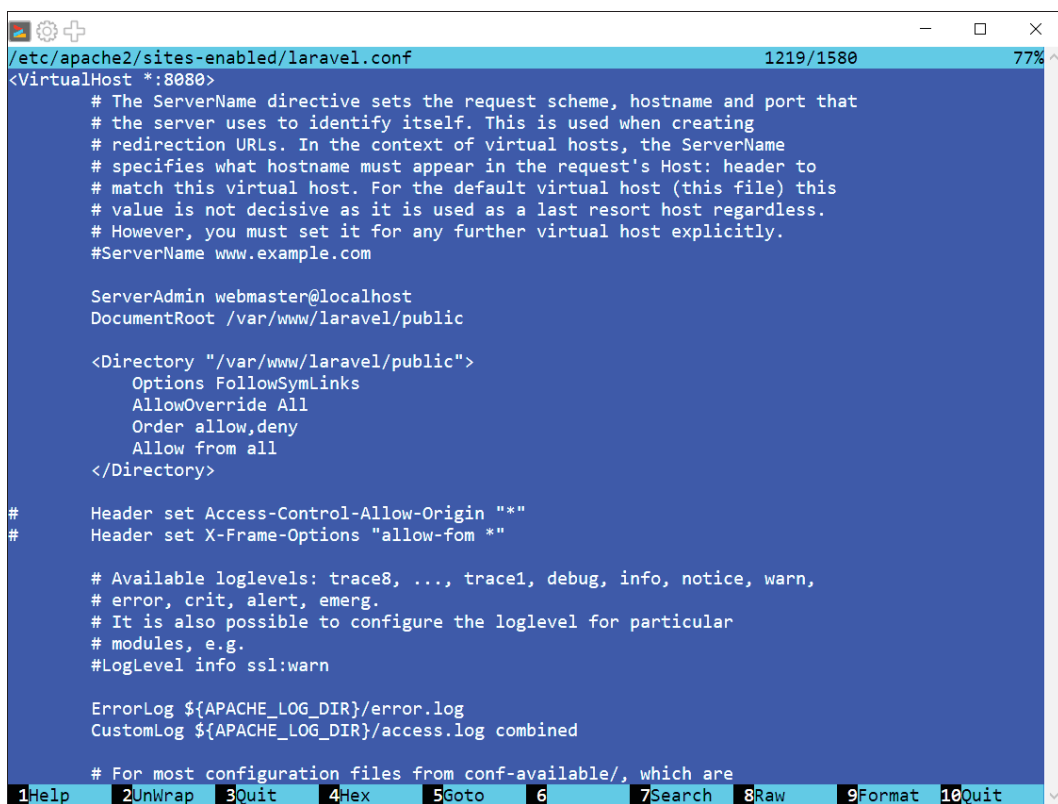
```
DB_HOST=127.0.0.1
```

```
DB_PORT=3306
```

```
DB_DATABASE=laravel
```

```
DB_USERNAME=root
```

```
DB_PASSWORD=укажите_пароль
```



```
/etc/apache2/sites-enabled/laravel.conf 1219/1580 77%
<VirtualHost *:8080>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/laravel/public

    <Directory "/var/www/laravel/public">
        Options FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>

    #
    Header set Access-Control-Allow-Origin "*"
    Header set X-Frame-Options "allow-fom *"

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
```

Рис. 37.4. Конфигурация Apache

Если БД пока не используется, ее параметры редактировать не нужно. Теперь перезапустите Apache:

```
sudo systemctl restart apache2
```

Обратитесь к вашему сайту в браузере. Если все сделано правильно, тогда вы увидите стандартную страницу Laravel (рис. 37.5).

Если страница не открылась, тогда вы что-то сделали не так. Наиболее вероятные причины:

- ☐ что-то не так с системой DNS. Обратитесь к сайту по IP-адресу. Если вы увидите стандартную страницу Laravel, причина на 100% в системе DNS:

- в настройках домена должна быть А-запись, указывающая на IP-адрес узла, на котором находится Laravel;
  - если вы недавно зарегистрировали домен, нужно дождаться, пока система DNS обновится. Выполните команду (на вашем клиенте, т. е. в Windows) `ipconfig /flushdns` и обратитесь к сайту. Если не помогло, подождите 2–3 часа, снова обновите DNS и обратитесь к сайту. Для ускорения процесса обновления DNS используйте DNS-серверы от Google: 8.8.8.8 и 8.8.4.4;
- ☐ проверьте, что доменное имя правильно указано в настройках Apache и Laravel. На всякий случай перезапустите Apache снова.

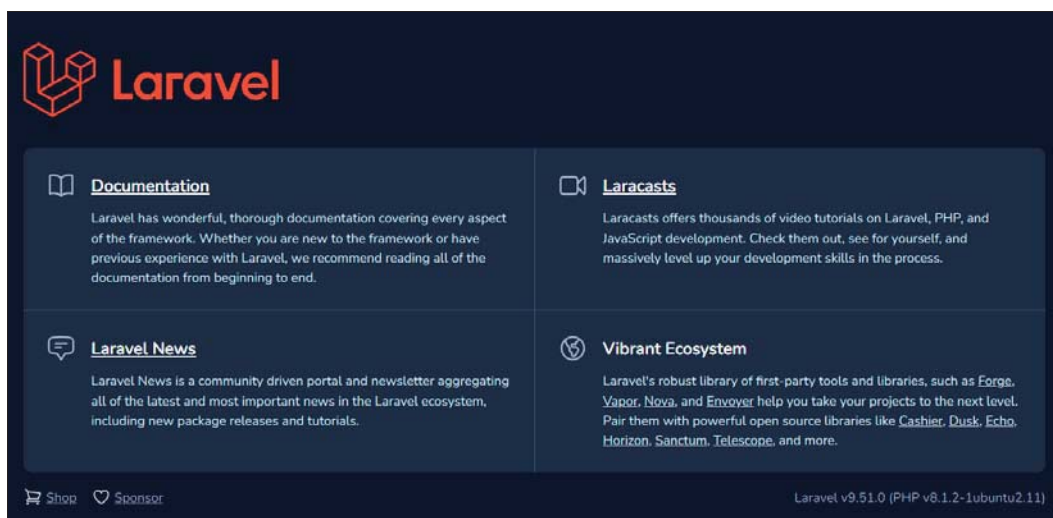


Рис. 37.5. Laravel установлен



## ГЛАВА 38

# Определяем маршруты

### 38.1. Запросы протокола HTTP

Протокол HTTP (HyperText Transfer Protocol) — это протокол передачи гипертекста. Принцип работы по протоколу HTTP следующий: клиент (браузер) отправляет на сервер запрос, сервер обрабатывает его и возвращает ответ. Начнем с запросов, которые могут быть следующими:

- ☐ POST — позволяет создать ресурс на сервере;
- ☐ HEAD — запрашивает версию GET-ответа, содержащую только заголовок;
- ☐ PUT — обновляет ресурс на сервере;
- ☐ PATCH — модифицирует ресурс;
- ☐ DELETE — удаляет ресурс с сервера;
- ☐ OPTIONS — запрашивает у сервера список команд, разрешенных для конкретного адреса.

Пример такого запроса:

```
GET / HTTP/1.1
Host: example.com
Accept: text/html
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36
```

Посмотрим, что он означает. В нашем случае клиент (судя по заголовку `User-Agent`, это популярный браузер Chrome, работающий под Windows 10) пытается получить индексный файл / (он может называться `index.php` или `index.html`. Как именно конкретно — задается в настройках сервера) веб-сервера `example.com`. Наш клиент принимает только данные MIME-типа `text/html`.

Сервер, получив такой запрос, берет содержимое индексного файла и отправляет ответ клиенту, в который будет включено содержимое индексного файла. Если это PHP-сценарий, то благодаря установленным компонентам (в случае с веб-сервером `nginx` — служба `php-fpm`, например) отправляет этот сценарий интерпретатору PHP, тот его обрабатывает и возвращает результат серверу, который, в свою оче-

редь, возвращает этот результат клиенту. В случае с запросом GET ответ будет таким:

```
HTTP/1.1 200 OK
Date: Mon, 03 Apr 2023 10:08:08 GMT
Server: nginx/1.8.1
Content-Type: text/html

<html>
<head><title>Index page</title></head>
...
</html>
```

Кроме самого содержимого запрашиваемого файла, сервер возвращает другую полезную информацию: код ответа (в нашем случае 200), наименование сервера (nginx/1.8.1), дату генерации ответа, а также тип содержимого (text/html). Сервер может возвращать и другие заголовки — все зависит от типа запроса и настроек сервера.

Код ответа может отличаться от 200. Код 200 означает, что файл найден и никаких ошибок при обработке запроса не произошло. Может быть возвращен и код 404 — файл отсутствует на сервере. В этом случае сервер возвращает код 404 и HTML-код страницы, которая специально сконфигурирована именно для таких случаев и содержит текст наподобие «Запрашиваемая страница не найдена».

Другой известный код: 500 — это ошибка на стороне сервера. Обычно такой код сервер возвращает, когда произошла ошибка в PHP-сценарии, но в настройках PHP запрещен вывод информации об ошибках в браузер по соображениям безопасности.

С другими кодами можно ознакомиться по адресу: [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

## 38.2. Модель. Представление. Контроллер

В полновесных книгах, посвященных только фреймворку Laravel или любому другому, этому вопросу посвящается целая глава. У нас такой роскоши нет, но не разобраться, хотя бы вкратце, с паттерном MVC (Model. View. Controller) мы не можем.

Паттерн MVC представляет собой способ организации кода, предполагающий выделение компонентов, отвечающих за разные задачи. Компоненты модели MVC следующие:

- *модель* — компонент, отвечающий за данные, а также определяющий структуру приложения. Например, если вы создаете галерею, то этот компонент будет определять список загруженных фото;
- *представление* — отвечает за взаимодействие с пользователем и внешний вид приложения. В случае с галереей представление будет определять вид списка фото в браузере, а также вид в нем отдельной фотографии. Ведь фото не просто отправляется в браузер — должна быть какая-то страничка, «обрамляющая» фотографию и предоставляющая средства навигации по галерее;

- контроллер — отвечает за связь между моделью и представлением. Код контроллера определяет, как сайт реагирует на действия пользователя. Именно контроллеры задают конечные действия, которые может произвести пользователь с фото: вывести список, просмотреть, отфильтровать, добавить комментарий и т. д. Контроллер является как бы мозгом MVC-приложения.

Пользователь взаимодействует с приложением посредством маршрутов. Например, маршрут `/photos` будет выводить список миниатюр фотографий галереи. Маршрут `/photos/view/<ID>` выведет конкретную фотографию со средствами навигации по галерее. А маршрут `/photos/add-comment/<ID>` позволяет добавить комментарий к фотографии. Количество маршрутов и их конфигурация практически никак не ограничиваются.

## 38.3. Простейшие маршруты.

### Сопоставление маршрута с контроллером

*Маршрут* — это сопоставление интернет-адреса (URL) приложения методам контроллера. У вас может быть несколько контроллеров — например, контроллер для фронтенда, контроллер для панели управления и др. Все зависит от вашего приложения. Никто не запрещает создать 100 контроллеров в одном приложении или же, наоборот, поместить все используемые методы в один контроллер. Но нужно мыслить здраво. Как правило, для каждого отдельного модуля выделяется отдельный контроллер. Например, если вы создаете интернет-магазин, можно выделить такие модули:

- каталог продуктов — выводит каталог магазина, а также страницы отдельных продуктов;
- корзина — управляет корзиной покупателя;
- модуль регистрации — позволяет пользователям зарегистрироваться и войти в личный кабинет, где можно управлять собственными заказами;
- модуль оплаты — управляет оплатой заказа. У вас может быть несколько модулей оплаты — по одному для каждого метода оплаты;
- модуль блога — выводит различные статические страницы (например, страницы с новостями компаниями и акциями), а также главную страницу магазина.

В приложении Laravel можно определять маршруты двух типов: веб-маршруты и API-маршруты. Первые описываются в файле `routes/web.php`, вторые — в `routes/api.php`. Сосредоточимся на обычных веб-маршрутах и рассмотрим два сопоставления:

```
Route::get('/', 'App\Http\Controllers\Controller@Main');  
Route::get('/about', 'App\Http\Controllers\Controller@About');
```

Первая строка говорит о том, что при запросе главной страницы с использованием HTTP-метода GET нужно передать управление методу `Main()` класса `Controller`. Обычно стараются хранить классы в одноименных файлах, поэтому класс

Controller будет храниться в файле Controller.php, который находится в каталоге app/http/Controllers.

Вторая строка аналогична первой. Если будет передан запрос страницы /about, управление будет передано методу About() контроллера Controller.

Иногда, в самых простых случаях, нет смысла выделять целый метод контроллера. Например, нам нужно просто отобразить то или иное представление — оно статическое и не требует получения информации из контроллера. В этом случае мы можем использовать анонимные функции:

```
Route::get('/contacts', function() {  
    return view('contacts');  
});
```

При запросе страницы /contacts будет отображено представление contacts. Обычно представления хранятся в каталоге resources/views, но мы еще поговорим о них.

Выражение Route::get означает, что этот путь должен быть запрошен с помощью метода GET. А что делать, если нам нужно использовать другие HTTP-команды — например, обработать POST-форму? Рассмотрим несколько примеров использования других HTTP-команд:

```
// POST-запросы  
Route::post('/search', 'App\Http\Controllers\Controller@Search' );  
Route::post('/subscribe', 'App\Http\Controllers\Controller@Subscribe' );  
  
// Другие запросы  
Route::put('/put', function () {  
    // обрабатываем PUT-запрос  
});  
Route::delete('/del', function () {  
    // обрабатываем DELETE-запрос  
});  
Route::any('/page', function () {  
    // обрабатываем любой запрос  
});  
Route::match(['get','post'], '/', function () {  
    // обрабатываем только GET/POST-запросы  
});
```

## 38.4. Параметры в маршрутах

Не всегда можно определить заранее известный URL в маршрутах. Например, при выводе новости нужно как-то передать ее ID (например, /news/1), иначе вам придется создавать по одному маршруту для каждой новости, что неудобно — не для того мы Laravel устанавливали.

Рассмотрим пример:

```
Route::get('/news/{nid}',  
['uses'=>'App\Http\Controllers\NewsController@GetPost'] );
```

Если мы собрались обрабатывать маршрут прямо в `web.php`, т. е. в анонимной функции, то конструкция будет такой:

```
Route::get('customers/{id}/orders', function ($id) {  
    // Обработка маршрута, переменная $id содержит параметр  
});
```

В некоторых случаях нам нужно передать более одного параметра — например, когда мы собираемся отобразить контент определенного раздела (задается параметром `cid`) какой-то книги (задается параметром `bid`):

```
Route::get('/blog/sections/{bid}/{cid}',  
    ['uses'=>'App\Http\Controllers\BlogController@GetSectionContent']);
```

В этом случае вы просто указываете в нужном месте название второго параметра. При этом количество параметров не ограничивается — вы можете задать три, четыре и более параметров.

Если параметр является необязательным, используйте вопросительный знак:

```
Route::get('/blog/{y?}', ['uses'=>'App\Http\Controllers\Blog@List']);
```

Здесь параметр `y` означает год. Если он указан, то метод `List()` выведет архив записей за указанный год. А если не указан, тогда `List()` выведет список журналов за последний доступный год.

При обработке маршрута с помощью анонимной функции не забывайте указывать значение по умолчанию для параметра — например:

```
Route::get('/blog/{y?}', function ($year = 2022) {  
    // Обработка маршрута  
});
```

Laravel позволяет использовать регулярные выражения при определении маршрутов. Например, если мы хотим, чтобы передаваемый параметр был числом, то можем использовать следующую конструкцию:

```
Route::get('news/{id}', function ($id) {  
    // Какие-то действия  
})->where('id', '[0-9]+');
```

Если у нас несколько параметров, то мы можем задать несколько ограничений:

```
Route::get('news/{id}/{s}', function ($id, $s) {  
    // Обработка маршрута  
})->where(['id' => '[0-9]+', 's' => '[A-Za-z]+']);
```

## 38.5. Имена маршрутов

Laravel позволяет ссылаться на маршрут в любом месте приложения. Проще всего ссылаться на маршрут по имени. Задать имя маршрута можно так:

```
Route::get('news/{id}', 'NewsController@Show')-> name('news.show');
```

Затем в представлении можно обратиться к маршруту так:

```
<a href="<?php echo route('news.show', ['id'=>15]); ?>">
```

Здесь мы ссылаемся на маршрут `/news/15`.

Вот еще пример:

```
route('news.reader', [2023, 3])  
// маршрут /news/2023/3
```

## 38.6. Префиксы маршрутов

Очень часто маршруты имеют один и тот же префикс. Например, у нас есть часть маршрутов, относящаяся к новостям:

```
Route::get('/news/list', 'App\Http\Controllers\NewsController@List');  
Route::get('/news/sections/{year}/{month}', ['uses'=>'App\Http\Controllers\  
NewsController@GetSectionContent']);  
Route::get('/news/view/{id}', ['uses'=>'App\Http\Controllers\  
NewsController@View']);
```

У каждого из этих маршрутов есть префикс `/news`. Мы можем упростить набор маршрутов и определить общий префикс:

```
Route::prefix('/news')->group(function() {  
    Route::get('/list', 'App\Http\Controllers\NewsController@List');  
    Route::get('/sections/{year}/{month}',  
        'uses'=>'App\Http\Controllers\NewsController@GetSectionContent']);  
    ...  
});
```

## 38.7. Маршруты новостной страницы

Сейчас мы определим несколько маршрутов для новостной страницы вашего приложения:

```
Route::prefix('/news')->group(function() {  
    Route::get('/list', 'App\Http\Controllers\NewsController@List');  
    Route::get('/archive/{year}/{month}',  
        'uses'=>'App\Http\Controllers\NewsController@GetArchiveContent']);  
    Route::get('/view/{id}',  
        'uses'=>'App\Http\Controllers\NewsController@View']);  
    Route::post('/add-comment',  
        'App\Http\Controllers\NewsController@AddComment')  
});
```

Первый маршрут выводит список последних новостей, второй — позволяет просмотреть новости за определенный месяц/год, третий — просмотреть конкретную новость по ее ID, а последний — добавить комментарий к новости. В следующей главе мы создадим контроллер, который реализует все эти действия.



## ГЛАВА 39

# Пишем контроллер

### 39.1. Создание контроллера

Контроллеры вашего приложения находятся в каталоге `app/Http/Controllers`. По умолчанию там есть один файл — `Controller.php`. Вы можете использовать его, а можете создать собственный. Рекомендуется руководствоваться здравым смыслом и создавать отдельный контроллер для отдельного логического блока вашего приложения. Например, нужна на сайте новостная страница, т. е. раздел, который будет информировать посетителей о новостях компании, значит, нужно создать для нее отдельный контроллер.

В *главе 38* мы определили маршруты, ссылающиеся на контроллер `NewsController`. Создать такой контроллер можно с помощью команды `artisan` или вручную. Первый вариант проще. Перейдите в каталог приложения (у нас это `/var/www/laravel`) и введите команду:

```
php artisan make:controller NewsController
```

Второй вариант — вы проделаете все, что проделывает `artisan`-команда, но вручную. Первым делом нам нужно перейти в каталог `/var/www/laravel/app/http/Controllers` и создать файл `NewsController.php`:

```
touch NewsController.php
```

Далее нужно добавить в этот файл PHP-код из листинга 39.1.

#### Листинг 39.1. «Болванка» для нового контроллера

```
<?php

namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;
use Illuminate\Http\Request;
```

```
class NewsController extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
}
```

Этот контроллер ничего не делает — в нем не определен ни один из методов. Но есть два важных момента: пространство имен и название класса — `NewsController`. Именно это название используется при определении маршрута в файле `web.php` (см. главу 38). Название файла может отличаться от этого названия, но мы будем придерживаться правила: «один контроллер — один файл», а имя файла при этом будет совпадать с названием класса, чтобы мы не запутались. Пространство имен будет одним и тем же для всех контроллеров (если вам не нужно иного), поэтому оставьте его без изменения.

## 39.2. Простой метод: *List()*

Представим, что у нас есть таблица `news` со следующими полями:

- ❑ `id` — числовой идентификатор новости;
- ❑ `dt` — дата и время создания новости;
- ❑ `title` — заголовок новости;
- ❑ `txt` — контент новости.

Другие поля для нашего примера не нужны, хотя на практике они наверняка будут. Метод `List()` должен вывести последние 10 новостей из этой таблицы.

Первым делом нам нужно подключить класс `DB`:

```
use DB;
```

Сам метод `List()` помещаем внутрь класса:

```
class NewsController extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;

    public function List() {
        $news = DB::table('news')->orderBy('id', 'desc')->limit(10)->get();
        return view('list',
            ['news' => $news]
        );
    }
}
```

Метод довольно прост. Он получает последние 10 записей из таблицы `news` и передает их в представление `list` — это шаблон, определенный в файле `/var/www/html/resources/views/list.*`. Если не делать никакой обработки информации, а просто вернуть представление, то наш метод будет еще проще:

```

public function List() {
    return view('list');
}

```

### 39.3. Методы с параметрами: *View()* и *GetArchiveContent()*

Метод *View()* возвращает контент выбранной новости и передает ее в специально подготовленное для этого представление. Идентификатор новости передается как обычный аргумент для функции *View()*:

```

public function View($id) {
    $post = DB::table('news')->where('id',$id)->get();
    if (is_object($post))
        return view('view',
            ['title' => $post->title,
             'content' => $post->txt,
             'date' => $post->dt]
        );
    else
        abort(404, 'Not found');
}

```

Здесь мы получаем идентификатор статьи в переменной *\$id* и пытаемся найти статью с таким идентификатором. Если мы ее не находим, т. е. наша переменная *\$post* не является объектом, тогда мы возвращаем ответ 404, 'Not found', в противном случае — возвращаем представление *view* и передаем в него три параметра: заголовок, контент и дату публикации новости.

Аналогичным образом строится и метод *GetArchiveContent()*, который должен передать в представление *list* не 10 последних новостей, а список новостей за выбранный период. Для удобной работы с датами мы будем использовать класс *Carbon*, поэтому первым делом подключите его:

```
use Carbon\Carbon;
```

Далее напишем сам метод:

```

public function GetArchiveContent($year, $month) {
    $date = \Carbon\Carbon::parse($year."-".$month."-01");
    $start = $date->startOfMonth()->format('Y-m-d H:i:s');
    $end = $date->endOfMonth()->format('Y-m-d H:i:s');
    $news = DB::table('news')
        ->whereBetween('dt', [$start, $end])
        ->get();

    if (is_object($news)) {
        return view('list',
            ['news' => $news]
        );
    }
}

```

```
    } else abort(404);  
}
```

Для упрощения кода метода мы не производили никакой проверки на ошибки (а не помешало бы — вдруг пользователь передаст число больше 12 в качестве месяца), а передаем в БД сгенерированные две даты: начало и конец запрашиваемого месяца. Затем методом `whereBetween()` мы отфильтровываем новости за сгенерированный период и передаем их в представление `list`.

## 39.4. Обработка POST-запроса

В нашем мини-приложении обработкой POST-запроса занимается метод `AddComment()`. Прежде чем написать код самого метода, нам нужно подключить класс `Request`:

```
use Illuminate\Http\Request;
```

Теперь напишем код самого метода:

```
public function AddComment(Request $request) {  
  
    $uname = $request->input('commenter-name');  
    $comment = $request->input('comment');  
    $postid = $request->input('postid');  
  
    DB::table('comments')->insert([  
        'id' => 0,  
        'uname' => $uname,  
        'comment' => $comment,  
        'postid' => $postid,  
        'dt' => DB::raw('now()')  
    ]);  
  
    return view('thank-you');  
}
```

Первым делом метод получает передаваемые из HTML-формы параметры. Передаются всего три параметра: `commenter-name` (имя комментатора), `comment` (текст комментария) и `postid` (ID новости, к которой добавляется комментарий).

Далее наш скрипт вставляет полученную информацию в таблицу `comments`. Обратите внимание на два момента: поскольку поле `id` помечено как поле со счетчиком при создании таблицы, нам достаточно передать 0, а БД сама правильно установит нужный номер. Второй момент — то, как мы вызываем БД-функцию `now()` для получения текущей даты/времени.

Наконец, метод выводит представление `thank-you`, содержащее текст вроде этого: «Благодарим за ваш комментарий. Он будет опубликован сразу же после одобрения модератором».

\* \* \*

Мы рассмотрели здесь основы «контроллеростроения», а в следующей главе поговорим о представлениях.



## ГЛАВА 40

# Создаем представление

### 40.1. Каталог resources/views

*Представление* (view) взаимодействует с пользователем. Его можно сравнить с шаблоном. Контроллер заполняет шаблон, передает данные в представление, а представление, в свою очередь, оформляет полученные данные в HTML-код, который будет выведен фреймворком в браузер.

Контроллер возвращает представление так:

```
return view('list',  
            ['news' => $news]  
            );
```

или так:

```
return view('contacts');
```

В первом случае контроллер возвращает представление `list` и передает ему данные в качестве второго параметра метода `view()`. Во втором случае возвращается представление `contacts` и при этом никаких параметров не передается.

Фреймворк ищет файл с переданным ему названием представления в каталоге `resources/views`. Расширение у имени файла может быть одно из следующих:

- ❑ `php` — обычный PHP-файл. То есть не шаблон Blade, а самый обычный что ни на есть файл, который может принять от контроллера данные и как-то обработать их. По сути, внутри этого файла может быть любой код. Однако представления на чистом PHP-коде используются очень редко. Гораздо чаще — шаблоны Blade;
- ❑ `blade.php` — Blade-шаблон. При желании или необходимости внутри такого шаблона вы также можете использовать PHP-код, поэтому такой формат представления наиболее универсальный. Синтаксис Blade был рассмотрен в *главе 23*, поэтому здесь мы не будем повторяться.

В сложных проектах вы можете помещать шаблоны, относящиеся к разным модулям, в разные подкаталоги каталога `resources/views`. Как минимум можно вынести

шаблоны для фронта и админки в разные подкаталоги, чтобы вам было удобно и чтобы не плодить длинные и неудобные имена файлов шаблонов.

Например, шаблоны для фронта можно поместить в каталог `resources/views/frontend`. Чтобы обратиться к шаблонам из этого подкаталога, нужно указать его в методе `view`:

```
return view('frontend/contacts');
```

## 40.2. Получение данных из контроллера

Посмотрим на следующий код:

```
return view('list',
           ['news' => $news]
        );
```

Он передает в представление `list` переменную `news`. Сейчас имена переменных в представлении и контроллере совпадают, но это не обязательно. С тем же успехом вы могли бы написать так:

```
return view('list',
           ['news' => $obj]
        );
```

И все равно в представление данные были бы переданы под именем `news`. Внутри кода шаблона вы можете определить переменную `$news` как обычную переменную PHP и работать с ней так, как предусмотрено исходной переменной, которая была передана в представление. Если переменная `$obj` была объектом, то и `$news` будет объектом, если — строкой, то переменная `$news` в шаблоне тоже будет строкой, и т. д.

Вывести переменную (если это скалярная переменная — т. е. строка или число) можно так:

```
{{ $переменная }}
```

Если переменная является массивом или объектом, то для ее вывода можно использовать цикл `foreach`, — например:

```
@foreach ($news as $item)
<div class="news-list">
    <div class="news-title">{{ $item->title }}</div>
    <div class="news-image"></div>
    <div class="news-date">{{ $item->dt }}</div>
    <div class="readmore"><a href="/news/{{ $item->id }}">Читать</a></div>
</div>
```

## 40.3. Очистка кеша страниц и кеша представлений

Laravel в процессе своей работы создает как кеш страниц, так и кеш представлений. Если вы внесли изменения в представление и не видите результат, введите команду:

```
php artisan cache:clear
```

Если и после этого изменений не видно, очистите кеш представлений:

```
php artisan view:clear
```

Эти команды нужно вводить из основного каталога приложения — каталога, в который вы установили Laravel.

Если же вы все равно не видите изменений, убедитесь, что вы редактируете должное представление и обновляете нужную страницу:

1. Посмотрите маршруты в файле `web.php`, чтобы понять, какой метод и какого контроллера отвечает за обработку той страницы, которую вы обновляете в браузере.
2. Откройте файл контроллера и посмотрите, как называется возвращаемое представление. В большинстве случаев проблема будет решена.

Также не забудьте обновить кеш браузера — попробуйте зайти на страничку в режиме инкогнито.



## ГЛАВА 41

# Запрос и ответ

### 41.1. Работаем с запросами. Класс *Request*

При программировании на PHP без фреймворка для получения информации о запросе вам нужно обращаться к массивам `$_SERVER`, `$_GET`, `$_POST`, `$_REQUEST`. Laravel предлагает класс `Request`, предоставляющий множество удобных методов получения такой информации.

Для использования класса `Request` в контроллере нужно объявить это с помощью `use`:

```
use Illuminate\Http\Request;
```

Далее, метод, использующий этот класс, должен принимать объект этого класса:

```
public function Action(Request $request) {
```

Получить доступ к параметру, отправленному методами `GET` или `POST`, можно с помощью метода `input()`:

```
$login = $request->input('login');
```

Класс `Request`, кроме метода `input()`, предоставляет множество других полезных методов. Методов действительно настолько много, что лучше всего разбить их на группы, — так будет удобнее.

#### 41.1.1. Основные методы

Основные методы класса `Request` позволяют получать вводимые пользователем данные. Посредством этих методов можно получить данные, переданные HTTP-методами `GET` и `POST` как в обычном формате (он называется *form data*), так и в формате `JSON`. Другими словами, вам не нужно устраивать парсинг `JSON`-данных в своем приложении — достаточно использовать методы класса `Request`, который понимает, что «на вход» подаются данные в формате `JSON`, и уже сам выполняет все необходимые данные по их разбору. Таблица 41.1 содержит основные методы класса `Request`.

Таблица 41.1. Основные методы класса Request

Метод	Описание
<code>all()</code>	Возвращает массив из всех введенных пользователем данных. Заниматься разборкой массива придется вручную
<code>input(имя_поля)</code>	Позволяет получить значение одного поля ввода
<code>only(имя_поля массив_из_имен_полей)</code>	Может использоваться как для получения значения одного поля ввода, так и для получения списка значений полей ввода, заданных в виде массива
<code>except(имя_поля массив_из_имен_полей)</code>	Возвращает массив из всех данных, за исключением указанных полей ввода
<code>exists(имя_поля)</code>	Проверяет, существует ли указанное имя поля (передано ли). Возвращает <code>true</code> , если это так, или <code>false</code> — в противном случае
<code>has(имя_поля)</code>	Используется для проверки существования, но, в отличие от <code>exists()</code> , вернет <code>false</code> , если поле существует, но равно <code>null</code>
<code>filled(имя_поля)</code>	Возвращает <code>true</code> , если указанное поле существует и содержит значение (заполнено)
<code>json(имя_ключа)</code>	Извлекает указанное значение из JSON-сообщения, которое было отправлено странице

### 41.1.2. Получение информации о пользователе/запросе

Эту информацию, как правило, мы получали из массива `$_SERVER`. Теперь нам нужно использовать методы из табл. 41.2.

Таблица 41.2. Методы получения информации о пользователе/запросе

Метод	Описание
<code>method()</code>	Позволяет получить название метода, которым был передан запрос (GET, POST, PATCH и т. д.)
<code>ip()</code>	Возвращает IP-адрес пользователя
<code>header([заголовок])</code>	Возвращает массив заголовков или только указанный заголовок, если он был передан в качестве параметра
<code>server([параметр])</code>	Возвращает массив переменных, хранящихся в массиве <code>\$_SERVER</code> , или только одно значение, если оно было передано в качестве параметра
<code>secure()</code>	<code>True</code> , если используется протокол HTTPS
<code>path()</code>	Позволяет получить путь (без домена) к странице. Например, если URL был <code>http://localhost/pages/contacts</code> , то этот метод вернет <b>pages/contacts</b>
<code>url()</code>	Возвращает полный URL страницы — например: <b>http://localhost/pages/contacts</b>

Таблица 41.2 (окончание)

Метод	Описание
is()	True, если есть частичное совпадение текущего маршрута с переданной в качестве параметра строкой
accepts()	True, если этот запрос страницы принимает содержимое указанного типа
isJson()	Возвращает true, если входные данные поступают в формате JSON

41.1.3. Работа с файлами

Методы для работы с файлами (для загрузки файла на сервер) следующие:

- ☐ file() — возвращает массив из всех загруженных на сервер файлов или только один файл, если его имя поля было задано в качестве строкового параметра;
- ☐ allFiles() — возвращает массив из всех загруженных на сервер файлов. Является псевдонимом для file() без параметров;
- ☐ hasFile() — возвращает true, если файл с соответствующим ключом (задается в виде строкового параметра).

Каждый загруженный на сервер файл является экземпляром класса `Symfony\Component\HttpFoundation\File\UploadedFile` (это не опечатка — Laravel создавался на базе Symfony в свое время), предлагающим ряд инструментов для обработки и сохранения загружаемых файлов.

Laravel также еще предоставляет методы для работы с сессиями и Cookies, но они будут рассмотрены в следующей главе.

41.1.4. Собирая все вместе...

Еще раз поговорим о том, как создать форму и передать параметр в Laravel-приложение. Допустим, у нас есть такая форма в вашем представлении:

```
<form method="post" action="/actions/form">
@csrf
<input type="text" name="test">
<input type="submit" values="Go!">
</form>
```

Для ее обработки нужно создать маршрут в файле `web.php`:

```
Route::post('/action/form', 'App\Http\Controllers\MainController@Form');
```

Метод Callback объявляется так:

```
use Illuminate\Http\Request;
...
public function Form(Request $request) {
    $uname = $request->input('test');
    // что-то делаем с полученными параметрами
}
```

## 41.2. Класс *Response*

Класс `Response` используется, чтобы отправить клиенту ответ — вместе со всеми заголовками, Cookies, данными и всем, что может понадобиться браузеру для правильного отображения контента.

Первым делом нужно подключить этот класс к своему контроллеру:

```
use Illuminate\Http\Response;
```

При желании `Response` можно использовать даже в файле `web.php` — на случай, когда контроллер не нужен:

```
Route::get('test', function() {  
    return new Illuminate\Http\Response('Test');  
})
```

Самый простой ответ выглядит так:

```
return response('OK');
```

Если вместе с представлением нужно отправить заголовки, HTTP-статус, то используйте метод `view()` объекта `Response`:

```
return response()  
->view('some-view', $someData)  
->header('Content-type', 'text/xml');
```

Метод `download()` инициирует загрузку файлов с сервера на локальный компьютер пользователя. Выглядит все это так:

```
return response->download('file.pdf');
```

Здесь мы просто отправляем в браузер пользователя файл `file.pdf`. Если нам нужно отправить файл под определенным именем, тогда на помощь приходит второй параметр этого метода:

```
return response->download('file.pdf', 'car.pdf', $headers);
```

Необязательный третий параметр (массив пар «ключ = значение») позволяет задать некоторые заголовки вроде `Content-type`.

Если после «отдачи» файла пользователю его нужно удалить, используйте метод `deleteFileAfterSend()`:

```
return response->download('file.pdf', 'car.pdf')->deleteFileAfterSend();
```

Для отправки ответа в формате JSON служит одноименный метод:

```
return response()->json(['name', 'admin']);
```

Как правило, в качестве параметров нужно передать массив пар «ключ = значение». Ответы JSON будут преобразованы в формат JSON (с помощью `json_encode()`), а типу содержимого будет автоматически присвоен тип `application/json`.

Ответы перенаправления также можно организовать хелпером `redirect()`:

```
return redirect('page/about');  
return redirect()->to('page/about');
```

```
return redirect()->route('account.changepswd');  
return redirect()->action('MainController@mainPage');  
return redirect()->away('https://example.com');
```

Первые два перенаправления аналогичны: происходит перенаправление на определенную страницу нашего приложения, — в отличие от последнего, где происходит перенаправление на стороннюю страницу (внешний домен).

Метод `route()` используется для редиректа на указанный маршрут, `action()` — на указанное действие (в качестве параметра передается имя контроллера и метода).

Разработчик может инициировать перенаправление и сохранить в сессии какие-то данные — например:

```
return redirect('/page/edit')->with('error', 'something wrong');
```

В обработчике маршрута `/page/edit` можно получить доступ к сессии так:

```
echo session('error');
```



## ГЛАВА 42

# Работа с данными

### 42.1. Файловая система

Данные мы можем хранить в базе данных, в файловой системе, сессии и в Cookies. О том, как работать с базой данных в Laravel, было рассказано в *главе 21*, а сейчас мы рассмотрим работу с файловой системой, сессиями и Cookies.

#### 42.1.1. Конфигурация фасада *Storage*

Доступ к файловой системе осуществляется через фасад *Storage*. С его помощью вы можете «из коробки» работать с локальной файловой системой, FTP, sFTP, а также с облачным хранилищем S3. Конфигурация этого фасада находится в файле `config/file-system.php`. По умолчанию используется локальная файловая система:

```
'default' => env('FILESYSTEM_DRIVER', 'local'),
```

Далее следует код, позволяющий настроить диски *Storage*:

```
'disks' => [

    'local' => [
        'driver' => 'local',
        'root' => storage_path('app'),
    ],

    'public' => [
        'driver' => 'local',
        'root' => storage_path('app/public'),
        'url' => env('APP_URL').'/storage',
        'visibility' => 'public',
    ],

    's3' => [
        'driver' => 's3',
        'key' => env('AWS_ACCESS_KEY_ID'),
```

```
'secret' => env('AWS_SECRET_ACCESS_KEY'),
'region' => env('AWS_DEFAULT_REGION'),
'bucket' => env('AWS_BUCKET'),
'url' => env('AWS_URL'),
'endpoint' => env('AWS_ENDPOINT'),
],
],
```

Как мы видим, есть возможность настроить облачный диск s3, указав URL, бакет, конечную точку, а также ключи доступа. Также настраиваются и два обычных диска: `local` (локальная файловая система) и `public` (файлы для публичного доступа). По умолчанию файлы для публичного доступа находятся в каталоге `app/public`. Любой файл, который вы поместите в этот каталог, будет доступен по протоколу HTTP/HTTPS.

Диск `local` обеспечивает доступ к локальной файловой системе и подразумевает взаимодействие с каталогом пути хранения `app`, т. е. с каталогом `storage/app`.

42.1.2. Методы фасада Storage

Прежде чем рассмотреть методы фасада `Storage`, нужно поговорить о главном методе — `disk()`, позволяющем выбрать диск, с которым производится работа:

```
Storage::disk('local')->get('image.jpg');
```

Здесь мы хотим получить из локального хранилища файл `image.jpg`. Метод `disk()` нужно использовать, если диск, с которым производится работа, отличается от диска по умолчанию. Для диска по умолчанию этот метод можно не задействовать:

```
Storage::get('image.jpg');
```

Таблица 42.1 содержит методы, которые вы можете использовать с любым диском.

Таблица 42.1. Методы фасада Storage

Метод	Описание
<code>get(имя_файла)</code>	Позволяет получить файл с заданным именем. Метод возвращает содержимое файла в виде строки. Аналог функции <code>file_get_contents()</code>
<code>put(имя_файла, \$contentOrStream)</code>	Позволяет поместить содержимое, заданное вторым параметром, в файл, заданный первым параметром
<code>putFile(каталог, \$file)</code>	Позволяет загрузить файл в каталог. Метод <code>putFileAs()</code> позволяет при этом задать имя файла в каталоге, которое задается третьим параметром
<code>exists(имя_файла)</code>	Проверяет существование файла. Возвращает как обычно <code>true</code> , если файл существует, и <code>false</code> — в противном случае
<code>getVisibility(путь)</code>	Получает статус видимости для указанного пути (путь задается строкой): <code>public</code> (публичный) или <code>private</code> (приватный)

Таблица 42.1 (окончание)

Метод	Описание
setVisibility(путь)	Устанавливает статус видимости для указанного пути: <code>public</code> (публичный) или <code>private</code> (приватный)
copy(файл, новое_имя)	Копирует файл, оба параметра — строковые
move(файл, новое_имя)	Перемещает файл, оба параметра — строковые
prepend(файл, текст)	Добавляет текст в начало файла, файл задается в виде строки
append(файл, текст)	Добавляет текст в конец файла
delete(файл)	Удаляет файл, указанный в виде строки
size(файл)	Возвращает размер указанного в виде строки файла (в байтах)
lastModified(файл)	Возвращает временную метку UNIX ( <code>timestamp</code> ) для времени последнего изменения файла
files(каталог)	Возвращает массив имен файлов, находящихся в указанном в виде строки каталоге
allFiles(каталог)	Возвращает массив имен файлов, находящихся в указанном в виде строки каталоге и во всех его подкаталогах
directories(каталог)	Возвращает массив имен каталогов, находящихся в указанном в виде строки каталоге
allDirectories(каталог)	Возвращает массив имен каталогов, находящихся в указанном в виде строки каталоге и во всех его подкаталогах
makeDirectory(каталог)	Создает новый каталог
deleteDirectory(каталог)	Удаляет пустой каталог

### 42.1.3. Загрузка файлов на сервер

В отличие от сессий и Cookies, которые вы не можете при использовании фреймворка задействовать напрямую (работать не будут), в случае с обычной файловой системой никто вам не запрещает использовать привычные PHP-функции. Но фасад Storage делает загрузку более простой и понятной. Рассмотрим код для загрузки файла:

```
class UserController
{
    public function uploadUserPic(Request $request, User $u)
    {
        Storage::put("profile-pics/{ $u->id }.jpg",
            file_get_contents($request->file('userpic')->getRealPath())
        );
    }
}
```

Метод `put()` помещает содержимое файла, которое мы получаем с помощью функции `file_get_contents()`, в файл `profile-pics/{ $u->id }.jpg`. Загруженный на сервер файл

является потомком класса `SplFileInfo`, поэтому мы можем вызвать метод `getRealPath()`, чтобы получить настоящий путь к файлу на сервере. Функция `file_get_contents()` сначала получает содержимое файла, а затем мы это содержимое методом `put()` помещаем в файл, имя которого соответствует идентификатору пользователя. Например, если у пользователя идентификатор 1, то загруженная на сервер картинка будет помещена в файл `profile-pics/1.jpg`.

#### 42.1.4. Метод *download()*

Метод `download()` служит для отправки файла в браузер пользователя. Например, так вы можете отправить файл с электронной книгой после оплаты за нее или прайс-лист после регистрации пользователя на сайте. Метод `download()` используется, когда нужно отправить пользователю файл, но при этом не предоставлять прямую ссылку на него. В случае с прайс-листом — это выход в ситуациях, когда нужно «отдавать» пользователям самые актуальные цены. То есть сначала вы можете сгенерировать прайс-лист по ценам в базе данных, а затем отправить пользователю уже сгенерированный файл методом `download()`:

```
public function downloadPrice()
{
    return Storage::download('price.pdf');
}
```

### 42.2. Сессии

Есть несколько способов доступа к сессии, но наиболее правильный и он же наиболее распространенный — использование хелпера `session()`:

```
session()->put('user', 'admin');           // помещаем данные в сессию
$user = session()->get('user');             // читаем данные из сессии
```

Казалось, что может быть проще? Однако при использовании сессий в Laravel есть несколько подводных камней, с которыми вы обязательно столкнетесь.

Вот первый из них. Зачем используются сессии? Как правило, для хранения информации и ее передачи между страницами. Например, вы помещаете в сессию корзину пользователя. У вас есть страница каталога `/catalog`, и вы добавили в корзину несколько элементов. Но когда вы перейдете на страницу `/cart` или `/checkout`, вы увидите, что добавленные данные недоступны. И лишь когда вы вернетесь на страницу `/catalog` — добавленные в корзину данные снова станут доступными. То есть по умолчанию данные в сессии сохраняются только для той страницы, на которой они были добавлены.

Для нормальной работы с сессиями нужно отключить `Middleware\AuthenticateSession`. Для этого откройте файл `app/Http/Kernel.php` и прокомментируйте строку, как показано далее:

```
protected $middlewareGroups = [
    'web' => [
```

```
\App\Http\Middleware\EncryptCookies::class,  
\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,  
\Illuminate\Session\Middleware\StartSession::class,  
// \Illuminate\Session\Middleware\AuthenticateSession::class,  
\Illuminate\View\Middleware\ShareErrorsFromSession::class,  
\App\Http\Middleware\VerifyCsrfToken::class,  
\Illuminate\Routing\Middleware\SubstituteBindings::class,  
],
```

Второй подводный камень на пути нормальной работы сессии — это необходимость иметь доменное имя. Если у вас нет доменного имени (когда вы обращаетесь по IP-адресу к вашему приложению), сессии работать не будут.

Кроме того, в файл `web.php` нужно добавить строку:

```
use Illuminate\Http\Request;
```

Без этого сессии также не будут работать между разными контроллерами.

Таблица 42.2 содержит методы для работы с сессиями. В ней приведены не все, а только самые полезные методы, которые вы будете использовать на практике.

Таблица 42.2. Методы для работы с сессиями

Метод	Описание
<code>get(ключ, резервное_значение)</code>	Извлекает из сессии значение указанного ключа. Если ему не присвоено значение, метод возвращает резервное значение, заданное вторым параметром. Если второй параметр не указан, то метод возвращает <code>null</code>
<code>put(ключ, значение)</code>	Сохраняет указанное значение в сессии по указанному ключу
<code>push(ключ, значение)</code>	Если какое-то значение сессии представляет собой массив, то этот метод используется для добавления нового значения в такой массив: <code>session()-&gt;put('letters', ['a', 'b']);</code> <code>session()-&gt;push('letter', 'c');</code>
<code>has(ключ)</code>	Возвращает <code>true</code> , если в сессии есть значение с указанным ключом. Если указанное значение существует и равно <code>null</code> , метод вернет <code>false</code> ! Пример: <code>if (session()-&gt;has('username')) {     // выполняем действия }</code>
<code>exists(ключ)</code>	В отличие от <code>has()</code> , этот метод вернет <code>true</code> , даже если значение запрашиваемого ключа равно <code>null</code> . Метод рекомендуется использовать для проверки существования ключа в сессии
<code>all()</code>	Возвращает массив со всеми значениями сессии, в том числе и со служебными. Может использоваться для отладки
<code>forget(ключ)</code>	Удаляет значение, заданное ключом

Таблица 42.2 (окончание)

Метод	Описание
<code>flush()</code>	Удаляет все значения сессии, включая заданные фреймворком (служебные)
<code>pull(ключ, резервное_значение)</code>	Работает как и метод <code>get()</code> , но после того, как <code>pull()</code> вернет значение ключа, он удалит это значение из сессии
<code>regenerate()</code>	Генерирует новый идентификатор сессии

### 42.3. Работа с Cookies

Еще один способ передавать данные при переходе от одной страницы к другой — Cookies. В отличие от сессий, Cookies хранятся на стороне клиента — в браузере. Для работы с Cookies также предусмотрен специальный фасад и хелпер, позволяющий задавать и сохранять значения.

Помните, что Cookies постоянно привязаны к запросам (в силу их природы) и ответам, поэтому работать с ними придется немного иначе. Файлы Cookies могут находиться в трех местах:

- ❑ поступать вместе с запросом — в момент перехода на страницу пользователь уже имеет файл Cookie, который можно считать с помощью фасада `Cookie` или в объекте запроса;
- ❑ поступать вместе с ответом — такой ответ дает указание браузеру сохранить Cookie-файл для последующих посещений страницы;
- ❑ находиться в очереди — при установке Cookie с помощью фасада `Cookie` он помещается в очередь `CookieJar`, а затем удаляется оттуда и добавляется в объект ответа с помощью `middleware AddQueuedCookiesToResponse`.

Глобальный хелпер `cookie()` используется для создания Cookie-файла. Когда вы передаете параметры функции `cookie()`, они отправляются методу `Cookie::make()`, что делает это самым быстрым способом создания Cookie:

```
cookie('user-id', 101, 150);
```

Можно использовать напрямую метод `make()`, которому вы можете передать следующие параметры:

- ❑ `$name` — имя Cookie-файла;
- ❑ `$value` — значение;
- ❑ `$minutes` — срок хранения;
- ❑ `$path` — путь, для которого файл будет действителен;
- ❑ `$domain` — список доменов, для которых должен использоваться Cookie-файл;
- ❑ `$secure` — указывает, должен ли Cookie-файл передаваться только через защищенное соединение HTTPS;

- `$httpOnly` — указывает, что файл будет доступен только по протоколу HTTP;
- `$raw` — Cookie-файл отправляет без кодирования URL-адреса;
- `$sameSite` — указывает, следует ли предоставлять доступ к Cookie-файлу для межсайтовых запросов (`lax`, `strict` или `null` — возможные варианты).

Пример чтения Cookie-файла из Request выглядит так:

```
Route::get('/catalog', function (Illuminate\Http\Request $request) {  
    $cart = $request->cookie('cart', array());  
});
```

Метод `cookie()` принимает два параметра: имя Cookie-файла и резервное значение (опционально).

Вот пример установки Cookie-файла:

```
Route::post('/cart/add', function() {  
    $cart = cookie('cart', Cookie::getCart());  
    return Response::view('cart')->cookie($cart);  
});
```



## **II. РАЗДЕЛ 9**

### **Безопасность сайта**

<b>Глава 43.</b>	Как взламываются сайты и как этому помешать? Основные сведения
<b>Глава 44.</b>	SSL-сертификат для сайта
<b>Глава 45.</b>	Защита PHP с помощью конфигурационного файла



## ГЛАВА 43

# Как взламываются сайты и как этому помешать? Основные сведения

### 43.1. Основные способы взлома сайта

Прежде чем начинать читать эту главу, вы должны знать две истины: первая — взломать можно абсолютно любой сайт. Вторая — теме безопасности можно посвятить отдельную книгу, и в одной главе мы не сможем рассмотреть все нюансы безопасности сайта или веб-приложения — слишком много переменных.

Как именно взломают ваш сайт, зависит прежде всего от сайта, а также от сервера, на котором он размещен. Иногда бывает проще взломать сервер, чем сам сайт. Если у вас VDS (не хостинг, а именно VDS), то начинать защиту своего сайта нужно с правильной настройки VDS, поскольку у вас может быть прекрасное и защищенное веб-приложение, но нет ничего хуже, чем VDS со стандартными настройками. Многие VDS поставляются даже с включенной учетной записью root и возможностью удаленного входа как root.

Основные рекомендации по защите VDS выглядят так:

- ☐ изменение пароля от учетной записи root — чтобы даже ваш облачный провайдер не смог залогиниться под root (как правило, панель управления VDS хранит пароль root, поэтому изменять его нужно по SSH командой `passwd root`);
- ☐ создать обычного пользователя и предоставить ему возможность использовать команду `sudo`;
- ☐ запретить удаленный логин (по SSH) как root;
- ☐ отключить службу FTP, если она включена;
- ☐ сменить порт SSH с 22 на любой другой — желательно использовать номера больше 1000;
- ☐ настроить брандмауэр;
- ☐ установить SSL-сертификат для веб-сервера;

- для еще большей безопасности настроить SSH на аутентификацию по ключу, а не по паролю и использовать этот способ входа, а аутентификацию по паролю — отключить.

Это минимальный список того, что нужно сделать с вашим VDS сразу после покупки. Однако все указанные действия относятся к администрированию сервера, а не к программированию, поэтому в этой книге о них мы говорить не будем, но имейте этот список в виду: если сами не сможете настроить сервер, обратитесь к специалистам.

Когда злоумышленник поймет, что сервер — неприступная стена, и ему там ловить нечего, он приступит к взлому самого веб-приложения — системы управления сайтом (CMS). Наверняка на вашем сайте есть или будет такая. Скорее всего, у вас там есть две части: «витрина» — т. е. контент, предназначенный для посетителей, и панель управления, где вы управляете этим самым контентом. В большинстве случаев злоумышленник для подбора вашего пароля будет использовать технику, называемую *брутфорсом*, — т. е. метод «грубой силы». Другой, похожий метод — *подбор пароля по словарю*. Но в любом случае техника схожая: у него есть логин для входа, есть имя сценария, который обрабатывает вход пользователя (это общедоступная информация, и имя этого сценария можно посмотреть в HTML-коде страницы входа), и все, что ему остается, — подобрать пароль. Он может подбирать пароль либо методом перебора символов определенного класса — например, букв или буквы и цифр (тот самый брутфорс), либо использовать заранее подготовленный словарь с наиболее вероятными паролями. Часто правильно составленный словарь гарантирует большую эффективность, нежели метод грубой силы. Уберечься от взлома этими двумя методами (атака по словарю и брутфорсинг) можно так:

- ограничить доступ к панели управления по IP-адресу. Например, это можно сделать через конфигурацию веб-сервера. Даже если у вас динамический IP-адрес — это очень эффективный метод. Пусть ваш IP-адрес — 192.168.1.123. Как правило, меняется только последняя часть IP-адреса, поэтому вы можете разрешить доступ к админке лишь пользователям из подсети 192.168.1.0, и злоумышленник уже сразу будет отброшен еще на этапе обращения к странице входа. IP-адрес легко подделать, но проблема в том, что злоумышленник не знает, с какого адреса вы входите, и ему придется каким-то образом узнать именно ваш IP-адрес. А потом — еще и подобрать ваш пароль, поэтому его задача существенно усложняется;
- сложные пароли длиной более 10 символов и не содержащие в себе словарных слов делают малоперспективной как атаку методом грубой силы, так и атаку по словарю;
- реализация механизма защиты учетной записи, когда после определенного количества неудачных попыток входа учетная запись блокируется на некоторое время. Для многих CMS такие модули уже разработаны, но вы можете разработать свой модуль самостоятельно даже для самописной CMS. В таблице с пользователями предусмотрите поле `active` (или с другим именем), которое будет содержать 0, если учетная запись заблокирована, и 1, если с ней все нормально. Также понадобится поле `errcnt` — счетчик неудачных попыток входа и `errdt` —

поле, содержащее время и дату последней неудачной попытки входа. Теперь при проверке пароля — если он не совпадает, вы увеличиваете `errcnt` на единицу и вносите в `errdt` текущую время и дату. Когда `errcnt` станет больше запрограммированного значения (обычно дается три или максимум пять попыток), значение поля `active` меняется на 0. Если значение этого поля 0, то система вообще не будет производить проверку пароля, а сразу выведет сообщение о том, что учетная запись заблокирована. Кроме того, при каждом входе нужно проверять `errdt` — если с момента последней неудачной попытки прошло более трех часов, `active` можно установить в 1, чтобы реальный пользователь также смог войти в систему. А можно и вообще ничего не делать — пусть администратор сам разблокирует пользователя (вручную через базу данных). Все это довольно просто, и конкретный алгоритм зависит от ваших предпочтений и поставленной задачи.

Следующий наиболее популярный метод взлома — использование *уязвимостей*. Уязвимости — это «встроенные» недостатки программного кода. Нет ничего идеального, и порой программист может написать неидеальный участок кода, который при определенном стечении обстоятельств способен стать причиной несанкционированного доступа к системе. Далее мы рассмотрим два самых распространенных метода взлома, основанных на уязвимостях программного кода.

## 43.2. Два самых распространенных метода взлома

Когда злоумышленник перепробовал ранее упомянутые способы взлома и у него ничего не вышло, ему ничего другого не остается, как начать изучать вашу систему методом «черного ящика» — искать в ней уязвимости, не имея доступа к ее программному коду.

При взломе сайтов часто применяются следующие методы: так называемый *межсайтовый скриптинг* (XSS) и *SQL-инъекции* (SQL injection). Первый метод заключается в том, что злоумышленник внедряет свой HTML- или PHP-код в сценарий страницы сайта. Причем возможности этого практически ничем не ограничены — можно вставить зловредный код даже через якобы безопасный тег `img`, предназначенный для отображения картинки. Как правило, от этого страдают плохо защищенные гостевые книги или форумы, позволяющие использовать HTML-теги.

SQL-инъекции еще более опасны, поскольку внедрение SQL-кода может привести к краже информации и даже к полному уничтожению базы данных. Тут уже и не знаешь, что хуже: потеря или кража информации. Возможен и третий вариант — кража с потерей, т. е. базу данных сначала украли, а потом уничтожили.

Далее мы рассмотрим оба метода достаточно подробно.

### 43.2.1. Межсайтовый скриптинг

Проблема XSS-атак известна уже давно, но тем не менее она до сих пор остается актуальной, поскольку им подвержено очень много сайтов, да и сами эти атаки

нельзя назвать безопасными. Особенность такой атаки заключается в том, что взламывается не сервер хостинг-провайдера (ведь в основном провайдеры уделяют огромное внимание безопасности собственных серверов), а плохо защищенный сценарий клиента. Естественно, далеко не все их авторы являются «специалистами с большой буквы» — вот и получается, что каждый день появляются уязвимые сайты. Дело в том, что разработчик, который создавал сценарий, даже и не подозревал, что кто-то может вот так просто взломать его сайт. Да и отслеживать такие атаки сложно, потому что веб-серверы протоколируют далеко не все запросы (например, POST-запросы вообще редко протоколируются, следовательно, можно передать любой POST-запрос и остаться незамеченным).

Как осуществляется XSS-атака? Злоумышленник конструирует специальный интернет-адрес (URL), который передает своей жертве. Цель? Чаще всего вставка определенного HTML-кода осуществляется для рекламы или перенаправления на другой сайт (который, как правило, находится под контролем злоумышленника). Бывают и сложные XSS-атаки, когда злоумышленник перехватывает сессию, но это довольно редкий тип атаки, поскольку злоумышленнику нужен полный доступ к веб-серверу, на котором запущен сценарий, а имея такой доступ, можно навредить сайту и без всяких XSS-атак.

Рассмотрим небольшой пример уязвимого сценария:

```
<?php  
echo "Привет $name";  
?>
```

Сценарию нужно передать имя (параметр `$name`), которое будет выведено в браузер:

```
http://localhost/index.php?name=Jack
```

Если передать просто имя, тогда мы увидим то, что и ожидали:

**Привет, Jack**

Но что если передать более сложный HTML-код — например:

```
http://localhost/index.php?name=<img+src=http://host.ru/bannder.jpg>
```

Будет выведено слово **Привет** и указанная нами картинка!

В нашем случае ничего страшного не произойдет. Просто злоумышленник порадует, увидев свою картинку. Но ведь иногда информация передается сразу в базу данных, без предварительной проверки! А потом при ее чтении из базы данных будет выведен HTML-код, оставленный «на память» злоумышленником.

Но и это еще не самая большая беда. Ведь злоумышленник таким образом может передать не только HTML-код, но и JavaScript, а иногда даже и PHP-код. Вот небольшой пример передачи JavaScript-кода:

```
http://localhost/index.php? name=<script>alert (document.cookie)</script>
```

Если просмотреть исходный код страницы, то он будет выглядеть так:

```
Привет <script>alert (document.cookie)</script>
```

Как же уберечься от XSS-атак? В PHP есть две функции, которые могут помочь делу:

- `strip_tags()` — удаляет из строки все HTML-теги, кроме разрешенных;
- `htmlspecialchars()` — заменяет все специальные символы на их HTML-эквиваленты (амперсанд заменяется `&amp;`, символ `<` — `&lt;` и т. д.).

Рекомендую сначала удалить все HTML-теги, а затем пропустить полученные извне данные через функцию `htmlspecialchars()` — для надежности:

```
$name = strip_tags($name);  
$name = htmlspecialchars($name);
```

Второй параметр функции `strip_tags()` позволяет указать список разрешенных HTML-тегов, например:

```
$name = strip_tags($name, '<p><b>');
```

Но я настоятельно не рекомендую разрешать какие-либо HTML-теги, даже самые безобидные — вроде `<p>` и `<b>`.

В Laravel все это делать не нужно, если вы используете класс `Request` для обработки поступающих запросов.

### 43.2.2. SQL-инъекции

SQL-инъекции по своей природе очень похожи на XSS-атаки и различаются только деталями. Цель XSS-атаки — выход за пределы HTML-тега, а SQL-инъекции — внедрение SQL-кода, что может привести к краже информации и даже к полному уничтожению базы данных.

Рассмотрим тривиальный пример SQL-инъекции. Предположим, что у нас есть таблица `users`, содержащая информацию о пользователях:

```
id int(11) NOT NULL auto_increment,  
login tinytext,  
password tinytext,  
email tinytext,  
PRIMARY KEY (id_user)
```

Авторизацию пользователя выполняет сценарий `login.php` (листинг 43.1).

#### Листинг 43.1. Уязвимый сценарий

```
<?php  
// В файле connect.php — параметры соединения с сервером MySQL  
// и само подключение к серверу, имя переменной $mysqli  
include "connect.php";  
  
// подключаемся к серверу  
  
// сценарию через форму передаются две переменные: $login и $pass
```

```
// если в таблице users будет запись, поле login которой содержит
// значение, равное значению переменной $login, и поле password
// содержит значение, равное значению переменной $pass, то результат
// этого запроса будет содержать одну строку, в противном случае
// число строк результата будет равно нулю $query = "SELECT * FROM users
                                WHERE login = \"\$login\" AND password = \"\$pass\"";
// выводим для отладки запрос, чтобы вы видели, что происходит
echo $query;

$r = $mysqli->query($query);

if($r->num_rows > 0)
{
    echo "<p>аутентификация прошла успешно";
}
else echo "<p>access denied";

?>
```

### ПРИМЕЧАНИЕ

Код этого сценария вы найдете в файле *43-1.php* каталога *Glava\_43* сопровождающего книгу электронного архива (см. *приложение 4*).

Нашему сценарию нужно передать два параметра: `$login` и `$pass`. Форма для передачи этих параметров может выглядеть так:

```
<form action=sql.php method=get>
Login <input type=text name=login>
Pass <input type=text name=pass>
<input type=submit name=go>
</form>
```

Предположим, что в нашей таблице есть пользователь `admin` с паролем `123`. Если ввести эти имя пользователя и пароль, то мы увидим в окне браузера следующий текст:

```
SELECT * FROM users WHERE login = "admin" AND password = "123"
аутентификация прошла успешно
```

Все как бы работает правильно. Теперь вместо имени пользователя введем `admin"/*`, а пароль вообще не будем указывать. В окне браузера увидим следующее:

```
SELECT * FROM users WHERE login = "admin"/*" AND password = ""
аутентификация прошла успешно
```

Получается интересная ситуация — вместо имени пользователя ввели строку, только отчасти похожую на нужную, пароль вообще не указали, а сценарий пишет, что авторизация прошла успешно.

Оказывается, ничего сверхъестественного не произошло. Вы только взгляните на наш SQL-запрос:

```
SELECT * FROM users WHERE login = "admin"/*" AND password = ""
```

На самом деле он выглядит так:

```
SELECT * FROM users WHERE login = "admin"
```

Символы `/*` считаются началом комментария, поэтому все, что находится после них, просто не учитывается. В результате наш запрос просто запрашивает строку, содержащую информацию о пользователе `admin`. Поскольку у нас есть пользователь с таким именем, то число строк результата будет больше нуля, следовательно, сценарий «радостно» сообщит нам, что:

### аутентификация прошла успешно

Выходит, все, что необходимо знать злоумышленнику, — это имя пользователя, которого нужно взломать.

Что же нам делать? Во-первых, следует изменить алгоритм проверки имени пользователя и пароля. Сначала нужно запросить всю запись, содержащую информацию о пользователе `admin`, а затем сравнить переданный пароль со значением поля `password`. Сказанное иллюстрирует листинг 43.2.

#### Листинг 43.2. Сценарий без уязвимости

```
<?php
include "connect.php";

$query = "SELECT * FROM users WHERE login = \"\$login\"";

echo $query;
$r = $mysqli->query($query);
if($r->num_rows > 0)
{
    $f = $r->fetch_array();
    echo "<p>А теперь проверяем пароль...";

    if ($f['password']===$pass)
        echo "<p>авторизация прошла успешно";
    else
        die("<p>access denied");
}
else echo "<p>access denied";

?>
```

### ПРИМЕЧАНИЕ

Приведенный здесь код вы найдете в файле `43-2.php` каталога `Glava_43` сопровождающего книгу электронного архива (см. приложение 4).

Желательно также обработать передаваемые переменные методом `$mysqli->real_escape_string()` или функциями `addslashes()` и `htmlspecialchars()`. Вторая функ-

ция нам уже встречалась, а первая, если вы не забыли, добавляет слешы перед кавычками:

```
$login = addslashes($login);  
$login = htmlspecialchars($login);
```

В нашем случае вызывать указанные функции для переменной `$pass` нет смысла, поскольку она не участвует в SQL-запросе.

Но даже все сделанное не всегда гарантирует стопроцентную защиту. А что если вместо имени пользователя для авторизации используется ID пользователя, т. е. его числовой идентификатор в системе? Тогда в запросе не будет кавычек, и злоумышленник сможет вообще исполнить любой SQL-код с помощью оператора `UNION`. Все, что ему потребуется ввести, — это строку `1 UNION <SQL-код>` в качестве идентификатора пользователя.

Чтобы не позволить злоумышленнику выполнить любой SQL-код, нужно перед помещением полученных данных (идентификатора пользователя) в SQL-запрос проверить их с помощью функции `is_numeric()`:

```
if (is_numeric($id)) {  
    $query = "select * from users where id = $id";  
    ...  
}
```

Так и только так вы защитите себя от этого вида SQL-инъекции.

Преимущество использования Laravel и других фреймворков PHP в том, что если вы используете предоставляемые средства для работы с БД, — например, конструктор запросов, то вам не нужно изобретать велосипед, и все подобные проверки уже сделаны за вас. Так что это еще один плюс в копилку Laravel.

## 43.3. Остальные методы

Мы рассмотрели далеко не все методы взлома веб-приложения. Например, с успехом могут использоваться фишинг и социальная инженерия. Вы можете получить от злоумышленника письмо якобы от хостера, что нужно сменить пароль, перейдете по ссылке, увидите страницу, как две капли воды похожую на страницу панели управления хостингом (в том же дизайне), укажете старый пароль (он как раз и нужен злоумышленнику), укажете новый пароль и успокоитесь. А злоумышленник тем временем получит доступ к вашему сайту благодаря паролю, который вы сами ему и сообщили.



## ГЛАВА 44

# SSL-сертификат для сайта

### 44.1. Выбор сертификата

Наличие SSL-сертификата на вашем сайте — это не только статус компании и соответствие требованиям современных браузеров. Выбор сертификата имеет самое прямое отношение к безопасности вашей бизнес-инфраструктуры, а поэтому заслуживает надлежащего внимания.

Существуют различные типы SSL-сертификатов. Но какой из них выбрать? Попробуем разобраться...

#### 44.1.1. Основные типы сертификатов

Сертификаты различаются не только брендом и ценой. При покупке сертификата нужно обратить внимание на его тип, определяющий функционал сертификата.

Итак, существуют три основных типа сертификатов:

- ☐ DV (Domain Validation) — используется для подтверждения домена;
- ☐ OV (Organization Validation) — используется для подтверждения организации и домена;
- ☐ EV (Extended Validation) — служит для расширенного подтверждения организации и домена.

Обратите внимание, что сертификаты типов OV и EV также можно использовать для подтверждения домена.

Все указанные сертификаты обеспечивают шифрование трафика между сайтом и браузером. Другими словами, если нужно организовать шифрование трафика по протоколу HTTPS, можно выбрать любой тип сертификата.

У всех этих сертификатов есть дополнительные опции: WildCard и SAN. Первая подтверждает домен и все его поддомены следующего уровня. То есть можно купить сертификат DV с опцией WildCard для домена example.com и использовать его для подтверждения не только example.com, но и всех его поддоменов следующего уровня, т. е. sales.example.com, dev.example.com, docs.example.com и т. д.

Опция SAN подтверждает домены по списку, указанному при получении SSL-сертификата.

### 44.1.2. Какой тип сертификата выбрать?

Ответ на этот вопрос зависит от поставленных задач. Если ставится задача минимум — т. е. обеспечение защиты пользователей своего сайта от появления у них предупреждений браузера о посещении непроверенного сайта, то достаточно обзавестись самым простым (и самым дешевым) SSL-сертификатом типа DV. Этого будет вполне достаточно, чтобы получить зеленый значок рядом с интернет-адресом сайта (URL) в любом современном браузере (рис. 44.1).

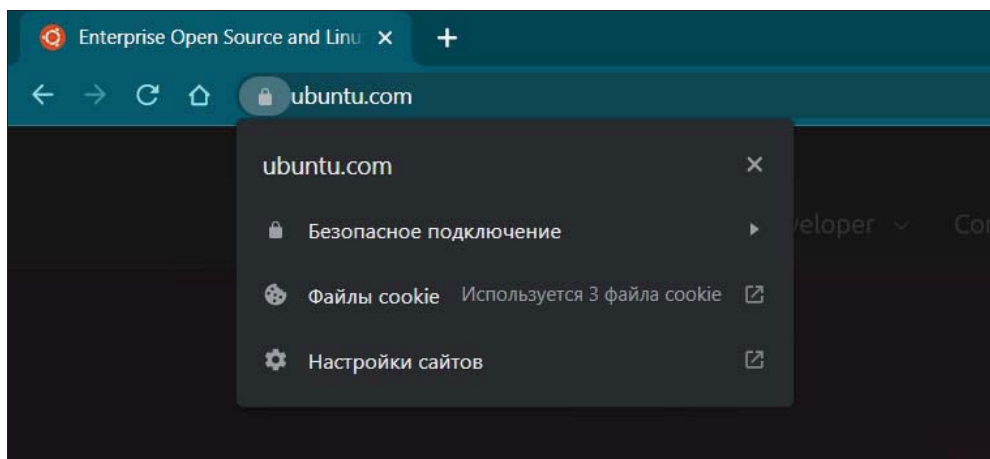


Рис. 44.1. Ubuntu: проверенный сайт

Если у вас есть поддомены вроде sales.example.com, не забудьте заказать опцию WildCard. Если у вашего сайта несколько адресов или у вас несколько сайтов (по отдельному сайту для каждого подразделения компании), то не забудьте указать опцию SAN, чтобы не покупать отдельный SSL-сертификат для каждого домена.

А вот если интернет-площадка используется для операций, требующих повышенного уровня сохранности данных компании и клиентов (например, операций с платежными картами), то лучше выбрать EV-сертификат.

### 44.1.3. Особенности SSL-сертификатов разных типов

Различия между всеми современными типами SSL-сертификатов можно разделить на две группы: по методу проверки и по сертифицируемым доменам. Рассмотрим первую группу различий:

- ❑ *сертификат с проверкой домена (Domain Validation, DV)* — подтверждает, что пользователь находится именно на том сайте, на который он осуществил переход, т. е. такой сертификат удостоверяет сервер, обслуживающий сайт. DV-сертификат не содержит информации о компании-владельце, поэтому не может

считаться безопасным для оказания коммерческих услуг. Представим, что у вас есть интернет-магазин, предоставляющий возможность онлайн-покупки чего-либо. Если вы обрабатываете платеж самостоятельно, то DV-сертификат нельзя считать достаточно безопасным. Однако если при нажатии кнопки **Купить (Оплатить)** или подобной) пользователь уходит на сайт системы, обрабатывающей платеж, — например, на сайт Сбербанка, то DV-сертификата будет вполне достаточно, а EV-сертификат пусть покупает себе Сбербанк. Вам же он не нужен. Все, что вам нужно, — это чтобы слева от адреса вашего сайта в браузере присутствовал зеленый значок замка (или серый, но обязательно закрытый), как показано на рис. 44.1;

- *сертификат с проверкой организации* (Organization Validation, OV) — позволяет подтвердить не только доменное имя, но и организацию-владельца веб-сайта. Подлинность организации проверяется по регистрационным данным юридического лица, которые пересылаются провайдеру SSL-сертификата при заказе OV-сертификата. Такие сертификаты наиболее популярны на сегодняшний день;
- *сертификат с расширенной проверкой* (Extended Validation, EV) — обладает самым высоким уровнем доверия со стороны других узлов. Если вам нужна строгая конфиденциальность передаваемых данных (например, обработка платежей именно на вашем сайте), то вам потребуется EV-сертификат.

EV-сертификаты заслуживают отдельного внимания как самые дорогие. Такие сертификаты подразумевают расширенную периодическую проверку данных владельца сайта для предотвращения подмены этих данных. Так, если OV-сертификат считается действительным на протяжении всего своего срока и проверка организации осуществляется только при покупке сертификата, то в случае с EV-сертификатом такая проверка может осуществляться несколько раз (с определенной периодичностью) на протяжении всего срока действия сертификата.

Изначально все продаваемые SSL-сертификаты были типа OV, т. е. подразумевали проверку данных организации. Однако маркетинг сделал свое дело — и появились DV-сертификаты с упрощенной проверкой, когда проверяется только сам домен. Это сыграло на руку мошенникам — ведь при наличии такого сертификата браузер не будет предупреждать пользователя о возможных проблемах с подлинностью сайта. Отчасти виновником этой проблемы стали браузеры, требующие SSL-сертификат. В погоне за зеленым значком в адресной строке многие стали покупать SSL-сертификаты, а спрос, как говорится, рождает предложение, и провайдеры SSL-сертификатов стали предлагать более дешевые DV-сертификаты.

EV-сертификат создан для решения этой проблемы. При использовании EV-сертификата в браузере появляется так называемая *зеленая панель* (green bar), в которой выводится не просто зеленый значок защиты, а название организации, которой выдан сертификат. На рис. 44.2 показано, как выглядит эта панель в браузере Firefox.

Если щелкнуть на названии организации, можно получить дополнительную информацию о сертификате: кому выдан, кем выдан (рис. 44.3). Если и этого мало, тогда нужно нажать кнопку **Подробнее**, а затем кнопку **Просмотреть сертификат**

(рис. 44.4) — это в браузере Firefox, а в браузере Chrome после щелчка на названии организации следует нажать ссылку **Действительный**. На рис. 44.2 приведена информация о сертификате — в частности, видно, что это расширенный сертификат.

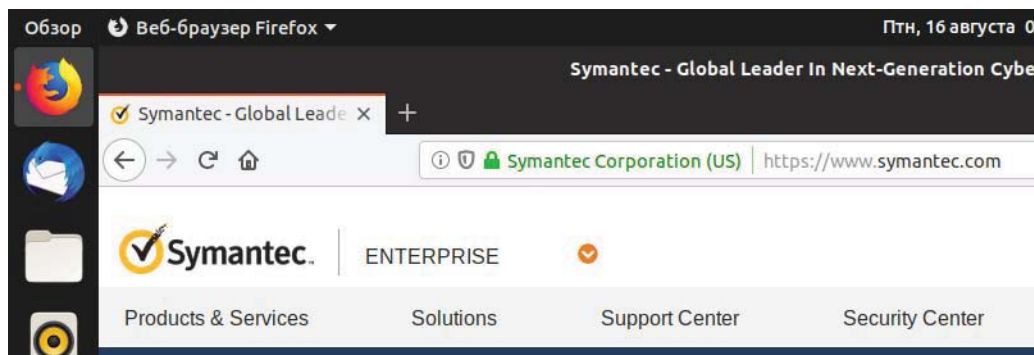


Рис. 44.2. Ubuntu: сведения об организации, которой выдан сертификат

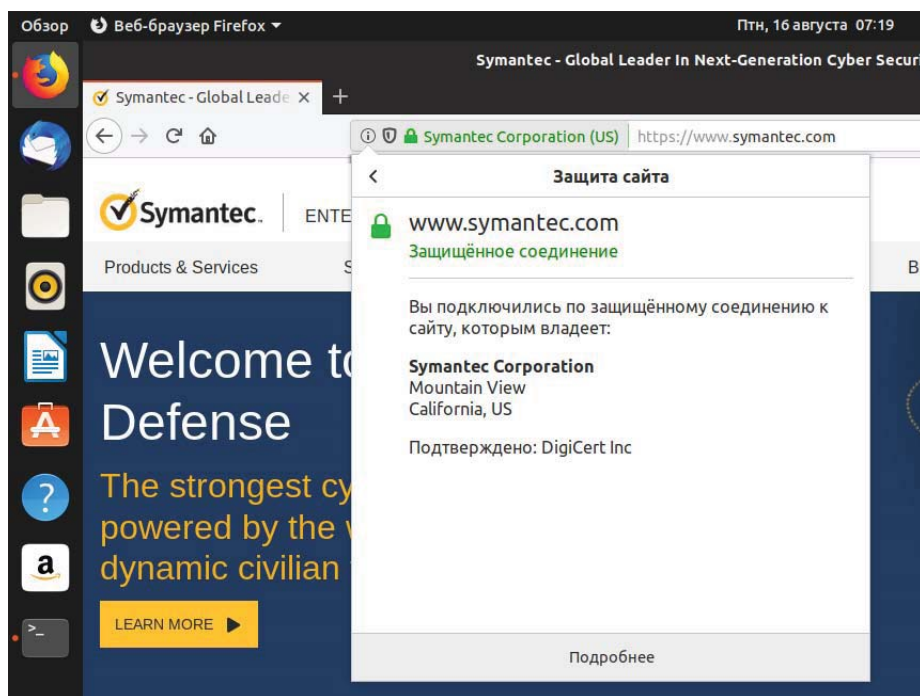


Рис. 44.3. Ubuntu: дополнительные сведения о сертификате

Опция green bar не предоставляется для DV- и OV-сертификатов. Поэтому если вы видите не просто зеленый значок защиты, а название компании, которой принадлежит сайт, то можете быть уверенными, что компания использует EV-сертификат и ей можно доверять. По крайней мере, ваши данные не будут украдены третьей стороной, сама компания не считается...

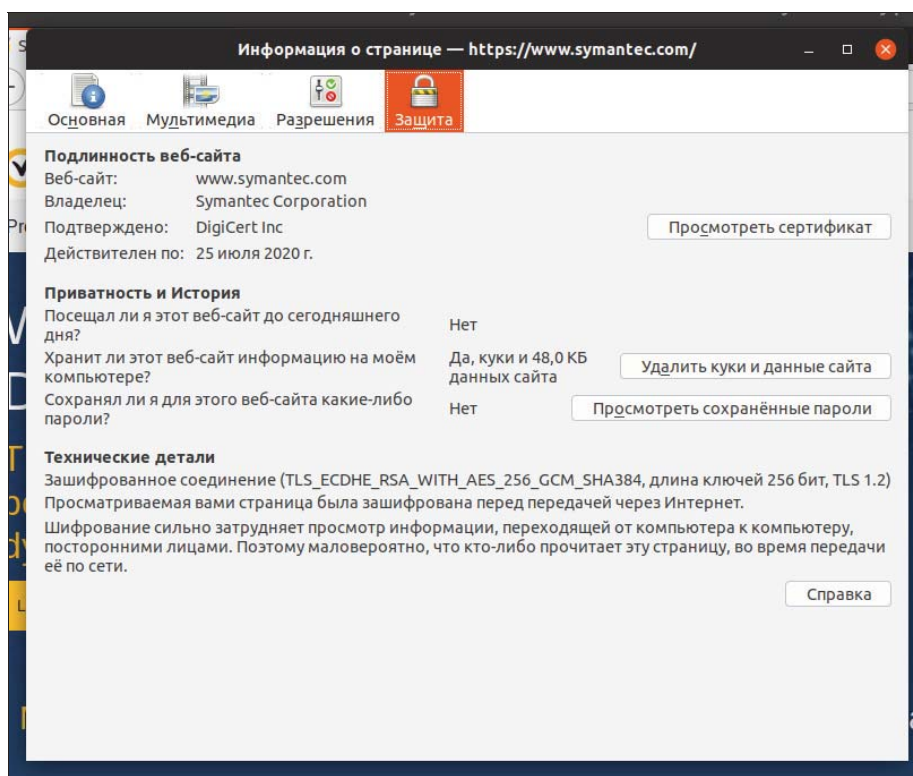


Рис. 44.4. Ubuntu: расширенные сведения о сертификате

Вторая группа различий — по сертифицируемым доменам. Здесь возможны следующие варианты:

- ☐ *сертификат на один домен* (Single Certificate) — работает только для того доменного имени, которое было указано при заказе сертификата. Если вы купили такой сертификат для своего домена `example.com`, то он уже не будет действителен для его поддоменов типа `sales.example.com`;
- ☐ WildCard SSL — используется не только для домена, но и всех его поддоменов, — например: `sales.example.com`, `docs.example.com` и т. д.;
- ☐ SAN (Subject Alternative Name) или UC (Unified Communications) — такие сертификаты могут использоваться сразу для нескольких доменов.

## 44.2. Где купить SSL-сертификат?

Правильнее всего, казалось бы, купить сертификат непосредственно у центра сертификации. Но не все так просто. Многие центры сертификации не работают непосредственно с клиентами, а только с посредниками (реселлерами). Дело в том, что реселлер берет на себя основную работу с клиентом — например, собирает все необходимые документы, а также запрашивает дополнительную информацию, которая может понадобиться в процессе сертификации.

Реселлером, как правило, является ваш хостинг-провайдер. Другими словами, сертификат можно купить у организации, которая предоставляет вам хостинг или виртуальный сервер. Если у вас виртуальный сервер, то сертификат придется «прикручивать» к нему вручную, или же можно обратиться в техническую поддержку — за дополнительную плату вам его установят. Далее будет показано, как самостоятельно установить сертификат на сервере, что сэкономит вам немного средств (см. разд. «Установка сертификата на веб-сервер»).

Сэкономить можно еще больше, если использовать бесплатный сертификат Let's Encrypt. Недостаток этого способа в том, что каждые три месяца нужно будет этот сертификат обновлять. Впрочем, планировщик `crontab` отлично справляется с такой задачей. О том, как установить бесплатный сертификат Let's Encrypt, показано в следующем руководстве: <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-22-04>.

## 44.3. Установка сертификата на веб-сервер

### 44.3.1. Веб-сервер Apache2

Итак, в результате генерирования сертификата или его заказа вам будут предоставлены два файла: SSL-файл сертификата (с расширением `pem`) и ключевой файл (с расширением `key` или `pem` — в случае с Let's Encrypt).

Перейдите в каталог `/etc/apache2/sites-available`. В нем хранятся конфигурационные файлы сайтов, работающих на вашем веб-сервере. Откройте файл, содержащий конфигурацию сайта, для которого вы купили сертификат. Далее представим, что наш сайт называется `example.com`. Конфигурация для него будет следующей:

```
<VirtualHost *:80>
    ServerName example.com
    Redirect permanent / https://example.com /
</VirtualHost>

<VirtualHost *:443>
    ServerAdmin admin@example.com
    ServerName example.com
    ServerAlias www.example.com xxx.xxx.xxx.xxx
    DocumentRoot /var/www/example.com/html
    ErrorLog /var/www/example.com/logs/error_log
    CustomLog /var/www/example.com/logs/access_log combined env=!loopback

    SSLEngine on
    SSLCertificate /etc/путь/fullchain.pem
    SSLCertificateKeyFile /etc/путь/privkey.pem
</VirtualHost>
```

Разберемся, что здесь к чему. Сначала мы создаем виртуальный хост для порта 80, который будет работать как перенаправление — на HTTPS-версию сайта. Можно

было бы сделать это и через файл `.htaccess`, но поскольку у нас есть доступ к конфигурации сервера, то можно сделать это прямо здесь.

Далее мы описываем виртуальный хост для порта 443 (используется SSL). Настройки такие же, как и для обычной версии сайта: `ServerName`, `DocumentRoot` и т. д. Различие заключается только в наличии трех SSL-директив. Первая включает SSL, вторая задает PEM-файл, третья — файл приватного ключа.

Затем нужно перейти к файлу `/etc/apache2/ports.conf` и указать, что серверу нужно слушать порт 443:

```
Listen 80
<IfModule ssl_module>
    Listen 443
</IfModule>
```

После этого нужно сохранить файл конфигурации и перезапустить Apache:

```
sudo systemctl restart apache2.service
```

Обратитесь к вашему сайту. Если вместо надписи «Не защищено» появилось изображение зеленого замка, то все хорошо и настройку можно считать завершенной.

Однако иногда замок отображается не зеленым, а серым и не закрытым, а открытым. Это означает, что не все ресурсы сайта загружаются по протоколу HTTPS. Откройте исходный код страницы и произведите поиск по строке: `http://`. Ваша задача — найти адреса ресурсов (JS, CSS, картинок), которые загружаются по протоколу HTTP. Исправьте интернет-адреса (URL) проблемных ресурсов на `https://` и снова обновите страницу сайта. Если вы все сделаете правильно, то увидите зеленый замок соединения.

### 44.3.2. Веб-сервер Nginx

Веб-сервер Nginx настраивается немного иначе. Откройте файл конфигурации вашего сайта и добавьте в него следующие строки:

```
server {
    listen      443 ssl http2;
    ssl on;
    server_name example.com www.example.com;
    ssl_certificate      /etc/путь/файл.pem;
    ssl_certificate_key  /etc/путь/server.key;
    ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers          HIGH:!aNULL:!MD5;
    ...
}
```

Здесь мы включаем SSL (`ssl on`), задаем порт 443, а также прописываем пути к сертификату. Параметры `ssl_protocols` и `ssl_ciphers` задают протоколы и шифры для сертификата. Их можно не указывать, если этого явно не требует сам сертификат. После изменения конфигурационного файла сервис нужно перезапустить:

```
sudo systemctl restart nginx.service
```



## ГЛАВА 45

# Защита PHP с помощью конфигурационного файла

### 45.1. Конфигурационный файл *php.ini*

Конфигурационный файл `php.ini` содержит все настройки интерпретатора PHP. А здесь мы рассмотрим параметры PHP, относящиеся к безопасности (прочие самые полезные настройки файла `php.ini` приведены в *главе 4*). Но прежде разберемся, где этот файл находится.

В Linux/UNIX файл `php.ini` обычно лежит в каталоге `/etc`, но иногда его можно найти в каталогах `/usr/lib` или `/usr/local/lib`. В Windows файл `php.ini` находится обычно в каталоге `C:\php` (хотя это зависит от способа настройки связки Apache + PHP). Проще всего определить местонахождение этого файла — выполнить функцию `phpinfo()`. В поле `Loaded Configuration File` вы найдете полный путь к конфигурационному файлу. Обычно этот файл находится в каталоге `/etc/php/<версия_PHP>/apache2/php.ini` или `/etc/php/<версия_PHP>/fpm2/php.ini` (при использовании `nginx`)

Редактировать `php.ini` можно в любом текстовом редакторе. Вот пример директив этого файла:

```
max_execution_time = 30      ; Maximum execution time of each script, in seconds
memory_limit = 8M           ; Maximum memory a script may consume
```

Если вас интересует какая-то директива, ее описание на русском языке можно найти на следующей странице: <https://www.php.net/manual/ru/ini.core.php>.

В Linux/UNIX для редактирования файла `php.ini` нужны права пользователя `root`. Другими словами, если у вас хостинг, а не выделенный сервер, вы его редактировать не сможете.

Все директивы конфигурационного файла `php.ini` мы рассматривать не станем. В этом нет смысла, поскольку в большинстве случаев вы не сможете их изменить.

Однако вы можете включить ту или иную директиву для своего сценария с помощью функции `ini_set()`:

```
string ini_set ( string $директива, string $значение )
```

Например:

```
ini_set("max_execution_time", "30");
```

Для установки значения директивы конфигурации служит функция `ini_get()`, например:

```
echo "max_execution_time =" . ini_get('max_execution_time');
```

Одна из очень полезных директив файла конфигурации — `error_reporting`. Она управляет сообщениями об ошибках и предупреждениями, которые будут выводиться интерпретатором. Во время разработки сценария нужно включить вывод всех ошибок и других сообщений интерпретатора. Для этого служит функция:

```
error_reporting(E_ALL);
```

Вызов этой функции должен идти в сценарии самым первым. Когда сценарий отлажен и работает без ошибок, вывод всех ошибок следует выключить:

```
error_reporting(0);
```

Дело в том, что при выводе ошибки PHP показывает информацию, которую не нужно знать злоумышленнику. Если у вас есть доступ к `php.ini`, то вы можете вообще выключить вывод сообщений интерпретатора в браузер, но включить их протоколирование. Для этого служат две следующие директивы:

```
display_errors = Off
log_errors = On
```

После этого все ошибки будут протоколироваться в журнал Apache.

## 45.2. Отключение потенциально опасных функций

Если вы администратор веб-сервера, отключите с помощью директивы `disable_functions` потенциально опасные функции (после этого они не будут доступны в сценариях). Вот список этих функций:

```
disable_functions = system, exec, passthru, shell_exec, proc_open
```

## 45.3. Рекомендованные значения некоторых конфигурационных директив

В табл. 45.1 приведены рекомендованные (с точки зрения безопасности) значения некоторых директив файла конфигурации `php.ini`.

Таблица 45.1. Рекомендованные значения директив

Директива и значение	Комментарий
<code>safe_mode = off</code>	Безопасный режим на самом деле ничего не помогает обезопасить, поэтому эту директиву нужно выключить
<code>error_reporting = off</code>	Не выводить сообщения об ошибках. Чем меньше злоумышленник знает о нашей системе, тем лучше

Таблица 45.1 (окончание)

Директива и значение	Комментарий
log_errors = on	Включить протоколирование ошибок в файл журнала
open_basedir = /tmp, ...	Здесь нужно указать два каталога: корневой каталог веб-сервера и каталог /tmp. Не забудьте его указать, иначе сессии работать не будут
expose_php = off	Не добавлять подпись PHP к заголовкам Apache (чтобы никто не узнал вашу версию PHP)
allow_url_open = off	Запрещает обращаться к удаленным файлам по протоколу HTTP



## **II. РАЗДЕЛ 10**

### **Полезные сведения**

<b>Глава 46.</b>	Устанавливаем визуальный редактор Summernote
<b>Глава 47.</b>	Работа с MongoDB средствами PHP



## ГЛАВА 46

# Устанавливаем визуальный редактор Summernote

### 46.1. Знакомство с редактором

Разработка собственного визуального (WYSIWYG) редактора — дело достаточно хлопотное даже для среднего размера команд. Как правило, берется и используется в проекте один из уже готовых визуальных редакторов.

Существуют различные варианты готовых редакторов: Summernote, TinyMCE, Editor.js, SPAW2, Toast Editor, Dante Editor, tiptap и др. Выбор редактора зависит от личных предпочтений разработчика и технического задания к проекту.

Как правило, в техническом задании о редакторе упоминается мало, и поэтому многие разработчики стараются установить отлично кастомизируемый редактор, ковым является Summernote.

Summernote легко интегрируется с популярными фреймворками в духе React, Angular, Vue.js, Bootstrap и jQuery, буквально встраиваясь в их интерфейс в виде нативного программного обеспечения.

Множество плагинов позволяют существенно расширить функциональность редактора, которая и без того лучше, чем у некоторых его аналогов. Далее мы покажем, как интегрировать редактор с Laravel-проектом. Получить редактор можно на сайте <https://summernote.org/>.

Прежде чем продолжить, нужно отметить, что редактор является OpenSource, поэтому вы без проблем можете использовать его в своих проектах абсолютно бесплатно.

### 46.2. Интеграция Summernote и Laravel

Представим, что у нас есть страница редактирования новости. Механика следующая: ваш проект имеет интернет-адрес (URL) вида:

```
/admin/news/edit/<ID>
```

Здесь `<ID>` — это уникальный идентификатор новости в базе данных. Контроллер, связанный в файле `web.php` с этим URL, получает из базы данных новость и передает ее в представление примерно так:

```
public function NewsEdit($id) {

    $item = DB::table('news')->where('id', $id)->first();

    return view('backend/admin-news-edit', [
        'title' => 'Редактирование новости',
        'item'=> $item
    ]);
}
```

Параметр `$item` — это переменная, содержащая объект записи. Для простоты мы не производим проверку на существование новости (вдруг на вход передан некорректный ID), хотя в реальных проектах такая проверка имеет место быть. Таким образом, в представление `admin-news-edit.blade.php` передается переменная `$item`.

Теперь посмотрим на само представление:

```
@include('backend/header')

<form action="/admin/news/save" method="post" enctype="multipart/form-data">
@csrf
<input type="hidden" name="id" value="{{ $item->id }}">

<textarea name="content" rows=25 cols=150>{!! $item->content !!</textarea>

<button style="width: 100px" class="btn btn-block btn-primary"
                                         type="submit">Изменить</button>

</form>

@include('backend/footer')
```

Опять-таки: для простоты весь лишний код мы опустили. Здесь главное, что у нас есть подключаемые представления `header.blade.php` и `footer.blade.php` — заголовок и «подвал» страницы. Они нам понадобятся как раз для подключения редактора Summernote. Внутри самого же представления `admin-news-edit` есть поле `TEXTAREA`, которое пока не содержит никаких визуальных элементов.

Находясь в этом представлении, отредактируйте это поле так:

```
<textarea name="content" rows=25 cols=150 id="summernote">{!! $item->content
                                                                    !!</textarea>
```

Мы добавили `id` элемента — `summernote`. Вы можете использовать любой другой идентификатор — главное указать его в скрипте, вызывающем редактор.

Теперь в заголовок страницы, т. е. в представление `header.blade.php`, добавьте CSS-файл редактора:

```
<!-- summernote -->
<link rel="stylesheet" href="/backend/summernote/summernote-bs4.min.css">
```

В «подвал» страницы (`footer.blade.php`) нужно добавить код:

```
<!-- Summernote -->
<script src="/backend/summernote/summernote-bs4.min.js"></script>
<script src="/backend/summernote/lang/summernote-ru-RU.min.js"></script>
```

```
<script>
$(function () {
  // Summernote
  $('#summernote').summernote({
    lang: 'ru-RU',
    height: 300
  })
})
</script>
```

После этого надо скачать сам редактор и поместить его в каталог `public/backend/summernote` — просто распакуйте загруженный архив и поместите его содержимое в этот каталог. Также проверьте имена файлов — они могут немного отличаться в зависимости от версии Summernote.

На рис. 46.1 показано, как выглядит редактор по умолчанию: из всего множества параметров мы изменили только высоту.

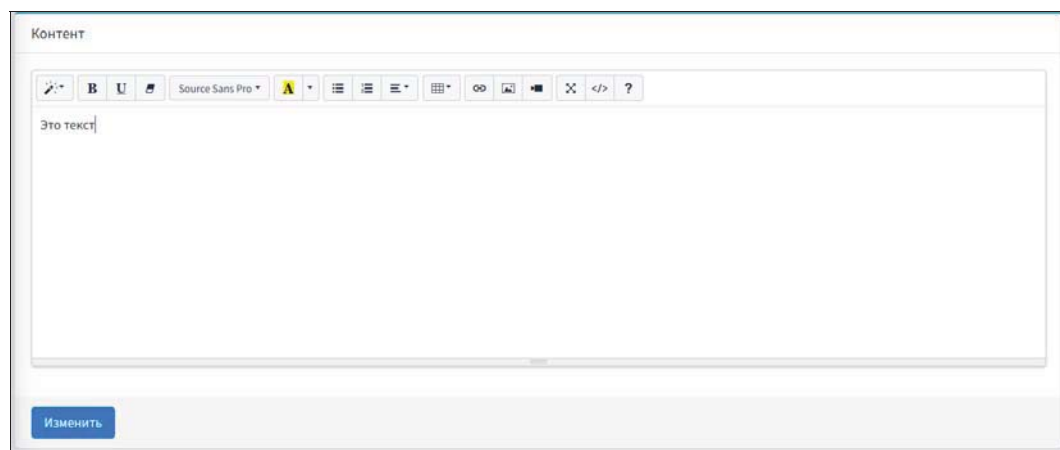


Рис. 46.1. Редактор Summernote

При желании вы можете самостоятельно формировать панели с кнопками:

```
$('#summernote').summernote({
  toolbar: [
    // [groupName, [list of button]]
    ['style', ['bold', 'italic', 'underline', 'clear']],
    ['font', ['strikethrough', 'superscript', 'subscript']],
    ['fontsize', ['fontsize']],
    ['color', ['color']],
    ['para', ['ul', 'ol', 'paragraph']],
    ['height', ['height']]
  ]
});
```

Подробности о кастомизации редактора приводятся в официальном руководстве: <https://summernote.org/deep-dive/#customization>.



## ГЛАВА 47

# Работа с MongoDB средствами PHP

### 47.1. Что такое MongoDB?

MongoDB — это документоориентированная (document-oriented database) система управления базой данных (СУБД), специально предназначенная для хранения иерархических структур данных (документов) и обычно реализуемая с помощью подхода NoSQL (для доступа к данным не используются SQL-операторы). В основе таких СУБД лежат документные хранилища (document store), имеющие структуру дерева (иногда леса).

Структура дерева начинается с *корневого узла* и может содержать несколько внутренних и листовых узлов. *Листовые узлы* содержат данные, которые при добавлении документа заносятся в *индексы*, что позволяет даже при достаточно сложной структуре находить место искомым данным (путь к ним). Интерфейс API для поиска позволяет находить по запросу документы и части документов. В отличие от хранилищ типа «ключ-значение», выборка по запросу к документному хранилищу может содержать части большого количества документов без полной загрузки этих документов в оперативную память.

Документы могут быть организованы (сгруппированы) в *коллекции*. Их можно считать отдаленным аналогом таблиц реляционных СУБД, но коллекции могут содержать другие коллекции. Хотя документы коллекции могут быть произвольными, для более эффективного индексирования лучше объединять в коллекцию документы с похожей структурой. Впрочем, обо всем этом мы еще поговорим.

Документоориентированные базы данных применяются в системах управления содержанием, издательском деле, документальном поиске и т. п. Примеры СУБД этого типа: MongoDB, CouchDB, Couchbase, MarkLogic, eXist.

В этой главе не будет рассказа о том, как работать с MongoDB традиционным образом (через клиенты вроде MongoDBCompass) — мы рассмотрим только, как работать с ней средствами PHP. Также мы не станем приводить основные понятия и термины MongoDB — если вы читаете эту главу, значит, вы уже в курсе. Вместо этого будет показано, как настроить PHP для работы с MongoDB, как добавить информацию в базу данных, как получить информацию и как преобразовать документ в массив для более удобной работы с ним.

## 47.2. Настройка интерпретатора PHP

Нужно отметить, что для поддержки MongoDB в PHP существуют два расширения. Синтаксис одного из них мне понравился больше, но почему-то оно так и не заработало... Разобраться с проблемами тоже не получилось, а сценарий нужно было писать уже сегодня, поэтому пришлось знакомиться с другим расширением. Его синтаксис не так хорош, зато оно работает.

Выполните пошагово следующие действия:

```
sudo apt-get install php-pear php-dev
pecl channel-update pecl.php.net
pecl install mongodb
```

Последняя команда как раз и установит расширение `mongodb.so`. Теперь осталось «прописать» это расширение в конфигурационных файлах PHP. Именно в *файлах* (во множественном числе)! Как уже было отмечено ранее, у PHP есть несколько конфигурационных файлов — под каждую конфигурацию. Если вы используете веб-сервер Apache, то нужный конфигурационный файл называется `/etc/php/<версия>/apache2/php.ini`. Если у вас nginx, то вы будете запускать PHP через сервис PHP-FPM, поэтому нужно редактировать файл конфигурации `/etc/php/<версия>/fpm/php.ini`. Также нужно в любом случае отредактировать файл `/etc/php/<версия>/cli/php.ini`. Этот файл PHP использует, когда запускается не через веб-сервер, а в режиме командной строки.

В каждый из этих файлов нужно добавить строку (можно в самый конец):

```
extension=mongodb.so
```

После этого надо перезапустить веб-сервер:

☐ при использовании Apache:

```
sudo systemctl restart apache2
```

☐ а вот если у вас nginx, то перезапускать нужно не веб-сервер, а PHP-FPM (вместо X и Y укажите версию PHP):

```
sudo systemctl restart phpX.Y-fpm
```

Проверяем, что в php появился новый модуль:

```
php -m | grep mongodb
```

Мы должны увидеть:

```
mongodb
```

Теперь проверим, действительно ли мы можем писать скрипты. Создадим файл `mongo_connect.php`:

```
mcedit mongo_connect.php
```

Содержимое этого файла приведено в листинге 47.1.

**Листинг 47.1. Файл mongo\_connect.php**

```
<?php

$options['tls'] = true;
$options['tlsAllowInvalidCertificates'] = true;
$connection = new MongoDB\Driver\Manager("mongodb://root:123@localhost",
$options);

$command = new MongoDB\Driver\Command(['ping' => 1]);
$cursor = $connection->executeCommand('admin', $command);
$result = $cursor->toArray();

print_r($result);

?>
```

Не спешите его выполнять! Обратите внимание на строку подключения. Как минимум здесь нужно отредактировать имя пользователя (*root*) и пароль (*123*). Если же вы вообще не задавали имя пользователя для подключения к БД, то строка подключения может быть другой:

```
$connection = new MongoDB\Driver\Manager("mongodb://localhost");
```

Здесь мы просто подключаемся к базе данных. Да, параметры (*\$options*) — не обязательны, и их можно не указывать.

Если вы хотите подключиться к облачной базе данных, предоставляемой сервисом MongoDB Atlas, то вам понадобится следующий код:

```
$connection = new MongoDB\Driver\Manager
("mongodb+srv://cluster0.gbbvve0.mongodb.net/drivers",
array("username" => "user",
      "password" => "pass123" ));
```

Разумеется, строку подключения (она выделена курсивом) нужно посмотреть в настройках самого Atlas (когда вы нажимаете кнопку **Connect**), имя сервера (*cluster0.gbbvve0.mongodb.net*) может быть у вас другим, как и имя базы данных (*drivers*). Имя пользователя и пароль (выделены полужирным) также нужно изменить.

Запускаем скрипт на выполнение:

```
php mongo_connect.php
```

Мы должны получить такой вывод:

```
Array
(
    [0] => stdClass Object
        (
            [ok] => 1
        )
)
```

## 47.3. Добавление данных в MongoDB

Для добавления информации в базу данных используется класс `MongoDB\Driver\BulkWrite`, собирающий одну или несколько операций записи, которые должны быть отправлены на сервер. После добавления любого количества операций вставки, обновления или удаления коллекция может быть выполнена с помощью команды `MongoDB\Driver\Manager::executeBulkWrite()`.

Операции записи могут быть отсортированы (по умолчанию) или не отсортированы. Отсортированные операции записи отправляются на сервер в указанном порядке для последовательного выполнения. В случае возникновения ошибки записи любые оставшиеся операции будут прерваны. Неотсортированные операции отправляются на сервер в произвольном порядке, где они могут выполняться параллельно. Сообщения об ошибках, которые возникают, будут отправлены после выполнения всех операций.

Класс `MongoDB\Driver\BulkWrite` можно использовать для вставки (метод `insert()`), обновления (метод `update()`) и удаления (метод `delete()`).

Смешанные операции записи (т. е. добавления, обновления или удаления) будут собраны в типизированные команды записи, которые в последовательном порядке будут отправлены на сервер. Пример операции записи приведен в листинге 47.2.

### Листинг 47.2. Пример выполнения операций записи

```
<?php
// Подключаемся к БД
$connection = new MongoDB\Driver\Manager("mongodb://root:test123@localhost",
$options);

// Формируем очередь операций записи
$bulk = new MongoDB\Driver\BulkWrite(['ordered' => true]);
// Вставка
$bulk->insert(['_id' => 1, 'x' => 1]);
$bulk->insert(['_id' => 2, 'x' => 2]);
// Обновление
$bulk->update(['x' => 2], ['$set' => ['x' => 1]]);
$bulk->insert(['_id' => 3, 'x' => 3]);
// Удаление
$bulk->delete(['x' => 1]);

// Производим запись в БД db, коллекция collection
$connection->executeBulkWrite('db.collection', $bulk);
?>
```

При одновременном добавлении множества документов принято сначала формировать очередь записи, а затем проверять корректность операций записи. Пример приведен в листинге 47.3.

**Листинг 47.3. Проверка корректности операций записи**

```

<?php

$bulk = new MongoDB\Driver\BulkWrite(['ordered' => true]);
$bulk->delete([]);
$bulk->insert(['_id' => 1]);
$bulk->insert(['_id' => 2]);
$bulk->insert(['_id' => 3, 'hello' => 'world']);
$bulk->update(['_id' => 3], ['$set' => ['hello' => 'earth']]);
$bulk->insert(['_id' => 4, 'hello' => 'pluto']);
$bulk->update(['_id' => 4], ['$set' => ['hello' => 'moon']]);
$bulk->insert(['_id' => 3]);
$bulk->insert(['_id' => 4]);
$bulk->insert(['_id' => 5]);

$manager = new MongoDB\Driver\Manager('mongodb://localhost:27017');
$writeConcern = new
MongoDB\Driver\WriteConcern(MongoDB\Driver\WriteConcern::MAJORITY, 1000);

try {
    $result = $manager->executeBulkWrite('db.collection', $bulk, $writeConcern);
} catch (MongoDB\Driver\Exception\BulkWriteException $e) {
    $result = $e->getWriteResult();

    // Если гарантия записи не может быть выполнена
    if ($writeConcernError = $result->getWriteConcernError()) {
        printf("%s (%d): %s\n",
            $writeConcernError->getMessage(),
            $writeConcernError->getCode(),
            var_export($writeConcernError->getInfo(), true)
        );
    }

    // Проверить, не выполнялись ли какие-либо операции записи
    foreach ($result->getWriteErrors() as $writeError) {
        printf("Operation#%d: %s (%d)\n",
            $writeError->getIndex(),
            $writeError->getMessage(),
            $writeError->getCode()
        );
    }
} catch (MongoDB\Driver\Exception\Exception $e) {
    printf("Прочая ошибка: %s\n", $e->getMessage());
    exit;
}

```

```
printf("Вставлено %d документов\n", $result->getInsertedCount());  
printf("Обновлено %d документов\n", $result->getModifiedCount());  
  
?>
```

Обратите внимание, как именно мы добавляем документ в коллекцию:

```
$bulk->insert(['_id' => 3, 'hello' => 'world']);
```

По сути, методу `insert()` мы передаем ассоциативный массив. Поэтому мы можем легко поместить в документ любой ассоциативный массив — например, все поступающие по POST-запросу данные или весь массив `$_SERVER`:

```
// формируем документ по массиву $_POST  
$doc = $_POST;  
$bulk->insert($doc);
```

## 47.4. Чтение информации из базы данных

Для чтения данных из БД, прежде всего, нужно сформировать запрос, — как и в случае с реляционной базой данных. Сделать это можно с помощью класса `Query`. Класс `MongoDB\Driver\Query` — это объект значения, представляющий запрос базы данных. Синтаксис конструктора класса выглядит так:

```
final class MongoDB\Driver\Query {  
    /* Методы */  
    final public __construct(array|object $filter, ?array $queryOptions = null)  
}
```

Первый параметр — это фильтр, т.е. выборка должна соответствовать этому фильтру. Второй параметр — это параметры запроса.

Рассмотрим пример:

```
// Фильтр и опции не установлены, выборка всех документов  
$filter = [];  
$options = [];  
// конструируем запрос  
$query = new MongoDB\driver\Query($filter, $options);  
// выполняем запрос: выбираем все из коллекции аms БД drivers  
// коллекция с таким именем должна существовать  
$rows = $connection->executeQuery('drivers.ams', $query);
```

После этого у нас в массиве `$rows` будут все полученные документы:

```
foreach ($cursor as $document) {  
    echo "<p>" . $document->_id;  
}
```

Этот пример выводит идентификатор каждого документа. Вывести другие поля документа можно, обратившись к ним по имени. Например, если в документе есть поле `temp`, то вывести его можно так:

```
echo "<p>" . $document->temp;
```

Проверить существование поля можно с помощью функции `isset()`:

```
if (isset($document->temp)) {
    echo "<p>" . $document->temp;
}
```

Итак, для получения данных из MongoDB нужно:

1. Сформировать запрос, используя класс `MongoDB\Driver\Query`;
2. Выполнить запрос методом `executeQuery()`, в который нужно передать имя базы данных и коллекции в виде (*база.коллекция*) и сам ранее сформированный запрос.
3. В случае успешного выполнения запроса мы получим объект класса `MongoDB\Driver\Cursor`.

Рассмотрим листинг 47.4. Приведенный в нем сценарий добавляет в коллекцию `db.collection` три документа, затем устанавливает параметры и фильтр выборки и, наконец, производит саму выборку.

#### Листинг 47.4. Пример чтения из базы данных

```
<?php

$manager = new MongoDB\Driver\Manager("mongodb://localhost");

$bulk = new MongoDB\Driver\BulkWrite;
$bulk->insert(['x' => 1]);
$bulk->insert(['x' => 2]);
$bulk->insert(['x' => 3]);
$manager->executeBulkWrite('db.collection', $bulk);

$filter = ['x' => ['$gt' => 1]];
$options = [
    'sort' => ['x' => -1]
];

$query = new MongoDB\Driver\Query($filter, $options);
$cursor = $manager->executeQuery('db.collection', $query);

foreach ($cursor as $document) {
    var_dump($document);
}

?>
```

Результат выполнения этого примера:

```
object(stdClass)#6 (1) {
    ["x"]=>
```

```

    int(3)
}
object(stdClass)#7 (1) {
    ["x"]=>
        int(2)
}

```

Самая важная часть: фильтр и опции:

```

$filter = ['x' => ['$gt' => 1]];
$options = [
    'sort' => ['x' => -1]
];

```

В нашем случае мы задаем, что поле `x` должно быть больше (`gt`) 1. Затем мы устанавливаем сортировку по полю `x` в обратном порядке (`-1` — от большего к меньшему).

Опция `limit` позволяет установить количество документов, которые будут выведены:

```

$options = ['sort'=>array('_id'=>-1),'limit'=>3];

```

Эти опции сортируют документы по полю `id` в обратном порядке и выводят три документа (т. е. будут выведены *последние* три документа).

## 47.5. Преобразование объекта в массив

Когда мы проходимся по возвращенным на запрос документам в цикле, мы работаем с объектом документа. Иногда проще обращаться не к свойствам объекта, а к элементам массива. В этом случае вам понадобится функция `obj2array()`:

```

function obj2array ( &$Instance ) {
    $clone = (array) $Instance;
    $rtn = array ();
    $rtn['__SOURCE_KEYS_'] = $clone;

    foreach ($clone as $key=>$value) {
        $aux = explode ("\0", $key);
        $newkey = $aux[count($aux)-1];
        $rtn[$newkey] = &$rtn['__SOURCE_KEYS_'][$key];
    }

    return $rtn;
}

```

Использовать ее можно так:

```

$cursor = $connection->executeQuery('cars.info', $query);
foreach ($cursor as $d) {
    $document = obj2array($d);
    $dt = $d->_id;
}

```

```
$sddt = $date = date('d-m-Y H:i:s', $dt->getTimestamp());
$field1 = $document['speed'];
$field2 = $document['consum'];
```

```
echo "                                <tr>
    <td>
        $sddt
    </td>
    <td>$field1</td>
    <td>$field2</td>

</tr>";
}
```

Зачем все это нужно? При использовании объекта мы должны заранее знать, как называются его поля. А что делать, если мы этого не знаем? Допустим, есть некоторый документ, описывающий структуру других документов, и в нем пользователем задаются поля, которые нужно вывести из каждого документа в таблицу. Пусть в поле `fields` документа содержится список полей, которые мы должны отобразить в виде таблицы. Сначала мы должны получить эти поля:

```
$fields = $document->fields;
```

Допустим, названия полей записаны через запятую. Превращаем список в массив:

```
$fieldsArray = explode(',', $fields);
$columns = count($fieldsArray);
```

```
// Формируем шапку таблицы с наименованием нужных нам полей
echo "<th>Date</th>";
```

```
foreach ($fieldsArray as $f) {
    echo "<th>$f</th>";
}
```

Далее мы получаем все документы из коллекции, заданной в переменной `$colid`, и выводим в таблице только те поля, которые записаны в `$fieldsArray`:

```
$queryString = "cars." . $colid;
$cursor = $connection->executeQuery($queryString, $query);
```

```
// перебираем все документы
foreach ($cursor as $d) {
    // документ - это уже массив, а не объект
    $document = obj2array($d);
    $dt = $d->_id;
    $sddt = $date = date('d-m-Y H:i:s', $dt->getTimestamp());
    echo "<tr><td>$sddt</td>";
```

```
// Выводим не все, а только поля, заданные в массиве
foreach ($fieldsArray as $f) {
    echo "<td>" . $document[$f] . "</td>";
}
echo "</tr>";

}
```

Также обратите внимание на небольшой трюк — как мы по `id` документа формируем дату и время его добавления в БД:

```
$dt = $d->_id;
$ddate = $date = date('d-m-Y H:i:s', $dt->getTimestamp());
```





# ПРИЛОЖЕНИЯ

Приложение 1.	Шаблоны проектирования
Приложение 2.	Профайлинг
Приложение 3.	Виртуальная машина HHVM
Приложение 4.	Описание электронного архива



# ПРИЛОЖЕНИЕ 1

## Шаблоны проектирования

### П1.1. Введение в шаблоны проектирования

*Шаблоны проектирования* (design patterns) — это проверенные и готовые к использованию решения часто возникающих в повседневном программировании задач. Принцип такой: зачем «изобретать велосипед» заново? В большинстве случаев вы — не единственный, кому приходится решать подобную задачу, и есть огромная вероятность, что кто-то уже решил такую задачу до вас.

Шаблон проектирования — это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Шаблон проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Самое интересное, что он не зависит от языка программирования.

Хороший шаблон может быть с легкостью реализован в большинстве современных языков программирования. Конечно, нужно помнить, что шаблон, примененный неправильно или к неподходящей задаче, может принести немало проблем. Тем не менее, если вы правильно примените шаблон, это поможет решить задачу легко и просто.

Существуют три типа шаблонов: структурные, порождающие и поведенческие:

- ☐ *структурные* шаблоны определяют отношения между классами и объектами, позволяя им работать совместно;
- ☐ *порождающие* шаблоны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом;
- ☐ последний тип шаблонов — *поведенческие*, используются для того, чтобы упростить взаимодействие между сущностями.

Зачем нужны шаблоны проектирования? Как уже отмечалось, шаблон — это продуманное решение той или иной задачи. Представьте, что вам нужно объединить два класса, выполняющих различные операции в зависимости от ситуации. Эти классы интенсивно используются существующей системой, поэтому вы не можете удалить один из них и добавить его функциональность во второй. Да и изменение кода потребует тщательного тестирования и, скорее всего, приведет к ошибкам. Вместо этого вы можете реализовать шаблоны «Адаптер» и «Стратегия» и решить

поставленную задачу с их помощью. Прежде, чем перейти к знакомству с этими шаблонами, рассмотрите код в листинге П1.1.

#### Листинг П1.1. Пример класса

```
class SampleClass {
    private $_class_1;
    private $_class_2;
    private $_context;

    public function __construct( $context ) {
        $this->_context = $context;
    }

    public function operation1() {
        if( $this->_context == "context_1" ) {
            $this->_class_1->operation1_in_class_1_context();
        } else ( $this->_context == "context_2" ) {
            $this->_class_2->operation1_in_class_2_context();
        }
    }
}
```

## П1.2. Шаблон «Стратегия»

«Стратегия» — это поведенческий шаблон, позволяющий выбрать поведение программы в процессе ее выполнения в зависимости от контекста путем инкапсуляции нескольких алгоритмов в разных классах.

Если посмотреть на код в листинге П1.1, то выбор стратегии основан в нем на значении переменной `$context` — оно формируется в момент создания объекта. Если значение было `context_1`, то программа будет использовать класс `class_1`.

Представим, что мы разрабатываем класс, который может создать или обновить запись в базе данных. Очень часто такая ситуация нужна, например, при импорте записей в базу данных: если запись не существует, то нужно ее создать (SQL-оператор `INSERT`), а если она существует, то ее нужно обновить (SQL-оператор `UPDATE`). В обоих случаях входные параметры будут одни и те же (например, артикул товара, описание, количество, цена), но в зависимости от ситуации класс должен будет использовать различные функции для обновления и создания записи. Можно каждый раз переписывать условие `if/else`, а можно создать один метод, который будет принимать контекст:

```
class Product {

    public function CreateOrUpdate($sku, $desc, $qty, $price)
    {
        if( is_null($sku) ) {
            // продукт не существует, создаем запись
```

```
        } else {  
            // запись есть, обновляем ее  
        }  
    }  
}
```

Как правило, этот шаблон подразумевает инкапсуляцию алгоритмов в классы, но в нашем случае это излишне. Помните, что вы не обязаны следовать шаблону слово в слово. Любые варианты допустимы, если они решают задачу и соответствуют концепции.

## П1.3. Шаблон «Адаптер»

«Адаптер» — это структурный шаблон, позволяющий использовать класс, который реализует нужные функции, и имеющий неподходящий интерфейс. Другое название этого класса — «Обертка» (wrapper).

Он «оборачивает» новый интерфейс вокруг класса для его использования. Классический пример: нужно создать класс предметной модели, имея классы объектов в базе данных. Вместо обращения к табличным классам напрямую и вызова их методов по одному можно инкапсулировать вызовы этих методов в одном методе в адаптере. Это не только позволит повторно использовать набор операций, но и избавит вас от постоянного переписывания большого количества кода, если вам потребуется выполнить тот же набор действий в другом месте.

Сравним два примера:

□ без адаптера:

```
$user = new User();  
$user->CreateOrUpdate( // параметры );  
  
$picture = new Picture();  
$picture->CreateOrUpdate( // параметры );
```

□ создадим обертку Profile():

```
class Profile()  
{  
    public function NewUser( // параметры )  
    {  
        $user = new User();  
        $user->CreateOrUpdate( // часть параметров );  
  
        $picture = new Picture();  
        $picture->CreateOrUpdate( // часть параметров );  
    }  
}
```

Использовать эту обертку можно так:

```
$user_profile = new Profile();  
$user_profile->NewUser( // параметры );
```

## П1.4. Шаблон «Фабрика»

«Фабрика» — порождающий шаблон, который представляет собой класс с методом для создания различных объектов.

Основная цель этого шаблона — инкапсулировать процедуру создания различных классов в одной функции, которая в зависимости от переданного ей контекста возвращает необходимый объект.

Обычно этот шаблон используется для создания разных вариантов базового класса. Например, у вас есть класс `Car` и три его варианта: `Sedan`, `Coupe`, `Cabrio`. Используя «Фабрику», вы можете создать разные варианты кнопок в зависимости от контента.

### Листинг П1.2. Класс `Car` и его варианты

```
abstract class Car {  
    protected $_html;  
  
    public function getHtml()  
    {  
        return $this->_html;  
    }  
}  
  
class Sedan extends Car {  
    protected $_html = "Sedan";  
}  
  
class Coupe extends Button {  
    protected $_html = "Coupe";  
}  
  
class Cabrio extends Button {  
    protected $_html = "Cabrio";  
}
```

Напишем нашу «Фабрику» (листинг П1.3).

### Листинг П1.3. Шаблон «Фабрика»

```
class CarFactory  
{  
    public static function createCar($type)
```

```

{
    $baseClass = 'Car';
    $targetClass = ucfirst($type).$baseClass;

    if (class_exists($targetClass) && is_subclass_of($targetClass, $baseClass))
    {
        return new $targetClass;
    } else {
        throw new Exception("The car type '$type' is not recognized.");
    }
}
}

```

Использовать «Фабрику» можно так:

```

$cars = array('Sedan', 'Coupe', 'Cabrio');
foreach($cars as $c) {
    echo CarFactory::createCar($c)->getHtml();
}

```

## П1.5. Шаблон «Одиночка»

Шаблон «Одиночка» (singleton) — довольно популярный в PHP-программировании. Это порождающий шаблон, позволяющий убедиться, что в процессе выполнения программы создается только один экземпляр класса с глобальным доступом.

Этот шаблон обычно используется как «точка координации» для других объектов, поскольку поля «Одиночки» будут одинаковы для всех, кто его вызывает. Если вам необходимо передавать определенный экземпляр из класса в класс, вы можете передавать его каждый раз через конструктор или использовать «Одиночку». Представим, что у вас есть класс `Session`, который содержит данные о текущей сессии. Поскольку сессия инициализируется только один раз, мы можем реализовать его так:

```

class Session
{
    private static $instance;

    public static function getInstance()
    {
        if( is_null(self::$instance) ) {
            self::$instance = new self();
        }
        return self::$instance;
    }

    private function __construct() { }

    private function __clone() { }
}

```

```
        // прочие методы сессии
        ...
        ...
        ...
    }

    // получаем экземпляр сессии
    $session = Session::getInstance();
```

После этого можно получить доступ к сессии из разных участков кода — даже из других классов. Метод `getInstance` всегда будет возвращать одну и ту же сессию.

\* \* \*

В этом приложении приведены далеко не все шаблоны проектирования, а только некоторые, — часто встречающиеся. Информацию о других шаблонах вы можете найти в Интернете. Как уже отмечалось, шаблоны проектирования не зависят от языка программирования. При использовании шаблонов помните, что решаете задачу правильным способом — при неправильном использовании шаблоны проектирования могут доставить больше проблем, чем решить.

## ПРИЛОЖЕНИЕ 2

# Профайлинг

### П2.1. Что такое профайлинг?

*Профайлинг* — это анализ производительности приложения. Профайлинг позволяет найти узкие места в производительности вашего РНР-приложения и существенно повысить скорость его работы. Профайлеры дают возможность пройти по всему стеку вызовов РНР и определить, какие функции вызывались, сколько раз, сколько заняло выполнение той или иной функции.

Раньше профайлеров не было, и весь профайлинг заключался в выводе времени до и после вызова функции. Потом в код функции вносились изменения, и процесс повторялся — программист мог сравнить новые результаты с предыдущими и сделать выводы об эффективности внесенных изменений. Такой процесс был довольно утомительным.

Ранее профайлеры были доступны для «серьезных» языков программирования — вроде C, Java, Pascal. Сейчас же появились профайлеры и для РНР (что косвенно подтверждает переход РНР в «высшую лигу» — теперь на нем можно создавать серьезные приложения, ведь никому в голову не придет выполнять профайлинг программы типа «Hello, world»).

Когда нужно использовать профайлер? Не следует запускать его прямо сейчас и для каждого своего приложения. Профайлер задействуется лишь тогда, когда появились проблемы производительности. Как узнать, что они появились? Например, вы заметили, что приложение стало работать медленнее, чем обычно. Возможно, увеличился объем данных, обрабатываемых приложением. Чтобы ускорить обработку данных в новом, увеличенном объеме, возможно, придется изменять алгоритм их обработки.

### П2.2. Типы профайлеров

Существуют два типа профайлеров: одни запускаются во время процесса разработки приложения, а вторые — уже на production-сервере. Примером профайлера первого типа может служить Xdebug (<https://xdebug.org/>). Поскольку этот профайлер

потребляет очень много ресурсов, его можно использовать только во время разработки. Его результаты не предназначены для чтения человеком, поэтому для их расшифровки понадобится программа вроде KCacheGrind или WinCacheGrind.

В качестве примера профайлера второго типа можно привести XHProf (<http://xhprof.io/>). Его можно использовать как при разработке, так и уже после запуска приложения в производство.

#### **ПРИМЕЧАНИЕ**

Оба профайлера: xhprof и xdebug — являются расширениями PHP, и вы можете их установить с помощью системного менеджера пакетов.

## **П2.3. Профайлер Xdebug**

Xdebug — один из самых популярных PHP-профайлеров. Настройки этого профайлера хранятся в вашем файле `php.ini`. Вот рекомендуемые параметры:

```
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 1
xdebug.profiler_output_dir = /каталог/с/результатами
```

Первая директива говорит PHP не запускать профайлер автоматически. Как уже отмечалось, он довольно требовательный к ресурсам, поэтому его запуск для каждого запроса может отрицательно повлиять на производительность.

Вторая директива указывает, что запуск профайлера будет производиться по требованию. Для этого нужно добавить параметр запроса `XDEBUG_PROFILE=1` к любому интернет-адресу (URL) приложения. Например, вам нужно выполнить профайлинг сценария `page.php`. Запустите его так:

```
http://example.com/page.php?id=1&XDEBUG_PROFILE=1
```

Здесь `id` — это один из параметров, которые передаются приложению, например `id` страницы.

Результаты профайлинга будут записаны в каталог, заданный третьей директивой. Для сложных приложений отчеты профайлера могут быть очень большими — 500 Мб и более. Этот факт нужно учитывать при выборе расположения для отчетов.

Для просмотра отчетов нужно использовать упомянутые ранее программы.

## **П2.4. XHProf**

XHProf — профайлер нового поколения. Его можно запускать как в процессе разработки приложения, так и уже на production-серверах. Он не собирает так много данных, как Xdebug, поэтому потребляет гораздо меньше системных ресурсов.

Для установки профайлера введите следующие команды (в зависимости от вашего дистрибутива):

```
# Ubuntu
sudo apt-get install build-essential
sudo pecl install mongo
sudo pecl install xhprof-beta
# CentOS
sudo yum groupinstall 'Development Tools'
sudo pecl install mongo
sudo pecl install xhprof-beta
```

После установки профайлера нужно добавить в файл `php.ini` строки, обеспечивающие его загрузку:

```
extension=xhprof.so
extension=mongo.so
```

Затем нужно перезапустить процесс PHP: или перезагрузить сервис PHP-FPM, или перезапустить веб-сервер Apache, если вы настроили выполнение сценариев PHP посредством модуля `mod_php`.

Для просмотра результатов этого профайлера нужно установить приложение XHGUI. Это веб-приложение, и оно требует наличия следующих компонентов: Composer, Git, MongoDB, PHP 7.2+, расширение `mongo` для PHP. Также возможно подключение к другой СУБД посредством PDO.

Будем считать, что кое-что мы уже установили (MongoDB, расширение `mongo` и PHP). Также у вас уже может быть установлен `git`, поэтому введите следующие команды для запуска:

```
cd /var/www
git clone https://github.com/perf-tools/xhgui.git
cd xhgui
php install.php
```

Здесь мы предполагаем, что репозиторий Git будет загружен в каталог `/var/www`. В нем будет создан новый подкаталог `xhgui`, который нужно добавить в конфигурацию вашего веб-сервера, чтобы вы могли запустить XHGUI.

Откройте файл конфигурации `config/config.default.php` в текстовом редакторе. По умолчанию профайлер собирает только 1% от всех HTTP-запросов. Это хорошо в production-окружении, но плохо во время разработки. Чтобы повысить степень сбора данных, найдите следующие строки:

```
'profiler.enable' => function() {
return rand(0, 100) === 42;
},
```

и замените их на эти:

```
'profiler.enable' => function() {
return true; // <-- Запуск при каждом запросе
},
```

Чтобы профайлер срабатывал, вам нужно подключить к вашему приложению файл `external.header.php` — в самом начале вашего приложения. Можно также использовать опцию `auto_prepend_file` в `php.ini`:

```
auto_prepend_file = /var/www/xhgui/external/header.php
```

Перезапустите PHP, и профайлер начнет собирать информацию и сохранять ее в MongoDB. Просмотреть собранную информацию можно с помощью XHGUI — обратитесь для этого по адресу, который вы задали в настройках виртуального узла для этого приложения.

## ПРИЛОЖЕНИЕ 3

# Виртуальная машина HHVM

### П3.1. Что такое HHVM?

Некоторые серьезные проекты, ранее написанные на PHP, превосходили возможности интерпретатора PHP, который больше не удовлетворяет запросы проектов по производительности.

Так вот, разработчики одного из таких проектов (не хочется его упоминать, дабы не создавать лишнюю рекламу) устали бороться с оптимизацией PHP, поэтому разработали язык Hack (основан на PHP) и виртуальную машину HHVM для его интерпретации.

Виртуальная машина HHVM (Hip Hop Virtual Machine) — это альтернативный движок PHP. Основная причина разработки HHVM заключалась в том, что на момент выпуска первой HHVM (а это 2010 год) производительность PHP оставляла желать лучшего, поэтому был нужен альтернативный, более быстрый движок.

Таким движком и стала HHVM, представляющая собой JIT-компилятор (Just In Time), который по производительности гораздо лучше, чем PHP-FPM. Многие разработчики после этого перешли на HHVM и рекомендуют эту виртуальную машину для своих проектов — например, WordPress, MediaWiki.

Но одним только движком дело не ограничилось. Был разработан также новый язык программирования — Hack, основанный на PHP. Hack обратно совместим с PHP, т. е. PHP-код можно запускать в HHVM, а вот Hack-код выполнить в PHP нельзя. Язык Hack расширяет возможности PHP, добавляя строгую типизацию переменных, новые структуры данных и др.

Язык PHP всегда был интерпретируемым. Даже после появления байт-кода язык по большому счету остается интерпретируемым. Это означает, что PHP-код остается PHP-кодом до тех пор, пока он не будет пропущен через интерпретатор и не превратится в данные, которые будут отправлены в браузер.

Да, позже был разработан байт-код, что позволило его кешировать и существенно повысить производительность PHP-приложений. Но все равно при этом PHP остался интерпретатором. А интерпретатор существенно проигрывает в скорости компилятору. Такой компилятор был-таки создан — PHP-to-C++, или HHVMc.

Этот компилятор конвертирует PHP-код в C++, а затем компилирует код C++ в исполняемый файл, который размещается на production-серверах. С помощью HHVM можно существенно повысить производительность PHP-приложений на своих серверах. Однако у HHVM есть и недостатки: он не полностью совместим с PHP, а также требует дополнительного времени на компиляцию, что в корне меняет весь процесс разработки.

На базе HHVM и была создана виртуальная машина HHVM. Она конвертирует и кеширует PHP-код в промежуточный формат байт-кода, а затем использует JIT-компилятор для трансляции и оптимизации этого байт-кода в машинный код x86\_64. Компилятор JIT предоставляет возможность низкоуровневой оптимизации производительности, которая была невозможна при использовании HHVM. Также HHVM не изменяет процесса разработки — для программиста она полностью прозрачная, большинство программистов, если бы не знали, что на сервере установлена HHVM, и не заметили бы этого факта. Вот, что вам нужно знать о HHVM прямо сейчас:

- ☐ программист запускает PHP-сценарии через исполнимый файл hhvm, а не php;
- ☐ этот же исполнимый файл используется для создания FastCGI-сервера — вместо `php-fpm`;
- ☐ HHVM использует `php.ini` для чтения конфигурации — директивы те же самые;
- ☐ HHVM поддерживает большинство расширений PHP.

Получить дополнительную информацию о HHVM можно в статье по адресу: <https://www.wpoven.com/blog/hhvm-vs-php-7-performance-showdown-wordpress-nginx/>. В этой же статье вы увидите график производительности HHVM и PHP.

## П3.2. Подойдет ли HHVM именно для вас?

Получить более высокую производительность — это очень заманчиво, но подойдет ли HHVM именно для вас? Чтобы ответить на этот вопрос, вам нужно просмотреть список совместимых с HHVM PHP-расширений, размещенный по адресу: <https://tinyurl.com/2twz9d2a>

Понятно, что если нужного вам расширения не будет в списке, то мигрировать на HHVM не получится. Список поддерживаемых расширений выглядит так: bz2, gd, curl, datetime, iconv, imap, json, libxml, mail, mcrypt, mysql, pdo, pdo\_mysql, pgsql, xml, xmlreader/xmlwriter, zip/zlib, xdebug.

Все эти расширения поддерживаются (и не только эти). Думаю, этого будет вполне достаточно для практически любых приложений. Кстати, заметьте, что, начиная с PHP 7, расширение `mysql` не поддерживается, зато его поддерживает HHVM. Также я не нашел в списке расширение `mysqli`, но зато есть расширения `pdo` и `pdo_mysql`. Отсюда можно сделать вывод: если ваше приложение использует `mysqli`, то его перенести на HHVM не получится, ну или придется переписать его с использованием либо расширения `mysql`, либо `pdo`.

## ПЗ.3. Установка HHVM

Установить HHVM очень просто, особенно учитывая, что HHVM входит в состав большинства популярных дистрибутивов Linux. Изначально HHVM разрабатывалась для Ubuntu, но сейчас ее можно встретить и в других дистрибутивах, в том числе в Debian и Fedora.

Для установки HHVM в Ubuntu введите следующие строки:

```
sudo apt-get install software-properties-common
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80
0x5a16e7281be7a449
sudo add-apt-repository "deb http://dl.hhvm.com/ubuntu $(lsb_release -sc) main"
sudo apt-get update
sudo apt-get install hhvm
```

В Debian нужно ввести следующие строки:

```
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80
0x5a16e7281be7a449
echo deb http://dl.hhvm.com/debian jessie main | sudo tee
/etc/apt/sources.list.d/hhvm.list
sudo apt-get update
sudo apt-get install hhvm
```

В Debian 7 команды установки будут такими:

```
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80
0x5a16e7281be7a449
echo deb http://dl.hhvm.com/debian wheezy main | sudo tee
/etc/apt/sources.list.d/hhvm.list
sudo apt-get update
sudo apt-get install hhvm
```

После установки HHVM давайте напишем простейший сценарий PHP (что-то вроде «Hello, world!») и попробуем его выполнить с помощью HHVM:

```
hhvm test.php
```

## ПЗ.4. Настройка HHVM

Параметры, относящиеся к PHP, HHVM будет автоматически брать из файла `php.ini` (он находится в каталоге `/etc/hhvm`). Как уже отмечалось, HHVM использует тот же формат файла и те же директивы.

Если вы планируете запускать HHVM как FastCGI-сервер (т. е. вместо PHP-FPM), вам нужно отредактировать файл `/etc/hhvm/server.ini`. Подробное описание этого файла вы найдете в официальной документации: <https://docs.hhvm.com/hhvm/configuration/INI-settings>.

Впрочем, стандартные настройки вполне приемлемы (листинг ПЗ.1).

**Листинг ПЗ.1. Файл server.ini**

```
; php options

pid = /var/run/hhvm/pid

; hhvm specific

hhvm.server.port = 9000
hhvm.server.type = fastcgi
hhvm.server.default_document = index.php
hhvm.log.use_log_file = true
hhvm.log.file = /var/log/hhvm/error.log
hhvm.repo.central.path = /var/run/hhvm/hhvm.hhbc
```

Обратите внимание: здесь используется тот же номер порта (9000), что и для PHP-FPM, поэтому, если PHP-FPM у вас уже запущен, его нужно деактивировать (завершить процесс и отключить запуск при загрузке).

Запустить FastCGI-сервер можно командой:

```
hhvm -m server -c /etc/hhvm/php.ini -c /etc/hhvm/server.ini
```

Здесь мы указываем режим сервера (`-m server`) и оба файла конфигурации. Впрочем, правильнее будет настроить HHVM на автоматический запуск. Для этого установите сначала nginx или Apache, а затем введите команды:

```
sudo /usr/share/hhvm/install_fastcgi.sh
sudo systemctl restart hhvm
sudo systemctl restart nginx          (если у вас nginx)
sudo systemctl restart apache2        (если у вас apache)
sudo update-rc.d hhvm defaults
```

Конфигурация веб-сервера останется такой же. В случае с nginx она будет выглядеть так:

```
server {
    listen 80;
    server_name example.com;
    index index.php;
    client_max_body_size 50M;
    error_log /home/deploy/apps/logs/example.error.log;
    access_log /home/deploy/apps/logs/example.access.log;
    root /home/deploy/apps/example.com/current/public;
    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }
    location ~ \.php {
        include fastcgi_params;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME u ment_root$fastcgi_script_name;
        fastcgi_pass 127.0.0.1:9000;
    }
}
```

Теперь у вас вместо PHP работает HHVM, и вы можете наслаждаться повышенной производительностью ваших сценариев. Но это еще не все. Нам нужно рассмотреть язык Hack!

## П3.5. Язык Hack

Hack подобен PHP — можно сказать, что он является диалектом языка PHP. Язык Hack добавляет некоторые новые структуры данных и интерфейсы, позволяющие экономить время. Также Hack использует статическую типизацию, что делает код более предсказуемым и стабильным.

*Статическая типизация* подразумевает обязательное указание типа при объявлении переменной. При динамической типизации, которая используется в PHP, тип переменной определяется автоматически — по присвоенному значению. Вот как выглядит код Hack (листинг П3.2).

### Листинг П3.2. Пример кода Hack

```
<?hh
class MyClass {
    const int MyConst = 0;
    private string $x = '';
    public function increment(int $x): int {
        $y = $x + 1;
        return $y;
    }
}
```

Первое, что бросается в глаза, — это использование тега `<?hh` вместо `<?php`. Тег `<?hh` — это признак Hack. Второе — обязательное указание типа при объявлении переменной. В PHP такого не было. Можно было смело использовать вот такой код:

```
$MyVar = 1;           // переменная MyVar - число (int)
$MyVar = "string";    // а сейчас MyVar - строка
```

С одной стороны, динамическая типизация — довольно удобна, но в конечном итоге она может привести к различным ошибкам, которые проявят себя только во время выполнения.

Кроме статической типизации, язык Hack поддерживает некоторые другие возможности — такие как обобщения, коллекции, лямбда-выражения, механизмы асинхронного программирования, составные *shape*-структуры, средства для переопределения имен типов.

Дополнительную информацию о языке Hack можно получить по адресам:

<https://docs.hhvm.com/hack/> и <https://hacklang.org/>.

# ПРИЛОЖЕНИЕ 4

## Описание электронного архива

Электронный архив к книге выложен на сервер издательства по адресу: <https://zip.bhv.ru/9785977518307.zip>. Ссылка доступна и со страницы книги на сайте <https://bhv.ru>.

Структура архива представлена в табл. П4.1.

*Таблица П4.1. Структура электронного архива*

Файл	Описание
software	Все необходимое программное обеспечение (Apache, PHP, MySQL, XAMPP и пр.)
Glava_2	Листинги из главы 2
Glava_6	Листинги из главы 6
Glava_7	Листинги из главы 7
Glava_12	Листинги из главы 12
Glava_15	Листинги из главы 15
Glava_16	Листинги из главы 16
Glava_17	Листинги из главы 17
Glava_19	SQL-код из главы 19
Glava_20	Листинги из главы 20
Glava_22	Листинги и шаблоны из главы 22
Glava_24	Листинги из главы 24
Glava_25	Листинги из главы 25
Glava_26	Листинги из главы 26
Glava_28	Листинги из главы 28
Glava_29	Сценарий загрузки файлов
Glava_30	Листинги из главы 30
Glava_31	Листинги из главы 31, классы отправки и приема почты

Таблица П4.1 (окончание)

Файл	Описание
Glava_32	Листинги из главы 32
Glava_33	Листинги из главы 33
Glava_34	Сценарий работы с MP3-файлами
Glava_35	Листинги из главы 35
Glava_39	Сценарий импорта новостей, парсер SimpleHTML
Glava_43	Листинги из главы 43
Glava_44	Листинги из главы 44
Glava_47	Листинги из главы 47

# Предметный указатель

## \$

`$_FILES` 338

## A

`addslashes()` 155  
`addslashes()` 155  
`APOP` 365  
`array()` 110  
`array_count_values()` 130  
`array_diff()` 127  
`array_diff_assoc()` 127  
`array_fill()` 126  
`array_fill_keys()` 126  
`array_flip()` 130  
`array_key_exists()` 129  
`array_keys()` 130  
`array_map()` 132  
`array_merge()` 114  
`array_merge_recursive()` 114  
`array_pop()` 123  
`array_product()` 128  
`array_push()` 123  
`array_rand()` 129  
`array_replace()` 131  
`array_reverse()` 120  
`array_search()` 131  
`array_shift()` 124  
`array_slice()` 126  
`array_sum()` 128  
`array_unique()` 129  
`array_unshift()` 124  
`array_values()` 130  
`array_walk()` 132  
`array_walk_recursive` 132  
`asort()` 118

## B

`basename()` 175  
`bin2hex()` 164  
Blowfish 158  
Breakpoint 325

## C

CGI 77, 81  
CGI-приложение 78  
`checkdate()` 139  
`chop()` 155  
`chr()` 157  
`compact()` 125  
Content-Transfer-Encoding 359  
Content-type 359  
`convert_cyr_string()` 156  
`convert_uuencode()` 156  
Cookies 306, 435  
`copy()` 177  
`count()` 107  
`count_chars()` 160  
`covert_uudecode()` 156  
CRC3 158  
`crc32()` 158  
`create_watermark()` 197  
`crypt()` 158  
CSV 159, 173, 363  
`current()` 108

## D

`date()` 139  
`define()` 53  
`DeleteMessage()` 366  
DES 158  
`dirname()` 180

`disable_functions` 73, 455  
`display_errors` 73, 455

## E

Eclipse 326  
`end()` 108  
`error_reporting` 73, 455  
`exif_imagetype()` 182  
`exif_read_data()` 183  
`explode()` 160  
`extract()` 125

## F

`fclose()` 171  
`feof()` 170, 171  
`fflush()` 173  
`fgetc()` 173  
`fgets()` 172, 200  
`fgetss()` 171, 172  
`file()` 87, 168, 171  
`file_exists()` 175  
`file_get_contents()` 168, 172  
`file_put_contents()` 172  
`file_uploads` 336  
`fileatime()` 178  
`filemtime()` 178  
`filesize()` 178  
`filetype()` 178  
FileZilla 29  
`flock()` 179  
`fopen` 169  
`fopen()` 168, 169  
`fprintf()` 163  
`fputc()` 174  
`fputs()` 200  
`fread` 168

fread() 170, 200  
fsockopen() 199  
ftp\_chdir() 353  
ftp\_connect() 353  
ftp\_exec() 355  
ftp\_fget() 354  
ftp\_fput() 354  
ftp\_login() 353  
ftp\_mkdir() 354  
ftp\_nlist() 354  
ftp\_pasv() 356  
ftp\_pwd() 354  
ftp\_rmdir() 354  
func\_get\_args() 213  
fwrite() 172, 200

## G

get\_defined\_constants() 57  
getenv() 81  
gethostbyaddr() 206  
gethostbyname() 206  
gethostbynameel() 207  
getTag() 386

## H

Header() 83, 186  
hex2bin() 164  
html\_entity\_decode() 153  
htmlentities() 153  
HtmlMimeMail() 360  
HtmlSpecialChars() 152  
HtmlSpecialChars() 443  
htmlspecialchars\_decode() 152  
HTML-код 45  
HTML-форма 85  
◇ элементы формы 88

## I

ID3-теги 384  
imagearc() 192  
imagechar() 188  
imagecharup() 188  
imagecolorallocate() 188  
imagecolorat() 189  
imagecolortransparent() 190  
imagecopyresampled() 194  
ImageCreateFromPng() 186  
imageellipse() 192  
imagefilledarc 192

imagefilledellipse 192  
imagefilledpolygon() 193  
imagefilledrectangle 192  
ImageGif() 186  
imagepolygon() 193  
imagerectangle() 192  
imagesize 194  
imagerotate() 193  
imagestring() 187  
imagestringup() 189  
imagesx() 194  
imagesy() 194  
imagefttext() 189  
implode() 160  
in\_array() 128  
ini\_set() 454  
is\_array() 94  
is\_numeric() 94  
is\_numeric(): 446  
is\_string() 94  
isset() 52

## J

JavaServer Pages 40  
JIT-компилятор (Just In Time)  
485  
join() 87, 160  
jpeg2wbmp() 185

## K

key() 108  
ksort() 119

## L

lcfirst() 154  
list() 110  
ListMessages(): 366  
log\_errors 73, 455  
ltrim() 155  
Лиректива @csrf 287

## M

mail() 359  
MD5 158  
md5() 158  
md5\_file() 158  
Mercury 40

Microsoft Internet Information  
Server 77  
mkdir() 180  
money\_format() 164  
move\_uploaded\_file() 177  
MP3\_id 385  
mt\_rand() 137  
Multipart-форма 336  
mysql\_num\_rows 250

## N

natcasesort() 122  
natsort() 122  
next() 108  
nl2br() 152  
number\_format() 164

## O

Open() 365  
opendir() 181  
Oracle 370  
ord() 157

## P

PEAR 370, 385  
PECL 371  
Photoshop 182  
PHP Expert Editor 27  
php.ini 454  
PHPMailer 41  
png2wbmp() 185  
pos() 109  
preg\_match() 208  
prev() 108  
primary index 234  
primary key 234  
print() 161  
print\_r() 115  
printf() 162

## Q

Quoted-Printable 360

## R

range() 127  
readdir() 181  
realpath() 175

rename() 177, 178  
 Reply-To 359  
 Request For Comments, RFC 201  
 reset() 108  
 RetrieveMessage() 366  
 rmdir() 180  
 rsort() 117  
 rtrim() 154

## S

scandir() 180  
 secondary index 234  
 setlocale() 163  
 SHA1 158  
 sha1() 158  
 sha1\_file() 158  
 shuffle() 120  
 sizeof() 107  
 Smarty 281  
 smtp\_port 360  
 socket\_set\_blocking() 206  
 sort() 117  
 sprintf() 162  
 SQL 231  
 SQL injection 441  
 SQL-модификатор  
 ◇ AUTO\_INCREMENT 237  
 ◇ DEFAULT 237  
 ◇ NOT NULL 237  
 ◇ PRIMARY KEY 237  
 SQL-оператор  
 ◇ CREATE 235  
 ◇ DELETE 240  
 ◇ GROUP BY 243  
 ◇ HAVING 244  
 ◇ INSERT 238  
 ◇ SELECT 239  
 ◇ UPDATE 238

SQL-функция  
 ◇ AVG 241  
 ◇ COUNT 241  
 ◇ MAX 241  
 ◇ MIN 241  
 ◇ SUM 241  
 SSL-сертификат 447, 448, 449  
 str\_ireplace() 151  
 str\_repeat 151  
 str\_replace() 151  
 str\_word\_count() 161  
 strcasecmp() 151  
 strcmp() 150  
 strcspn() 149, 152  
 strip\_tags() 154, 443  
 stripos() 149  
 stripslashes() 155  
 stristr() 150  
 strlen() 148  
 strnatcasecmp() 151  
 strnatcmp() 151  
 strncasecmp() 151  
 strncmp() 151  
 strpos() 149  
 strev() 155  
 strripos() 150  
 strrpos() 150  
 strspn() 149, 152  
 strstr() 150  
 strtolower() 155  
 strtotime() 138  
 strtoupper() 155  
 substr() 150  
 substr\_count() 161

## T

time() 138  
 timestamp 138, 178  
 tmpfile() 173  
 Tomcat 40

touch() 178  
 TPL-шаблоны 278  
 trim() 155

## U

uasort() 120  
 ucfirst() 156  
 ucwords() 156  
 uksort() 120  
 unlink() 178  
 unserialize() 126  
 unset() 52, 112  
 upload\_max\_filesize 336  
 upload\_tmp\_dir 336  
 urldecode() 155  
 urlencode() 155  
 usort() 120  
 UTF-8 164

## V

vfprintf() 163  
 vprintf() 163

## W

Watch-список 325  
 watermark 195  
 wordwrap() 156

## X

XAMPP 39  
 XSS 441  
 XSS-атака 442

## Z

Zend Studio 27

**А**

Ассоциативный массив 108  
Атрибуты 305

**Б**

База данных 233  
Блок  
◊ catch 313  
◊ finally 315  
Блокировка 179  
Брутфорс 440

**В**

Визуальный редактор  
Summernote 459  
Виртуальная машина HHVM  
485  
Виртуальный выделенный  
сервер, VDS 405  
Владелец 166  
Вторичный индекс 234  
Выражение 57

**Г**

Генераторы 303  
Глобальный обработчик  
исключений 317

**Д**

Дескриптор 199  
Деструктор 292  
Динамическая типизация 489  
Директива  
◊ @forelse 284  
◊ @if 283  
◊ error\_reporting 330  
◊ short\_open\_tag 44  
Директивы Blade 282  
Документ RFC 201  
Документоориентированные  
базы данных 462  
Доменное имя 407

**З**

Заголовки HTTP 82  
Запись 234

**И**

Импорт  
◊ константы 302  
◊ функции 302  
Имя  
◊ переменной в PHP 325  
◊ маршрута 416  
Индексы 462  
Инкапсуляция 289  
Инструкция  
◊ echo 44  
◊ include 274  
◊ include\_once 274, 277  
◊ namespace 300  
◊ require 95, 274, 275  
◊ require\_once 274, 277  
Инструмент cURL 390  
Исключение 313  
Итераторы 298

**К**

Класс 288  
◊ MongoDB\Driver\BulkWrite  
465  
◊ MongoDB\Driver\Query 467  
◊ PHPMailer 361  
◊ POP3 365  
◊ Request 425  
◊ Response 428  
◊ абстрактные методы 295  
◊ защищенные переменные  
294  
◊ модификаторы 294  
◊ частные переменные 294  
Клонирование 296  
Ключевое слово throw 313  
Код ответа 413  
Кодировка символов 153, 156  
Коллекции 462  
Команда  
◊ chmod 167  
◊ получение вывода  
команды в переменную 62  
Комментарии 44  
Константы 53  
Конструктор 291  
◊ запросов 262  
Контроллер 418  
Корневой узел 462

**Л**

Листовые узлы 462  
Локаль 163

**М**

Маршрут 414  
Массив 107  
◊ \$\_GET 79  
◊ \$\_POST 80  
◊ \$\_REQUEST 80  
◊ замена в массиве 131  
◊ поиск в массиве 131  
◊ сравнение 127  
Метод  
◊ AddComment() 421  
◊ download() 433  
◊ GET 79  
◊ GetArchiveContent() 420  
◊ insert() 262  
◊ List() 419  
◊ POST 79  
◊ select() 261  
◊ statement() 262  
◊ update() 262  
◊ View() 420  
◊ where() 263  
◊ «грубой силы» 440

**Н**

Наследование 289, 293  
Натуральная сортировка 121

**О**

Объект 289, 295  
ООП 288  
Оператор  
◊ -> 291  
◊ break 66  
◊ continue 66  
◊ foreach 108  
◊ if 63  
◊ instanceof 298  
◊ switch-case 66  
◊ равенства 60  
Операции  
◊ арифметические 57  
◊ логические 58

Операции (*прод.*)

- ◊ получение вывода системной команды 62
- ◊ строковые 61

**П**

- Паттерн MVC 413
- Первичный индекс 234
- Переменная 48
  - ◊ инициализация 48
  - ◊ логическая (булевая) 51
  - ◊ проверка существования 52
  - ◊ уничтожение 52
  - ◊ функции проверки типа 50
- Переменные
  - ◊ типы 49
  - ◊ окружения 81
- Поведенческий шаблон 476
- Подбор пароля по словарю 440
- Поле 234
- Полиморфизм 289
- Порождающий шаблон 478
- Права доступа 166
- Префикс маршрута 417
- Примеси 302
- Пространства имен 299
- Протокол
  - ◊ HTTP 412
  - ◊ IMAP 367
  - ◊ POP3 365
- Профайлер
  - ◊ Xdebug 481
  - ◊ XHProf 482
- Профайлинг 481
- Псевдонимы 301

**Р**

- Расширение mongodb.so 463
- Рекурсия
  - ◊ непрямая 212
  - ◊ прямая 212
- Репозиторий Git 319

**С**

- Свойство 290
- Сертификат Let's Encrypt 452
- Сессии 311
- Система
  - ◊ Eclipse 327
  - ◊ Eloquent 265
  - ◊ автоматического тестирования SimpleTest 327
  - ◊ контроля версий 318
- Скалярная переменная 283
- Сокет 199
- Список 108
- Ссылки 52
  - ◊ символические 52
- Статическая типизация 489
- Структурный шаблон 477
- СУБД 233
- Сырые (raw) запросы 261

**Т**

- Таблица 233
- Теги 44
- Точка останова 325

**У**

- Указатель \$this 291
- Уязвимости 441

**Ф**

- Файл конфигурации
  - ◊ Apache 407
  - ◊ Laravel 409
- Фасад Storage 430
- Фреймворк 401
  - ◊ CodeIgniter 404
  - ◊ Laravel 260, 402, 409
  - ◊ Symfony 403
  - ◊ Yii 403
- Функции для работы с файлами 200

## Функция

- ◊ die() 44, 326
- ◊ is\_cc\_num() 96
- ◊ is\_email() 95, 208
- ◊ setcookie() 307
- ◊ vsprintf() 163
- ◊ вложенность 211
- ◊ область видимости 211
- ◊ проверка доступа к файлу или каталогу 176
- ◊ проверка типа переменной 50

**Х**

## Хелпер

- ◊ cookie() 435
- ◊ session() 433

**Ц**

- Циклы 64

**Ш**

- Шаблонизатор 278
- Шаблоны проектирования 475

**Э**

- Экземпляр класса 289

**Я**

- Язык программирования Hack 485