Научитесь создавать динамические Web-страницы с использованием PHP!

Более 100 миллионов для "чайников" продано во всем мире

PHP 5 для "чайников"™

Для сомневающихся

Исчерпывающее описание PHP 5 для Windows, Unix, Linux и Mac

Джанет Валейд
Автор книги
PHP & MySQL for Dummies

Диалектика

PHP 5

для "чайников"™

Джанет Валейд

ДИАЛЕКТИКА

Москва ◆ Санкт-Петербург ◆ Киев

2005

ББК 32.973.26-018.2.75
В15
УДК 681.3.07

Компьютерное издательство "Диалектика"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция канд. техн. наук *А.Ю. Шелестова*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Валейд, Джанет

В15 PHP 5 для "чайников".: Пер. с англ. — М. : Издательский дом "Вильямс", 2005. — 320 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0851-5 (рус.)

Данная книга является введением в область Web-программирования на языке PHP 5. С ее помощью можно быстро написать сценарий для Web, обеспечить взаимодействие с файлами и базами данных, а также решить другие задачи. Материал книги также позволит избежать многих распространенных ошибок. Описание основных возможностей языка сопровождается примерами.

В книге можно также найти рекомендации по установке модуля PHP 5 для Web и для работы в командной строке, а также установке и настройке популярных Web-серверов Apache и IIS.

Данная книга будет полезна для начинающих разработчиков, а также всех тех, кто интересуется вопросами программирования для Web.

ББК 32.973.26-018.2.75

ISBN 5-8459-0851-5 (рус.)
ISBN 0-7645-4166-8 (англ.)

Оглавление

Содержание

Об авторе

Джанет Велейд (Janet Valade) — автор нескольких книг по языку PHP, а также некоторых глав к книгам по системе Linux и Web-дизайну.

Джанет Велейд имеет двадцатилетний опыт общения с компьютером. Она работала Web-дизайнером и Web-программистом в специализированной компьютерной компании, а до этого в течение нескольких лет была системным аналитиком в университете. Она руководила развертыванием и использованием компьютерных ресурсов, разрабатывала архивы данных, обеспечивала техническую поддержку сотрудников и студентов, написала множество технических статей, разрабатывала и проводила семинары по различным техническим вопросам.

Посвящение

Эта книга посвящается всем, кто найдет ее полезной.

Благодарности

Хочется выразить благодарность всему сообществу разработчиков открытого кода. Без них, отдавших время и талант делу своей жизни, язык PHP не развивался бы так бурно, и мне было бы не о чем писать. Более того, я бы никогда не смогла самостоятельно освоить PHP без списков рассылки, в которых опытные разработчики терпеливо отвечают на глупые вопросы новичков. Многие идеи этой книги родились именно благодаря изучению этих вопросов и ответов на них.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@dialektika.com

WWW: http://www.dialektika.com

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152

Введение

Раз вы держите в руках эту книгу, значит, вас интересуют сценарии PHP. Вероятно, вы хотите научиться программировать и слышали, что PHP — очень простой язык. Это действительно так. С языка PHP можно начинать изучение программирования.

Возможно, вы разрабатываете интерактивный Web-узел и в качестве языка хотите использовать PHP. И здесь вы правы. С помощью этого языка вы сможете обеспечить очень эффективное взаимодействие с посетителями своего Web-узла.

Возможно, вам нужно в короткий срок разработать обычное приложение. Вы слышали, что PHP освоить очень легко. Да, это так. При создании языка PHP ставилась отдельная цель — обеспечить *простоту его изучения*.

А может, вы администрируете некоторую систему и надеетесь, что язык PHP поможет обеспечить эффективную работу с файлами или обработку данных. Он действительно может почти все. С его помощью можно даже написать электронное сообщение своему шефу с предложением перейти к использованию PHP.

Об этой книге

Эту книгу можно рассматривать как доступное введение в область программирования на языке PHP: в ней рассматриваются основные вопросы программирования вообще и особенности языка PHP в частности. Книга начинается с основ PHP. Сначала вы сможете определить, нужно ли вам устанавливать PHP. (Детальные инструкции по установке PHP содержатся в приложении А.) В книге описаны основные возможности языка PHP с примерами их использования. Если у вас есть опыт программирования, вы можете пропустить этот раздел.

В книге описаны основные варианты использования PHP. Вы узнаете, как написать сценарий для Web, взаимодействовать с файлами и базами данных, а также научитесь решать другие типичные задачи. Кроме того, в книге содержатся рекомендации по эффективному использованию PHP и рассказывается, как избежать ошибок. Информация, содержащаяся в приложениях, поможет новичкам и опытным программистам быстро написать много интересных приложений.

Как использовать эту книгу

Книга написана как справочник, а не как учебное пособие, поэтому вам не придется читать ее от корки до корки. Вы можете начать изучение книги с любого вопроса. Книга хорошо структурирована, поэтому по оглавлению можно легко найти нужную главу. В каждой главе содержатся ссылки на другие главы, в которых можно найти информацию, связанную с данным вопросом.

В книге содержится множество примеров кода на PHP, включающих как несколько строк, так и целые программы. Операторы PHP в книге выделены следующим образом:

```
Это оператор сценария PHP
```

В тексте книги операторы PHP тоже выделяются специальным шрифтом, например, `вот так`

В примерах кода некоторые слова выделены курсивом. Они обычно описывают параметры, которые необходимо заменять реальными значениями. Например, в команде

```
echo число1, число2
```

выделенные курсивом параметры нужно заменить реальными числами. При использовании этого оператора в сценарии он может иметь вид

```
echo 3, 127
```

Кроме того, примеры кода могут содержать многоточия, которые не нужно вводить в сценарии. Они лишь означают, что в списке можно использовать любое количество параметров. Например, если вам встретится строка

```
echo число1, число2, ...
```

в своем сценарии многоточие вводить не нужно, вместо него можно использовать реальные параметры.

Очевидные предположения

Надеюсь, вы достаточно много знаете о компьютерах, и вас не пугают такие термины, как файл, каталог, путь, или другие основные понятия, связанные с операционной системой. Предполагается, что вы умеете скопировать файл в нужный каталог.

Предполагается также, что вы умеете создавать файлы, редактировать их в редакторе, удалять, копировать и перемещать.

Если вы хотите работать с PHP, то должны использовать операционную систему, которая поддерживает этот язык. Операционная система должна быть сравнительно новой версии. Например, Windows 95 является очень старой, как и Mac OS 9. Даже Windows 98 нельзя назвать новой операционной системой, хотя некоторые специалисты запускают PHP под ее управлением.

Если вы планируете использовать язык PHP для создания Web-приложений, вам понадобится применять и язык HTML (HyperText Markup Language). Надеюсь, вы знакомы с этим языком. Поэтому в книге не поясняются многочисленные примеры использования этого языка. Если вы не знакомы с языком HTML, прочтите специализированную книгу, а затем попробуйте создать простую Web-страницу. Если у вас совсем нет терпения, можно приступать к изучению PHP даже без базовых знаний HTML. Из этой книги вы сможете почерпнуть все сведения, необходимые для создания своего Web-приложения. Однако на всякий случай положите рядом книгу по HTML.

Надеюсь, вам уже приходилось создавать хотя бы одну Web-страницу, а может быть, даже статический Web-узел. Тогда вы должны знать, где нужно разместить Web-страницу, чтобы она была доступна пользователям Web.

Книга *не* предполагает знания каких-либо языков программирования. Эта книга рассчитана на новичков, которые хотят научиться писать сценарии на языке PHP. Поэтому, если у вас нет опыта программирования, не печальтесь. В книге вы найдете всю необходимую информацию. Если же вы умеете программировать на другом языке, особенно на C, эта книга поможет вам быстро освоить новый язык программирования.

Структура книги

Материал этой книги делится на шесть частей, в которых изложена вся необходимая информация о PHP, от основ этого языка до его приложений.

Часть I. "Познакомьтесь с языком сценариев PHP"

В этой части приводится обзор возможностей PHP и принципов его работы. Вы узнаете, как настроить среду для использования языка PHP и как на нем написать первый сценарий.

Часть II. "Переменные и данные"

Переменные — основа языка PHP. В этой части речь идет о создании и использовании переменных. Здесь рассказывается о типах данных и способах их обработки. Вы узнаете, как создавать и использовать сложные переменные, получившие название *массивов*.

Часть III. "Основы программирования на PHP"

В этой части показано, как писать сценарии на языке PHP, а также изложены основы объектно-ориентированного программирования на PHP.

Часть IV. "Стандартные PHP-приложения"

В этой части изложены приемы программирования на языке PHP, необходимые при создании типичных приложений. Вы узнаете, как писать сценарии для Web-узла, отображать формы и обрабатывать их данные. Вы также научитесь использовать язык PHP для взаимодействия с базами данных и для выполнения системных задач, в том числе записи файлов на жесткий диск и выполнения команд операционной системы.

Часть V. "Великолепные десятки"

В этой части содержатся полезные рекомендации о том, что следует делать при написании сценариев PHP, а чего делать не нужно. Здесь же приводится список полезных Web-ресурсов, связанных с использованием PHP.

Часть VI. "Приложения"

В этой части приводятся детальные рекомендации по установке PHP. В приложении Б содержится список функций PHP, которые могут оказаться полезными при создании сценариев PHP.

Пиктограммы, используемые в книге

Для удобства работы с книгой в ней содержатся специальные пиктограммы, определяющие различные типы информации.

Эта пиктограмма связана с дополнительной информацией по конкретному вопросу. Советы позволят сэкономить время и силы, поэтому на них стоит обращать особое внимание.

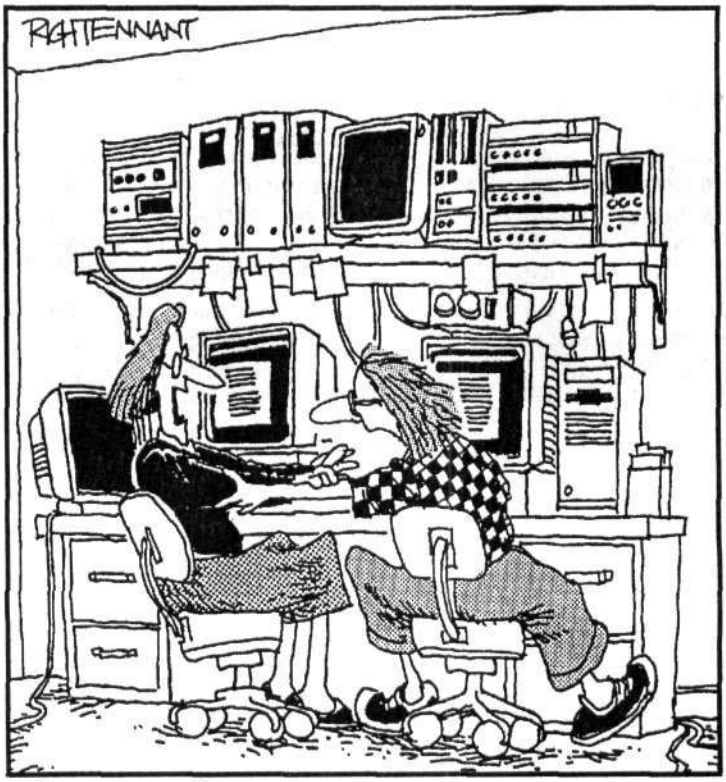
Эта пиктограмма сопровождает информацию, которую желательно запомнить.

Обращайте особое внимание на предупреждения! В них описаны действия, которые нужно выполнить (или, наоборот, которые выполнять не следует) во избежание неприятностей.

Эта пиктограмма связана с технической информацией. Она может быть интересной и полезной, однако не является обязательной для понимания основного содержания книги.

Часть I

Познакомьтесь с языком сценариев PHP



"Ты полный болван, Мартин. Но ты совершенно не мой болван".

В этой части...

В этой части рассматриваются основные возможности PHP. Дается описание этого языка, основных принципов его работы и назначения. После этого описываются настройки PHP в вашем рабочем окружении и рассматриваются различные способы доступа к интерпретатору PHP.

Далее предлагается описание процесса разработки на языке PHP. Рассматривается создание первых сценариев и обсуждается несколько простых примеров.

Глава 1

Знакомство с PHP

В этой главе:

- Обзор особенностей PHP
- Описание работы PHP
- PHP как открытое программное обеспечение

Итак, вы желаете познакомиться с языком PHP. Возможно, это ваш первый опыт в программировании. И вы выбрали PHP по совету своего товарища, который сказал вам, что язык PHP очень простой. Может быть, он и прав, поскольку PHP является одним из самых легких для понимания языков программирования. И разработчики PHP постоянно прилагают все усилия, чтобы этот язык оставался несложным в использовании.

Возможно, вы уже имеете опыт программирования на других языках. И вы решили изучить PHP, поскольку он является наилучшим средством для создания новых Web-приложений. Это решение заслуживает похвалы, так как PHP — это как раз то, что нужно для создания динамических Web-приложений. Начать программировать на PHP очень легко. Более того, PHP предоставляет ряд современных средств, которые будут полезны для опытных программистов. Если вы знакомы с языком программирования C, то вы уже имеете неплохой задел, поскольку PHP имеет аналогичный синтаксис.

В этой главе описывается PHP и рассказывается о том, что этот язык позволяет делать и каким образом.

Особенности языка PHP

PHP является широко используемым открытым универсальным языком разработки сценариев. Первоначально он был создан для разработки Web-узлов. Созданный Расмусом Лердофом (Rasmus Lerdof), PHP фактически использовался как средство для создания пользователями своих Web-страниц (Personal Home Page — персональные домашние страницы). Однако PHP оказался настолько полезным и популярным, что быстро стал полноценным языком программирования. При этом он приобрел новое название — Hypertext Preprocessing, — которое отображает его расширенные возможности по предварительной обработке Web-страниц перед их отображением.

PHP приобрел большую популярность благодаря следующим преимуществам.

- ✓ **Быстродействие Web-узлов**. Поскольку код PHP встраивается в HTML-страницу, время, необходимое для обработки и загрузки Web-страницы, невелико.
- ✓ **Открытость**. PHP является доказательством того, что бесплатные обеды все же существуют, и вы можете получить больше того, за что платите.
- ✓ **Простота использования**. Синтаксис PHP достаточно прост для понимания и использования даже для *не*программистов. При этом PHP разработан таким образом, чтобы быть легко встраиваемым в HTML-страницы.

- ✓ **Универсальность**. PHP можно использовать под управлением разных операционных систем, включая Windows, Linux, Mac OS и большинство систем семейства Unix.
- ✓ **Многосторонняя техническая поддержка**. Вы можете посетить официальный Web-узел PHP (`www.php.net`), где предлагается обширный список тем для обсуждения, покрывающий различные вопросы, такие как основы PHP, PHP под Windows или базы данных и PHP. К тому же перечень тем для обсуждений можно найти также по адресу `news.php.net`.
- ✓ **Безопасность**. Если ваш сценарий PHP разработан правильно, то его программный код пользователи увидеть не смогут.
- ✓ **Настраиваемость**. Открытость PHP позволяет программистам модифицировать программное обеспечение, добавлять или изменять его функциональность, необходимую для решения конкретных задач. PHP обеспечивает достаточный контроль над окружением, позволяя уменьшить вероятность ошибок.

Различные применения PHP

PHP является универсальным средством разработки сценариев общего назначения. Сценарии представляют собой компьютерные файлы, которые содержат написанные на языке PHP инструкции, выполняющие определенные действия, такие как вывод на экран строки `"Привет"` или сохранение некоторой информации в базе данных. Большинство сценариев содержат последовательности инструкций, позволяющих решать задачи, начиная от разработки Web-страниц до навигации по файловой системе. Поскольку PHP создавался для Web, он обладает многими возможностями, которые как раз и предназначены для использования в сценариях создания динамических Web-страниц. Возможно, в данный момент у вас возникнут трудности при использовании PHP для создания Web-страниц, однако его применение для других целей трудно переоценить.

PHP является очень популярным средством для разработки Web-узлов. Согласно информации, размещенной на Web-узле PHP (`www.php.net/usage.php`), около 11 миллионов доменов используют PHP. Даже Web-узел `Yahoo!`, который, вероятно, является самым посещаемым в мире, решил заменить свой собственный язык создания сценариев на PHP.

Использование PHP для Web-приложений

С самого начала Web-страницы были статическими, т.е. они представляли собой простые документы. Пользователи посещали Web-узлы, чтобы прочитать нужную информацию. При этом документы были связаны между собой, так что пользователь мог спокойно найти информацию, которую он искал. Однако Web-страницы оставались неизменными. Каждый пользователь, посещающий определенную Web-страницу, всегда видел одно и то же.

Впоследствии разработчики Web-страниц пожелали большего. Они захотели взаимодействовать с посетителями, получать от них информацию и настраивать Web-страницы под конкретного пользователя. При этом было разработано несколько языков, позволяющих создавать динамические Web-узлы. PHP является одним из самых успешных таких языков, который быстро нашел свое применение и приобрел большую популярность.

PHP является *серверным языком* (server-side language) для написания сценариев. Это означает, что сценарий выполняется на *сервере* (компьютер, на котором установлен Web-узел). В этом состоит отличие от другого популярного языка для создания динамических Web-страниц — JavaScript. Сценарий, написанный на JavaScript, выполняется клиентским браузером. Таким образом, JavaScript — *клиентский язык* (client-side language). Web-серверы и взаимодействие серверов и клиентов описываются в разделе "PHP для Web" далее в этой главе.

Поскольку сценарий PHP выполняется на сервере, он позволяет динамически генерировать HTML-код Web-страницы. Таким образом, каждый пользователь может видеть "свою" Web-страницу. При этом посетитель видит результат выполнения сценария, а не его программный код.

PHP предоставляет ряд возможностей, специально предназначенных для разработки Web-узлов.

- ✓ **Взаимодействие с HTML-формами**. PHP позволяет отображать данные HTML-форм и обрабатывать информацию, введенную пользователем в браузере.
- ✓ **Взаимодействие с базами данных**. PHP позволяет взаимодействовать с базами данных для хранения информации, введенной пользователем, или извлечения информации для ее отображения.
- ✓ **Создание безопасных Web-страниц**. PHP позволяет разработчикам создавать безопасные Web-страницы, требующие ввода достоверных имени пользователя и пароля перед выводом содержимого Web-страницы.

Эти и другие возможности PHP позволяют очень легко создавать динамические Web-узлы.

Как уже отмечалось, PHP — серверный язык, который не может взаимодействовать с пользователем напрямую. Это означает, что невозможно инициировать какие-либо действия, основываясь на состоянии пользовательского компьютера, например, среагировать на щелчок кнопкой мыши или уменьшение размера окна. Другими словами, с помощью PHP невозможно создать такие популярные эффекты, как выпадающее или изменяющее цвет меню. С другой стороны, JavaScript является клиентским языком написания сценариев, который не имеет доступа к серверу. Например, с помощью JavaScript невозможно сохранять информацию на сервере или извлекать ее из серверной базы данных. Но, к счастью, вам ничего не нужно выбирать. Вы спокойно можете использовать PHP и JavaScript вместе для создания таких Web-страниц, которые невозможно разработать, используя только один из этих языков. Вопросам совместного использования PHP и JavaScript посвящена глава 11.

Использование PHP для взаимодействия с базами данных

Существенным преимуществом языка PHP являются его возможности по взаимодействию с базами данных. PHP поддерживает практически все возможные форматы баз данных, о которых вам когда-либо приходилось слышать. PHP устанавливает соединение с базой данных и взаимодействует с ней. При этом вам нет необходимости знать о каких-либо технических деталях такого взаимодействия. Вы указываете имя базы данных и ее местоположение, а сценарий PHP сам выполняет все необходимые действия: устанавливает соединение, выполняет запрос и возвращает результат.

В настоящее время PHP поддерживает большинство известных форматов баз данных:

- ✓ dBASE
- ✓ Informix
- ✓ Ingres
- ✓ Microsoft SQL Server
- ✓ mSQL
- ✓ MySQL
- ✓ Oracle
- ✓ PostgreSQL
- ✓ Sybase

PHP поддерживает также и другие форматы баз данных, такие как filePro, FrontBase и InterBase. К тому же PHP поддерживает открытый интерфейс доступа к базам данных ODBC (Open Database Connectivity), который позволяет взаимодействовать с такими базами данных, как Access и IBM DB2.

PHP идеально подходит для разработки Web-узлов, предназначенных для взаимодействия с базами данных. PHP-сценарий позволяет сохранять и извлекать информацию из любой поддерживаемой базы данных. Кроме того, PHP может взаимодействовать с базами данных вне Web-окружения. Таким образом, работа с базами данных — одно из наилучших свойств PHP.

Использование PHP для взаимодействия с файловой системой

PHP предоставляет все необходимые средства для взаимодействия с файловой системой: с папками и файлами, находящимися на локальном жестком диске или на компьютере, доступ к которому возможен через сеть. PHP позволяет осуществлять запись в файл, считывать их содержимое и создавать новые файлы. PHP также предоставляет возможности по созданию папок, копированию, переименованию, удалению или изменению атрибутов файлов, а также множество других разных операций. Таким образом, язык PHP позволяет выполнять практически все возможные задачи, относящиеся к файловой системе.

Большинство Web-узлов требует взаимодействия с файловой системой. Например, временную информацию Web-приложению целесообразно сохранять в файле, а не в базе данных, и, следовательно, существует необходимость считывать ее из файла.

Поддержка и администрирование любой программной системы также часто требуют взаимодействия с файловой системой. Например, PHP-сценарии можно использовать для создания копий файлов, очистки папок или обработки текстовых файлов для изменения формата их содержимого. PHP позволяет справиться с этими задачами очень легко.

Использование PHP для запуска системных команд

Язык PHP предоставляет необходимые средства для взаимодействия с операционной системой, позволяя выполнять системные команды и получать результаты их выполнения. Например, в сценарии PHP можно использовать команду `dir` или `ls` для получения перечня файлов в папке.

Возможность взаимодействия PHP с операционными системами является особенно полезной при решении задач системной поддержки и администрирования. Например, используя системные команды в сценарии PHP, можно получить список файлов в папке, выбрать, а затем удалить ненужные файлы с определенным расширением.

Возможность выполнения системных команд позволяет также запускать и другие программы в системе, т.е. с помощью сценариев PHP можно выполнять программы, написанные на других языках, и использовать результаты их выполнения. Не стало ли вам легче от того, что не нужно переписывать программы, которые вы используете в настоящее время? Используя PHP-сценарий, вы можете просто выполнять программы, написанные на Perl, C или любом другом языке, или запускать *сценарии оболочек* (shell scripts). Таким образом, PHP позволяет добавить новую функциональность к системным средствам, не тратя дополнительного времени на переписывание уже существующих программ.

Принципы функционирования PHP

PHP является языком программирования высокого уровня. Это означает, что он является дружественным для пользователей и подобен английскому языку. Однако поскольку компьютер не понимает английский язык, то для взаимодействия с ним можно использовать PHP.

При этом интерпретатор PHP преобразует код сценария в язык, понимаемый компьютером, который, в свою очередь, и обеспечивает выполнение всех необходимых действий.

Интерпретатор PHP поставляется в двух различных версиях. Первая из них предназначена для работы с Web-узлами, а вторая — для запуска сценариев из командной строки независимо от Web. При этом вы можете установить как обе версии, так и одну из них.

PHP как универсальный язык

Если PHP используется в качестве универсального языка написания сценариев, необходимо установить PHP CLI. Эта версия PHP как раз и была разработана для этих целей. При этом доступ к интерпретатору PHP осуществляется через командную строку, из которой и происходит запуск сценариев PHP. Такой подход аналогичен принципам использования многих других языков программирования, таких как Perl или C. Вопросы, связанные с запуском сценариев с помощью PHP CLI, более подробно рассматриваются в главе 3.

Как работает World Wide Web

Будет полезным немного рассказать о том, как работает "всемирная паутина", или World Wide Web (WWW). Web представляет собой сеть компьютеров, на которых находятся Web-страницы. Количество таких Web-узлов достигает миллионов. Для того чтобы было легче найти Web-узел во "всемирной паутине", ему присваивается определенный адрес, называемый *URL* (Uniform Resource Locator — универсальный локатор ресурсов). URL Web-узла состоит из имен домена и файла, например `www.mycompany.com/welcome.html`. Если пользователь хочет посетить определенную Web-страницу, ему необходимо набрать нужный URL-адрес в браузере. При этом происходит следующий процесс.

1. Web-браузер посылает сообщение в Web с запросом на получение необходимой страницы.
2. Сообщение передается компьютеру, который находится по URL-адресу, указанному в запросе.
3. Web-сервер, установленный на удаленном компьютере, получает отправленное сообщение.
4. Web-сервер отыскивает необходимый HTML-файл, имя которого указано в URL-адресе запроса.
5. Отыскав запрашиваемый файл, Web-сервер передает его обратно браузеру. (Если запрашиваемого файла не существует, Web-сервер посылает сообщение об ошибке.)
6. Web-браузер отображает на экране HTML-код полученной Web-страницы.

PHP для Web

Если PHP используется для создания Web-узлов, он должен использоваться совместно с Web-сервером. Дело в том, что никакой Web-узел не может существовать без Web-сервера. Web-сервер представляет собой специальное программное обеспечение, предназначенное для отображения Web-страниц во "всемирной паутине". Поэтому PHP не может не взаимодействовать с Web-сервером.

При разработке Web-узлов PHP используется в качестве *встроенного языка написания сценариев* (embedded scripting language). Это означает, что для размещения кода PHP в HTML-файле используются специальные дескрипторы HTML, т.е. HTML-страницы, содержащие код PHP, можно создавать и редактировать таким же образом, как и обычные страницы.

Если модуль PHP установлен на компьютере, то Web-сервер настроен таким образом, чтобы обрабатывать встроенный PHP-код страниц с определенными расширениями. Зачастую таковыми являются страницы с расширением `.php` или `.phtml`, однако Web-сервер можно настроить для обработки файлов с любыми расширениями. Получив запрос на получение файла с определенным расширением, Web-сервер передает обратно HTML-код без каких бы то ни было изменений. В свою очередь, PHP-код сначала обрабатывается модулем PHP, а затем результаты его интерпретации также отправляются обратно клиенту.

При этом результат обработки кода PHP возвращается в формате HTML, а сам код PHP в файл, передаваемый обратно браузеру, не включается. Это существенно повышает безопасность. Кроме того, вся обработка выполняется незаметно для пользователя. Рассмотрим следующий пример кода на PHP:

```
<?php echo "<p>Здравствуй, мир"; ?>
```

В этом выражении дескриптор `<?php` является открывающим, а `?>` — закрывающим. Оператор `echo` позволяет выводить текст, в том числе и как HTML-код. Модуль PHP выполнит обработку этого выражения и выдаст следующий результат:

```
<p>Здравствуй, мир
```

Полученная строка содержит обычный HTML-код, который и отсылается клиентскому браузеру.

Таким образом, очевидно, что PHP и Web-сервер функционируют совместно. Модуль PHP можно интегрировать далеко не с каждым Web-сервером, а только с наиболее популярными из них. Поскольку PHP был разработан в рамках одного из проектов Apache Software Foundation, лучше всего его использовать вместе с Web-сервером Apache. Однако модуль PHP можно использовать и совместно с Microsoft IIS/PWS, iPlanet (в прошлом Netscape Enterprise Server) и другими Web-серверами.

Немного о Web-серверах

Программное обеспечение, которое обеспечивает отправку Web-страниц во "всемирную паутину", называется *Web-сервером* (Web server). В настоящее время существует большое количество Web-серверов, но среди них наиболее популярным является Apache. Согласно результатам исследований, приведенным на узлах `www.netcraft.com` и `www.securityspace.com/s_survey/data/`, около 60% Web-узлов используют Apache. Web-сервер Apache относится программному обеспечению с открытым кодом, т.е. бесплатному, и позволяет взаимодействовать со многими операционными системами. Он автоматически устанавливается вместе с операционной системой Linux и Mac OS X. Более подробную информацию о Web-сервере Apache можно найти по адресу `httpd.apache.org`. Поскольку язык PHP был разработан в рамках одного из проектов Apache Software Foundation, наилучшим образом он работает именно вместе с Web-сервером Apache.

Однако существуют также и другие Web-серверы. Вторым по популярности является Internet Information Server (IIS), который используется на около 30% Web-узлов. Разработанный компанией Microsoft, IIS-сервер работает только под управлением операционной системы Windows. При этом он устанавливается по умолчанию вместе с серверными версиями этой системы. Среди других Web-серверов следует выделить Zeus, NCSA и Sun ONE. Остальные Web-серверы используются не более чем на 2,5% Web-узлов.

Отслеживайте изменения

Язык PHP относится к программному обеспечению с открытым кодом. Если до сих пор *вы* использовали только коммерческое программное обеспечение, разработанное такими компаниями, как Microsoft, Macromedia или Adobe, то увидите, что программное обеспечение с открытым кодом представляет собой нечто совсем иное. Оно обычно разработано различными программистами, которые занимаются разработкой в свое свободное время для удовольствия и абсолютно бесплатно. Не существует никакого корпоративного офиса, с которым можно было бы связаться по каким-либо вопросам, ни менеджера по продажам, который бы убеждал вас во многих преимуществах разработанного программного обеспечения, ни телефона технической поддержки, по которому можно было бы позвонить.

Все это звучит так, как будто не существует никакой поддержки PHP, не правда ли? Однако все обстоит иначе. PHP поддерживается невероятным количеством разработчиков

и пользователей. И необходимо постоянно следить за изменениями, так как это является частью работы как пользователей, так и разработчиков PHP.

Открытое программное обеспечение часто подвергается изменениям, чаще чем один или два раза в год, как в случае с коммерческим. Оно изменяется в тот момент, когда разработчики считают, что для этого пришло время, или вследствие возникающих проблем. Когда обнаруживается серьезная проблема, например изъян в обеспечении безопасности, новая версия, устраняющая ее, может быть выпущена в течение нескольких дней. При этом вы не получите цветных буклетов и не увидите красочную рекламу в цветных журналах перед выходом новой версии. Если не сделать все возможное для получения новой информации, можно пропустить выпуск новой версии или оставаться в неведении о серьезных проблемах, связанных с текущей версией.

Чаще посещайте официальный Web-узел PHP и постоянно знакомьтесь с приведенной там информацией. Подпишитесь на рассылку сообщений, которые содержат достаточно оперативную информацию. Когда вы приступите к использованию PHP, то большое количество ценной информации по различным вопросам можно будет найти в своем почтовом ящике. Впоследствии вы сможете помочь многим другим разработчикам, основываясь на своем собственном опыте. По крайнем мере подпишитесь на рассылку *уведомлений* (announcement mailing lists), которые рассылаются не так часто. Из них можно узнать о серьезных проблемах, найденных в программном обеспечении, или о выходе новых версий. Это именно то, что вам нужно.

Итак, пока вы не забыли, посетите официальный Web-узел PHP и подпишитесь на различные новости по адресу `www.php.net/mailing-lists.php`.

PHP 5

Большинство существенных изменений, внесенных в PHP 5, не касаются общих правил написания кода. В основном они относятся к вопросам, связанным с производительностью PHP. Был существенно улучшен механизм Zend (волшебный и скрытый управляющий механизм), и, как результат, сценарии выполняются теперь намного быстрее и эффективнее.

Большой акцент в PHP 5 сделан на объектно-ориентированном программировании, возможности которого существенно улучшены по сравнению с версией PHP 4. Создание и использование объектов осуществляется намного быстрее. Добавлены многие объектно-ориентированные свойства, включая обработку *исключений* (exceptions). Программисты, предпочитающие объектно-ориентированный подход, должны быть довольны изменениями, внесенными в PHP 5. (Объектно-ориентированное программирование описано в главе 9.)

С выходом PHP 5 изменилось и название интерпретатора PHP. Теперь интерпретатор для Web называется `php-cgi`. PHP CLI называется просто `php`, как `php.exe` под Windows. Оба файла можно найти в папке, в которой установлен модуль PHP. До появления PHP 5 обе программы имели одинаковое имя `php.exe`, но содержались в разных подкаталогах.

В PHP 5 добавлена поддержка формата баз данных MySQL 4.1 и выше. Однако она не включена по умолчанию, поэтому поддержку MySQL 4.0 или MySQL 4.1 нужно активизировать после установки PHP. До версии PHP 5 базы данных MySQL 4.0 и ниже поддерживались автоматически.

В PHP 5 по умолчанию поддерживается расширение SQLite, которое предоставляет быстрые и простые средства для хранения и получения информации из *текстовых файлов* (flat files).

Предыдущие версии PHP

Следует знать о существенных изменениях, которые касаются предыдущих версий PHP. Это связано с тем, что сценарии, написанные для более ранних версий интерпретатора PHP, могут вызывать проблемы при использовании его более поздних версий. Ниже приведены самые существенные изменения.

- ✓ **Версия 4.3.1**. Решены проблемы обеспечения безопасности, которые были обнаружены в версии 4.3.0. Поэтому для поддержки Web-узла не стоит использовать версии PHP 4.3.0 и ниже.
- ✓ **Версия 4.3.0**. Внесены существенные изменения в версию PHP CLI, которые позволяют устанавливать ее по умолчанию при компиляции исходного кода PHP (этот процесс описан в приложении А). Если версия PHP для командной строки вам не требуется, то при установке необходимо отключить соответствующий параметр.
- ✓ **Версия 4.2.0**. Значением переменной `register_globals` по умолчанию является `Off`. Сценарии, написанные с помощью предыдущих версий языка PHP, могут использовать значение `On` и, следовательно, будут работать некорректно. Поэтому целесообразно переписать код сценария так, чтобы он не зависел от значения этой переменной.
- ✓ **Версия 4.1.0**. Введены суперглобальные массивы. Сценарии, использующие суперглобальные переменные (они рассматриваются в главе 6), не будут поддерживаться ранними версиями. До версии PHP 4.1.0 необходимо использовать массивы старого формата, такие как, например, `$HTTP_POST_VARS`.

Возможно, к моменту выхода в свет этой книги уже появятся обновления в PHP версии 5. Однако вряд ли крупные отделы информационных технологий и Web-хостинговые компании сразу же этим воспользуются. Всегда помните о нюансах различных версий используемого программного обеспечения.

Глава 2

Настройка программного окружения

В этой главе...

- Получение доступа к PHP через Web-серверы хостинговых компаний
- Создание личного Web-узла с нуля
- Тестирование PHP-сценариев

Теперь, когда вы решили использовать язык PHP, первая задача состоит в настройке рабочего окружения для разработки Web-приложений. Как уже упоминалось, PHP чаще всего используется для разработки динамических Web-узлов, поэтому данная глава в основном посвящена настройке PHP для дальнейшего использования на Web-узле. Если вы планировали использовать PHP только как язык сценариев общего назначения, независимо от Web, то процесс настройки рабочей среды намного упрощается. Вы можете пропустить разделы, посвященные настройке окружения для работы в Web, и сразу перейти к разделу "Настройка PHP для написания сценариев общего назначения".

Настройка Web-окружения

Как говорилось в главе 1, язык PHP для Web-разработки используется в паре с Web-сервером. Таким образом, для работы Web-узла требуется Web-сервер. При использовании PHP на вашем узле Web-сервер должен поддерживать обмен информацией со сценариями PHP. Значит, PHP должен быть установлен на сервере, который удовлетворяет данному требованию. Окружение узла включает в себя не только Web-сервер и модуль PHP. Приведем несколько других требований.

- ✓ Компьютер должен быть подключен к Internet.
- ✓ Компьютер должен иметь достаточно ресурсов, таких как дисковое пространство и память, для управления ожидаемым Web-трафиком.
- ✓ Для работы сценария PHP может понадобиться другое программное обеспечение, например для управления базами данных.

Вы можете интересоваться или не интересоваться установкой вашего личного окружения, вы можете считать, что его настройка — это развлечение, или можете думать, что все программы уже установлены и функционируют. Если вы хотите установить личное Web-окружение с нуля, то это можно осуществить. Возможно, даже на вашем компьютере уже работает Web-узел, а вам нужно только обеспечить расширение его функциональности с использованием языка PHP. Если вы не хотите устанавливать личное Web-окружение, то можете использовать окружение, установленное и поддерживаемое кем-то другим, например информационным отделом вашей компании или коммерческими компаниями, которые предоставляют услуги Web-хостинга. Возможно, вы уже имеете Web-узел, который хотели бы сделать более динамическим. Язык PHP можно использовать во всех случаях, независимо от того, сами вы создаете Web-окружение или оно предоставлено кем-то другим.

Другой вариант типичной среды разработки включает как вашу личную Web-среду, так и поддерживаемую кем-то другим (зачастую разработчики организуют тестирование Web-

среды на своих личных компьютерах, на которых они создают и отлаживают Web-страницы). Потом, когда все будет работать корректно, Web-страницы пересылаются на корпоративный Web-узел, который поддерживается информационным отделом или компанией, предоставляющей услуги Web-хостинга.

Приведем некоторые преимущества использования готового Web-окружения.

- ✓ **Это легче, чем настроить свое собственное окружение.** Вы только копируете Web-страницы на другой компьютер. Вам не нужно устанавливать другое программное обеспечение, компьютерное оборудование или решать компьютерные проблемы. Кто-то другой сделает все это за вас.
- ✓ **Требуется меньше технических навыков работы.** Вам необходимо знать лишь языки программирования в Web, такие как HTML и PHP. При этом не нужно ничего знать о соединениях Internet, Web-серверах, администрировании компьютеров и других технических вопросах. Это интересно некоторым людям, однако далеко не всем.

Преимущества использования собственной Web-среды состоят в следующем.

- ✓ **Управляемость.** Вы принимаете все решения самостоятельно и можете настроить Web-среду по своему усмотрению.
- ✓ **Доступ.** Вы имеете доступ к компьютеру всегда, когда захотите работать с вашим узлом.
- ✓ **Стабильность.** Вы знаете, что узел будет работать столько, сколько вам нужно. Вы полностью защищены от ситуации, когда однажды проснувшись, узнаете, что компания, которая предоставляла вам услуги хостинга, отошла от дел, и вам необходимо за два дня переместить ваш узел куда-нибудь в другое место.
- ✓ **Безопасность.** Поскольку вы контролируете Web-среду, то только вы должны иметь доступ к компьютеру. По мере необходимости его можно заблокировать. Если вы пользуетесь услугами хостинговых компаний, то к компьютеру имеют доступ и другие люди. Один из них может быть тем плохим парнем, который завладеет вашими секретами.

Использование существующей Web-среды

Если вы используете Web-среду, настроенную кем-то другим, вам не обязательно нужно понимать все нюансы процесса установки и администрирования программного обеспечения Web-узла. Кто-то другой — информационный отдел вашей компании, коммерческие хостинговые компании или ближайший сосед — отвечает за функционирование узла. Их обязанность — предоставить вам возможность работать с Web-узлом, включая PHP. Ваше задача — лишь написать и установить файлы Web-узла.

Для использования существующей Web-среды вам понадобится следующая информация от администратора Web-узла.

- ✓ **Местоположение Web-страниц.** Для того чтобы узел был доступен для всего мира, Web-страницы должны располагаться в особом месте на компьютере. Web-сервер, который доставляет Web-страницы во внешний мир, рассчитывает найти файлы в специальном каталоге. Вам необходимо знать, где находится этот каталог, и иметь к нему доступ.
- ✓ **Процесс доставки Web-страниц**. Необходимо знать, как установить файлы. В большинстве случаев вы отправляете файлы с использованием протокола FTP в надлежащее место. Протокол передачи файлов FTP (File Transfer Protocol) — это метод копирования файлов с одного компьютера на другой по сети. В некоторых

случаях файлы можно копировать непосредственно или использовать другие методы доставки Web-страниц. Для установки файлов нужно иметь *идентификатор пользователя* (user ID) и пароль.

- ✓ **Имя файла, используемое по умолчанию.** Когда пользователи вводят в окне браузера адрес URL, им передается некоторый файл. В настройках Web-сервера указывается имя файла, отсылаемого по умолчанию, если URL-адрес указывает на каталог. Зачастую этот файл называется `index.htm` или `index.html`, но иногда он имеет имя `default.htm`. Необходимо знать, как назвать свой файл, передаваемый по умолчанию.
- ✓ **Расширение файлов PHP.** После установки модуля PHP Web-сервер ожидает встретить операторы PHP в файлах со специальным расширением. Часто таким расширением является `.php` или `.phtml`, но могут использоваться и другие расширения. Операторы PHP в файлах с некорректным расширением не будут выполняться. Необходимо знать, какое расширение нужно использовать для сценариев PHP.

Одним из неудобств хостинга узла в существующей Web-среде является отсутствие контроля над средой разработки. Администраторы предоставляют Web-среду, которая, по их мнению, работает лучше всего. Например, модуль PHP имеет бесчисленное количество настроек, которые можно установить, отменить или инициализировать разными значениями. Администраторы, задавая значения параметров, руководствуются своими нуждами, которые могут быть оказаться не столь подходящими для ваших целей. Возможно, такая настройка окружения обоснована легкостью ее обслуживания, низкой стоимостью и минимальными неудобствами клиента. Вы не сможете изменить некоторые параметры среды, а можете только просить, чтобы администраторы изменили их. Они будут неохотно изменять рабочие параметры, поскольку подобная модификация может привести к проблемам с системой или с другими клиентами.

Выбор хостинговой компании

Хостинговые компании предоставляют все, что нужно для работы Web-узла, включая дисковое пространство и все программное обеспечение узла. Вы только создаете файлы своих Web-страниц и перемещаете их в местоположение, указанное хостинговой компанией.

Услуги хостинга предоставляет огромное количество компаний. При этом обычно взимается ежемесячная плата, которая часто является очень маленькой или совсем отсутствует. Компании, предоставляющие услуги бесплатно, требуют размещения их рекламы на страницах своих клиентов. Конечно, месячная плата изменяется в зависимости от ресурсов, выделенных для вашего Web-узла. Например, плата за узел, который занимает 2 Мбайт дискового пространства, как правило, меньше, чем за узел размером 10 Мбайт.

Убедитесь в том, что хостинговая компания позволяет использовать сценарии PHP (некоторые этого не делают). Также убедитесь, что компания предлагает последнюю версию PHP. Хостинговая компания может не поддерживать последнюю версию, которая только выпущена, но обязана обновить ее незадолго после выхода новой версии.

Не пользуйтесь услугами компаний, которые предлагают поддержку лишь PHP 3. PHP 4.3.1 был выпущен в феврале 2003 года, поэтому ни одна хостинговая компания не должна предоставлять PHP более ранних версий, поскольку в PHP 4.3.1 были устранены проблемы с безопасностью, обнаруженные в более ранних версиях. Остается надеяться, что к моменту выхода в свет этой книги большинство компаний будет предлагать хостинг PHP 5.

Среди других мотивов выбора хостинговой компании следует отметить следующие.

- ✓ **Надежность.** Вам необходима компания, которой можно доверять и которая завтра не обанкротится и не исчезнет. Выберите компанию, которая имеет необходимые вычислительные мощности и другие ресурсы для поддержки вашего узла. Узел, который больше простаивает, чем работает, абсолютно бесполезен.
- ✓ **Скорость.** Страницы, которые медленно загружаются, раздражают пользователей и вынуждают их посетить другие Web-страницы. Медленная загрузка страниц может быть результатом того, что хостинговая компания малобюджетна и не имеет хорошего оборудования. В любом случае компании, пересылающие Web-страницы слишком медленно, не могут представлять особого интереса. Поэтому при выборе хостинговой компании следует поинтересоваться скоростью загрузки страниц, которую она обеспечивает. Иногда такая информация содержится на узле хостинговой компании, в противном случае ее можно получить у менеджера.
- ✓ **Техническая поддержка.** Некоторые хостинговые компании не имеют специалистов, готовых отвечать на вопросы или решать проблемы клиентов. Техническая поддержка часто обеспечивается только через рассылку по электронной почте. Такой способ может быть приемлем только в том случае, если ответ предоставляется в кратчайший срок. Иногда качество услуг компании можно проверить, позвонив по телефону технической поддержки или засекая время ответа по электронной почте.
- ✓ **Доменное имя.** Каждый узел имеет свое доменное имя, с помощью которого Web-браузеры находят его в сети. Каждое имя домена регистрируется за небольшую ежегодную плату, так что пользоваться им может только один узел. Одни хостинговые компании разрешают независимо использовать зарегистрированное вами доменное имя; другие помогают с регистрацией и использованием нового имени домена, а третьи требуют использовать их доменное имя. Например, допустим, что ваша компания называется Good Stuff и вы хотите, чтобы узел назывался JanetsGoodStuff. Одни компании разрешат назвать ваш узел `JanetsGoodStuff.com`, а другие потребуют, чтобы узел имел имя `JanetsGoodStuff.Web-hostingcompanyname.com`, или `JanetsGoodStuff.Web-hostingcompanyname.com/~GoodStuff`, или что-то в этом роде. Вообще-то, узел будет выглядеть более профессионально, если вы будете использовать собственное доменное имя.
- ✓ **Свойства.** Свойства следует выбирать в зависимости от назначения узла. Обычно хостинговая компания предлагает определенные возможности в пакетах или планах. Но чем больше возможностей, тем выше цена. Некоторые свойства, которые следует рассмотреть, приведены ниже.
  - **Дисковое пространство.** Как много (Мбайт/Гбайт) места на диске займет ваш узел? Медиа-файлы (графика и музыка) могут занимать действительно много дискового пространства.
  - **Передача данных.** Некоторые хостинговые компании взимают плату за отправку Web-страниц пользователям. Если вы рассчитываете на интенсивный трафик, то эти затраты следует принять во внимание.
  - **Адреса электронной почты.** Многие компании обеспечивают своих клиентов некоторым количеством адресов электронной почты. Например, если ваш узел называется `JanetGoodStuff.com`, вы можете позволить своим посетителям писать по адресу `me@JanetGoodStuff.com`.

- **Программное обеспечение.** Хостиговые компании предлагают доступ к множеству программных продуктов для разработки Web-приложений. Помимо PHP, который является предметом рассмотрения этой книги, некоторые компании предлагают базы данных, такие как MySQL или PostgreSQL, другие средства Web-разработки, такие как FrontPage, программы разработки электронных магазинов, проверки правильности данных кредитных карточек и т.д.
- **Статистика.** Часто компании помогают своим клиентам собрать статистику о Web-трафике на вашем узле, сведения о количестве пользователей, времени доступа к Web-странице и т.д.

✓ **Резервное копирование**. *Резервные копии* (backup) — это копии файлов ваших Web-страниц и баз данных, которые хранятся на случай потери или повреждения файлов или базы данных. Нужно удостовериться в том, что компания регулярно и часто делает резервные копии приложений своих клиентов. При этом следует также поинтересоваться, как быстро ваш узел перейдет из резервных копий в рабочий режим при возникновении какой-либо проблемы.

Хостинговые компании сложно исследовать с нуля — поиск в Google на запрос "web-hosting" выдает около 4 миллионов результатов. Наилучший путь для выбора хостинговой компании — попросить рекомендации у людей, имеющих опыт работы с этими компаниями. Люди, которые пользовались услугами хостинговых компаний, могут предостеречь вас от услуг компаний с медленным сервисом или неустойчивой работой компьютеров. Собрав сведения о нескольких хостинговых компаниях, услугами которых довольны их клиенты, вы можете сузить этот список до одной, которая вам больше подходит.

Игра "доменное имя"

Каждому узлу необходим уникальный адрес в сети. Он используется компьютерами для нахождения узла и называется *IP-адресом*. Это последовательность из четырех чисел от 0 до 255, разделенных точками, например `172.12.204.2`.

Так как IP-адрес — это комбинация цифр и точек, их тяжело запомнить. К счастью, большинство IP-адресов имеет соответствующие имена, которые намного легче запомнить, например `amazon.com`, `www.irc.gv` или `mycompany.com`. Имя, которое соответствует адресу узла, называется *доменным именем* (domain name). Доменом может быть как один компьютер, так и группа компьютеров, связанных между собой. Если домен состоит из нескольких компьютеров, то каждый из них может иметь свое имя. Имя, которое включает в себя собственное имя компьютера, такое как `thor.mycompany.com`, называется *поддоменом* (subdomain) домена `mycompany.com`.

Имя домена или поддомена является неотъемлемой частью URL-адреса, который пользователь вводит в окне браузера для определения Web-узла, который он хочет посетить. URL-адрес может содержать больше составляющих, нежели просто имя домена, но зачастую имя домена (например, `amazon.com`) — это все, что нужно. Иногда достаточно имени поддомена (`janet.valade.com`). Если используется только имя домена, то Web-сервер передает файл с именем по умолчанию (`index.htm` или `index.html`). Иногда в адрес кроме имени домена включают также имя файла, например `janet.valed.com/links.html`.

Для того чтобы имя домена могло служить адресом, оно должно быть уникальным в Internet. Тогда система регистрации доменных имен сможет удостовериться в том, что нет двух узлов с одинаковыми именами. Каждый клиент может зарегистрировать любое имя домена, если это имя еще никем не используется. При регистрации имени домена вы проверяете его на наличие в сети. Если все в порядке (имя уникально), вы регистрируете его на свое имя или на имя компании и платите взнос. Обычная плата за регистрацию доменного имени составляет 35 долларов в год. Обычно больше платить не приходится, а вот поторговаться можно.

Доменное имя можно зарегистрировать на многих Web-узлах, включая множество хостинговых компаний. Поиск в Google по запросу "регистрация доменного имени" дает около 2 миллионов результатов. Удостоверьтесь, что вы нашли самую низкую цену. Многие узлы позволяют по доменному имени узнать, за кем оно закреплено. Эти Web-узлы используют поиск в базе данных доменных имен с помощью команды `whois`. Такой поиск можно организовать на узлах Allwhois (`Allwhois.com`) и Better-Whois (`betterwhois.com`).

Об именах можно спросить у коллег или друзей. На PHP-форумах посетители часто задают вопросы о хостинговых компаниях. Многие посетители форумов имеют богатый опыт использования PHP с хостинговыми компаниями и будут рады вам помочь. Поскольку подобные вопросы возникают очень часто, то соответствующую информацию можно поискать и в архивах, расположенных по адресу `marc.theaimsgroup.com/`.

Настройка собственного Web-окружения

Если вы начинаете разработку Web-узла с нуля, нужно хорошо понимать структуру программного обеспечения. Необходимо принять несколько решений относительно аппаратного и программного обеспечения. Потребуется установить Web-сервер и модуль PHP, а затем самостоятельно поддерживать, администрировать и обновлять систему. Такой путь требует больше усилий и знаний.

Приведем основные действия по настройке Web-окружения, необходимой для работы программ, описанных в этой книге.

1. **Настройте компьютер.**
2. **Установите Web-сервер.**
3. **Установите модуль PHP.**

Первый шаг здесь мы рассматривать не будем. Наверняка у вас есть компьютер, но вполне возможно, что для развертывания своего Web-узла вы планируете использовать (или приобрести) другой компьютер. Более подробную информацию относительно компьютеров можно найти в книге *ПК для "чайников"*, 9-е издание, которая вышла в издательстве "Диалектика". Web-серверы и модули PHP существуют почти для всех типов аппаратных средств и операционных систем, включая множество версий Unix и Linux, Windows и MacOS.

Установка Web-сервера

Если компьютер уже куплен и настроен, следует решить, какой Web-сервер устанавливать. Наилучший выбор — это, конечно же, сервер Apache, поскольку он имеет ряд следующих преимуществ.

- ✓ **Бесплатный.** Что еще можно добавить к этому?
- ✓ **Работает с разными операционными системами.** Сервер Apache работает под управлением систем Windows, Linux, MacOS, FreeBSD и большинства версий Unix.
- ✓ **Популярность.** Согласно исследованиям, результаты которых приведены по адресу `www.netcraft.com/survey` и `www.securityspace.com/s_survey/data/`, на 60% узлов используется Web-сервер Apache. И это действительно так. Это также означает, что при возникновении проблем вам сможет помочь большое количество пользователей.

- ✓ **Надежность.** После установки и настройки сервер Apache будет работать столько, сколько проработает ваш компьютер. Проблемы с этим сервером возникают очень редко.
- ✓ **Возможность настройки.** Открытый исходный код дает возможность программистам модифицировать сервер Apache, добавляя или модифицируя модули по своему усмотрению.
- ✓ **Безопасность.** Существует бесплатное программное обеспечение, которое позволяет настроить Apache для работы с сервером безопасности SSL. Протокол SSL применяется для обеспечения повышенной безопасности Web-узлов, использующих очень важную информацию. Это значит, что информация, передаваемая между Web-сервером и браузером, шифруется, поэтому никто не сможет перехватить и прочитать ее. Безопасность — необходимый компонент для узлов электронной коммерции.

Сервер Apache автоматически устанавливается при установке полной версии системы Linux. Кроме того, Apache обычно входит в состав операционной системы Mac. Для большинства версий Unix можно самостоятельно скачать ядро Apache и скомпилировать его, а можно использовать *двоичные файлы* (binary) (скомпилированные программы для определенных операционных систем). Если вы работаете с операционной системой Windows, вам необходимо установить бинарный файл (в основном Apache предназначен для работы с операционными системами Windows NT/2000/XP, хотя иногда его используют и в системах Windows 98/Me).

Как уже говорилось, наиболее стабильной текущей версией является Apache 1.3.27. Apache 2 — это также стабильная версия, но до сих пор модуль PHP работает с ней в экспериментальном режиме. Однако в любом случае обратитесь на узел `www.php.net` и проверьте текущее состояние дел по этому вопросу. Информацию о сервере Apache, загрузочные модули, документацию, рекомендации по установке в различных операционных системах можно найти по адресу `httpd.apache.org`.

Для работы с языком PHP вполне подойдут и другие Web-серверы. Компания Microsoft предлагает сервер IIS (Internet Information Server), который занимает второе место по популярности в сети Internet: на нем работает около 27% Web-узлов. Компания Sun предлагает сервер iPlanet (раньше он назывался Netscape Enterprise Server), который обслуживает менее 5% узлов глобальной сети.

Установка модуля PHP

Многие компьютерные системы поставляются вместе с модулем PHP. Он включен в состав большинства дистрибутивов Linux. Некоторые более новые версии операционной системы Mac OS X тоже поставляются вместе с модулем PHP. Перед тем как установить этот модуль, проверьте, не был ли он установлен на вашем компьютере ранее. Для этого поищите на диске PHP-файлы. В различных операционных системах это можно сделать следующим образом.

- ✓ **Linux/Unix/Mac.** В командной строке введите команду

  ```
  find / -name "php*"
  ```
- ✓ **Windows.** Воспользуйтесь средством поиска (выберите команду Пуск⇨Поиск) для нахождения файлов по шаблону `php*`.

Если вы не нашли PHP-файлов, значит, модуль PHP отсутствует. Для его установки потребуется доступ к Web-серверу вашего узла. Например, если вы устанавливаете PHP для работы с сервером Apache, нужно отредактировать его конфигурационный файл. Всю необходимую информацию и требуемые программные модули можно найти на узле `www.php.net`. Подробные указания по установке приведены в приложении А.

Если же PHP-файлы нашлись, значит, модуль PHP уже установлен и переустанавливать его не нужно. Чтобы определить, нужно ли переустанавливать модуль PHP, воспользуйтесь следующими рекомендациями.

- ✓ **Параметры установки**. Модуль PHP может быть установлен не с теми параметрами, которые нужны именно вам. Например, он может быть установлен без поддержки базы данных, которую вы планировали использовать. Обычно при установке модуля PHP по умолчанию включен режим поддержки ODBC, но параметры поддержки MySQL, Oracle, MS SQL и других баз данных следует задавать в явной форме. Аналогично, при установке PHP всегда обеспечивается поддержка SQLite, XML, COM, FTP и т.д., но поддержка других форматов по умолчанию отключена. Если вы планируете использовать другую базу данных или программное обеспечение, то, видимо, следует переустановить модуль PHP с поддержкой дополнительных возможностей.

  Можно проверить, какие настройки использовались при установке модуля PHP. Для этого следуйте рекомендациям по тестированию, приведенным в следующем разделе. Если тестовый сценарий работает корректно, то в таблице, которую выведет функция `phpinfo()`, будут приведены все режимы, которые поддерживаются вашей версией PHP. Проверьте, включена ли поддержка нужных вам расширений. Если нет, следует переустановить модуль PHP. Подробные указания по установке приведены в приложении А.

- ✓ **Версия.** Установленная версия может оказаться не самой новой. Следует проверить номер установленной версии модуля PHP. Сделать это можно с помощью команды

  ```
  php-cgi -v
  ```

  Для версий, более ранних, чем PHP5, эта команда имеет следующий вид:

  ```
  php -v
  ```

  Чтобы использовать одну из этих команд, следует перейти в каталог, содержащий файл `php-cgi.exe` (или `php.exe`). В результате вы увидите следующую информацию:

  ```
  PHP 5.0.0 (cgi-fcgi), Copyright (c) 1997-2003 The PHP Group
  Zend Engine v2.0.0, Copyright (c) 1998-2003 Zend Technologies
  ```

  Если установленная ранее версия оказалась не самой новой, модуль PHP следует переустановить. Более подробную информацию о его последней версии можно найти по адресу `www.php.net/downloads.php`.

Тестирование PHP

После того как получена вся необходимая информация, следует протестировать модуль PHP и убедиться в корректности его работы.

1. **Найдите папку, в которой будут размещаться сценарии PHP.**

   Именно в этой папке и ее подпапках будут храниться файлы вашего узла. Обычно при использовании сервера Apache эта папка по умолчанию называется `htdocs`. Она содержится в папке, в которой установлен Web-сервер. Для сервера IIS — это папка `Inetpub\wwwroot`. В системе Linux это может быть каталог `/var/www/html`. При установке Web-сервера можно указать любое имя папки, поэтому если Web-сервер устанавливал кто-то другой, необходимо узнать имя папки, используемой для хранения файлов PHP. Если вы пользуетесь услугами хостинговой компании, она должна предоставить вам имя этой папки.

2. **Создайте файл с именем** `test.php` **и добавьте в него следующий код:**

```
<html>
<head>
<title>Тестирование PHP</title>
</head>
<body>
<p>Это строка HTML
<?php
    echo "<p>Это строка PHP</p>";
    phpinfo();
?>
</body>
</html>
```

3. **Укажите в браузере URL-адрес файла** `test.php`, **который был создан при выполнении п. 2.**

   URL будет иметь вид `http://www.mycompany.com/test.php`. Если же Web-сервер, модуль PHP и сам файл находятся на локальном компьютере, можно ввести URL-адрес `localhost/test.php`.

Чтобы файл обрабатывался интерпретатором PHP, нужно получить доступ к нему через Web-сервер, а не с помощью команды меню браузера Файл⇨Открыть.

В результате в окне браузера вы увидите следующее:

```
Это строка HTML
Это строка PHP
```

Под этими строками должна появиться большая таблица, отображающая всю информацию о PHP в вашей системе. В ней будут содержаться пути и имена файлов, значения переменных, поддерживаемое программное обеспечение и значения различных параметров.

Эта таблица создается функцией сценария `phpinfo()`. Если у вас возникнут вопросы по поводу настройки модуля PHP, всегда можно воспользоваться этой функцией, чтобы отобразить таблицу и проверить установки. Функция `phpinfo()` будет часто использоваться в этой книге.

Если с установкой модуля PHP возникли какие-либо проблемы, то в результате выполнения тестового сценария можно получить следующую информацию.

- ✓ Выводится только текст `Это строка HTML`. Строки PHP и таблица не отображаются.
- ✓ Отображается пустая страница.
- ✓ Браузер не отображает Web-страницу.

Если при запуске тестового файла возникли проблемы, а вы не являетесь системным администратором, то нужно поговорить с тем, кто устанавливал и поддерживает программное обеспечение. Это их работа — диагностировать и решать подобные проблемы.

Если же вы системный администратор и у вас возникла проблема с тестовым файлом, то сначала проверьте, установлен ли модуль PHP. В командной строке перейдите в каталог, в котором установлен модуль PHP, и введите команду

```
php-cgi -v
```

или

```
php -v
```

Если интерпретатор PHP возвращает информацию о своей версии, значит, он установлен. Убедитесь в том, что вы обращаетесь к тестовому файлу именно так, как описано в п. 3.

Удостоверьтесь, что тестовый файл находится в вашем дисковом пространстве, как описано в п. 1. При использовании сервера Apache можно просмотреть файл `httpd.conf` и проверить правильность строки

```
DocumentRoot "C:/Program Files/Apache Group/Apache/htdocs"
```

Эта строка указывает серверу Apache, где искать файлы Web-страниц.

Еще раз убедитесь в правильности кода. Тестовый сценарий можно загрузить с Web-узла автора этой книги — `janet.valade.com`.

Если же сценарий написан правильно и пути заданы корректно, то проблема, скорее всего, в конфигурационных параметрах модуля PHP. Просмотрите инструкции по настройке модуля PHP в конце приложения А и удостоверьтесь в том, что вы следовали всем указаниям. Особенно проверьте следующее.

- ✓ Web-сервер настроен так, что по расширению определяет файлы с кодом PHP. Проверьте, что файл `httpd.conf` содержит строку

  ```
  AddType application/x-httpd-php .php
  ```

- ✓ Эта строка указывает серверу Apache, что код PHP следует искать в файлах с расширением `.php`. Для сервера IIS нужно получить доступ к консоли, как описано в приложении А, и проверить список расширений.
- ✓ Проверьте корректность других строк в файле `httpd.conf`, как описано в рекомендациях в приложении А. Удостоверьтесь также, что необходимые строки содержатся в нужных местах.
- ✓ Если вы используете сервер IIS, найдите в файле `php.ini` следующую строку:

  ```
  cgi.force_redirect = 0
  ```

- ✓ Если в файле `php.ini` нет такой строки — добавьте ее. Если она есть, но перед ней стоит точка с запятой, удалите этот символ. Если в этой строке указано значение `1` — замените его на `0`.

Если вы все тщательно проверили, но проблема не исчезла, возможно, у вас что-то необычное с настройками компьютера или Web-сервера. Прочтите всю информацию о компьютере, которая имеет отношение к установке модуля PHP, а также о проблемах, которые могут возникать при его установке.

Если проблема по-прежнему существует, отправьте свой вопрос в одну из групп новостей. Сначала просмотрите архивы по адресу `marc.theaimsgroup.com/`. Возможно, кто-то уже задавал этот вопрос, и вы сможете быстро найти ответ в архивах. Если нет — отправьте запрос на форум. Включите в свой вопрос следующую информацию.

- ✓ Укажите название и версию операционной системы, которую вы используете.
- ✓ Укажите версию PHP, которую вы пытаетесь установить.
- ✓ Скопируйте содержимое тестового файла в письмо.
- ✓ Опишите точный результат, который вы видите на Web-странице.

Как правило, участники форума являются очень эрудированными специалистами. Они наверняка помогут решить все проблемы.

Настройка PHP для написания сценариев общего назначения

PHP можно использовать как самостоятельный язык программирования общего назначения. В этом случае устанавливать Web-сервер не нужно. Версия PHP для командной строки — PHP CLI — является отдельной программой, которая не используется для поддержки Web-узлов. Ее необходимо установить отдельно.

Даже если на вашей машине уже установлен модуль PHP, версия PHP CLI на ней может отсутствовать. Проверьте, установлен ли на вашем компьютере модуль PHP и какой версии. Поищите соответствующий файл в каталоге PHP. Исполняемый файл PHP CLI называется `php.exe`, а PHP CGI — `php-cgi.exe`. (В версиях до PHP 5 оба файла назывались `php.exe`, которые хранились в различных подкаталогах. PHP CLI размещался в подкаталоге `/cli`.) Возможно, версия PHP CLI была установлена в каком-нибудь другом месте. Все файлы PHP на жестком диске можно найти с помощью следующей команды.

- ✓ **Linux/Unix/Mac.** Введите в командной строке следующую команду:
  `find / -name "php*"`
- ✓ **Windows.** Используйте средство поиска (выберите команду Пуск⇨Поиск) и в качестве шаблона поиска введите строку `php*`.

Если вы нашли файлы PHP, которые, по-вашему мнению, могут относиться к PHP CLI, перейдите в каталог, где находятся эти файлы, и введите следующую команду:

```
php -v
```

На экране появится следующая строка, содержащая фрагмент `cgi` или `cli`:

```
PHP 5.0.0 (cli) (built: Jun 15, 2003 23:07:34)
```

Обратите внимание на то, что в данном случае в строке содержится фрагмент `(cli)`. При проверке версии интерпретатора для Web, в результирующей строке будет содержаться фрагмент `(cgi)`. Указанная команда может также использоваться для проверки корректности работы программы PHP CLI. Если в результате выводится номер версии, а не сообщение об ошибке, значит, программа работает правильно.

Если вы не нашли версию PHP CLI, ее нужно установить. Только после этого вы сможете использовать PHP для решения задач, не связанных с Web. В приложении А детально описана процедура установки PHP, включая инструкции и для установки PHP CLI.

Если вы собираетесь использовать PHP как для разработки Web-узлов, так и для создания сценариев общего назначения, необходимо установить две различные PHP-программы — версию для Web и версию PHP CLI. Обе программы должны быть одинаковой версии, т.е. если вы устанавливаете PHP 5.0.0 для Web, убедитесь, что вы также используете PHP CLI 5.0.0. В системе Windows для работы PHP требуется файл `php5ts.dll`, который находится в главном каталоге PHP. Необходимо использовать одинаковые версии PHP, чтобы обе PHP-программы могли работать с одним файлом `php5ts.dll`. (Более детальная информация по этому вопросу содержится в приложении А.)

Настройка PHP

Модуль PHP характеризуется чрезвычайной гибкостью. Поведение интерпретатора PHP определяется конфигурационными параметрами. Например, именно эти параметры определяют режим отображения сообщений об ошибках. Конфигурационные параметры хра-

нятся в файле php.ini. Редактируя этот файл, можно вносить изменения в конфигурационные параметры.

Как описывается в приложении А, при установке модуля PHP создается файл php.ini. Если вы устанавливаете PHP самостоятельно, запомните местоположение этого файла. Возможно, позже вам понадобится его изменить. Если вы только используете модуль PHP, а кто-то другой является его администратором (например, если вы пользуетесь услугами хостинговой компании), вы вряд ли будете иметь доступ к файлу php.ini. Если вам нужно изменить параметры модуля PHP, необходимо попросить об этом администратора. В некоторых случаях можно добавить соответствующие команды в свой сценарий, чтобы временно изменить настройки только для этого сценария. Определенные операторы, которые временно изменяют настройки PHP, обсуждаются в различных главах этой книги.

Использование специальных средств создания PHP-сценариев

PHP-сценарии — это простые текстовые файлы. Значит, при их создании можно пользоваться любыми известными средствами работы с текстовыми файлами. Множество сценариев написано с использованием редакторов vi, Notepad или WordPad. Кроме того, существуют средства, которые существенно облегчают весь процесс разработки.

Заслуживают внимания редакторы для создания программ и интегрированные среды разработки IDE (Integrated Development Environment). Эти средства предоставляют возможности, которые могут уберечь вас от огромных затрат времени в процессе разработки. Загрузите некоторые демонстрационные версии программ, испытайте их и выберите наиболее подходящую. Время, потраченное на поиск и выбор редактора, вы с лихвой наверстаете впоследствии.

Редакторы для написания программ

Такие редакторы предоставляют богатые возможности для написания программ. Большинство подобных программных средств предоставляют следующие возможности.

- ✓ **Выделение цветом.** Цветовое выделение частей сценария, таких как дескрипторы HTML, строки текста, ключевые слова и комментарии, позволяет легко ориентироваться в коде программы.
- ✓ **Структурированное расположение текста.** Автоматический отступ внутри скобок облегчают чтение сценариев.
- ✓ **Нумерация строк.** Добавление временной нумерации строк упрощает работу над ошибками, поскольку в сообщениях об ошибках указывается номер строки сценария PHP. Самостоятельно сложно отсчитать 872 строки от начала файла и найти ту, в которой была обнаружена ошибка.
- ✓ **Работа с несколькими файлами.** Можно работать с несколькими файлами одновременно.
- ✓ **Простота вставки кода.** Многие редакторы содержат кнопки для вставки фрагментов кода, таких как дескрипторы HTML или операторы и функции PHP.
- ✓ **Библиотека фрагментов кода.** Фрагменты своего кода можно сохранить, а затем вставить в сценарий одним щелчком мышью.

В Internet содержится много бесплатных или недорогих редакторов. Среди наиболее популярных можно выделить следующие.

- ✓ **Arachnophilia** (`www.arachnoid.com/arachnophilia/`). Этот мультиплатформенный редактор написан на языке Java и относится к бесплатному программному обеспечению.
- ✓ **BBEdit** (`www.barebones.com/products/bbedit/index.shtml`). Этот редактор предназначен для использования на компьютерах Mac. Редактор BBEdit стоит 179 долларов. Существует бесплатная версия этого редактора BBEdit Lite. Его развитие и поддержка были прекращены, однако редактор все еще можно найти и легально использовать. В качестве замены BBEdit Lite предлагается редактор TextWrangler, который стоит всего 49 долларов.
- ✓ **EditPlus** (`www.editplus.com`). Этот редактор разработан для использования на машинах под управлением операционной системы Windows. EditPlus — условно-бесплатная программа, лицензия на использование которой стоит 30 долларов.
- ✓ **Emacs** (`www.gnu.org/software/emacs/emacs.html`). Редактор Emacs работает под управлением систем Windows, Linux и Unix и относится к бесплатному программному обеспечению.
- ✓ **HomeSite** (`www.macromdia.com/software/homesite/`). Редактор HomeSite предназначен для использования под управлением системы Windows и стоит 99 долларов.
- ✓ **HTML-Kit** (`www.chami.om/html-kit/`). Это еще один редактор для Windows, который можно использовать бесплатно.
- ✓ `vim` **и** `gvim`: (`www.vim.org/`). Эти бесплатные улучшенные версии редактора `vi` можно использовать в операционных системах Windows, Linux, Unix и MacOS. Редактор `gvim` имеет графический пользовательский интерфейс, который позволяет пользователям Windows чувствовать себя как дома.

Интегрированная среда разработки

Интегрированная среда разработки (Integrated Development Environment — IDE) — это целостное рабочее пространство для разработки приложений. Наряду с другими средствами она включает редактор для написания программ. Необходимо отметить следующие возможности, которыми обладает большинство IDE.

- ✓ **Средства отладки.** *Содержит встроенные возможности по отладке* программного кода.
- ✓ **Предварительный просмотр.** Обеспечивает отображение Web-страницы, генерируемой сценарием.
- ✓ **Средства тестирования.** Имеет встроенные средства тестирования сценариев.
- ✓ **FTP.** Имеет встроенную поддержку протокола FTP и его использования для загрузки файлов. Позволяет отслеживать файлы, относящиеся к Web-узлу, и обновлять его содержимое.
- ✓ **Управление проектом.** Организует сценарии в проекты; управляет файлами проекта, включая отладку файлов и их регистрацию.
- ✓ **Резервное копирование.** Периодически (автоматически) создает резервные копии Web-узла.

Интегрированные средства разработки более сложны в изучении, чем обычные редакторы. Некоторые из них довольно дороги, но множество их преимуществ стоят этих денег. Интегрированная среда разработки особенно полезна, когда сценарии для одного проекта разрабатываются несколькими программистами. Интегрированная среда разработки существенно упрощает координацию проектов и обеспечивает совместимость кода.

Отметим некоторые популярные интегрированные среды разработки.

- ✓ **Dreamweaver MX** (`www.macromedia.com/dreamweaver`). Эта интегрированная среда разработки доступна для платформ Mac и Windows. Она обеспечивает визуальные средства макетирования, позволяя создать Web-страницу, перетягивая элементы с панели инструментов и щелкая на соответствующих кнопках. Dreamweaver может самостоятельно написать HTML-код вместо вас. Она включает редактор HomeSite, поэтому вы можете написать свой собственный код, а также поддерживает язык PHP. Стоимость Dreamweaver — 399 долларов.
- ✓ **Komodo** (`www.activestate.com/Products/Komodo/`). Komodo существует для платформ Windows и Linux. Это интегрированная среда разработки с использованием языков с открытым кодом, включая Perl, Python, а также PHP. Стоимость этой системы — 29,95 доллара для использования в личных или образовательных целях и 295 долларов — для коммерческого использования.
- ✓ **Maguma** (`www.maguma.com`). Maguma доступна только для Windows. Это интегрированная среда разработки с использованием Apache, PHP и MySQL под управлением операционной системы Windows. Она поставляется в двух версиях, Maguma Studio Desktop и Maguma Studio Enterprise, имеющих разную стоимость, и предоставляет средства разработки очень больших узлов с несколькими серверами. Maguma Studio для PHP — это свободно распространяемая версия с поддержкой только языка PHP.
- ✓ **PHPEdit** (`www.phpedit.net/products/PHPEdit/`). Эта бесплатная интегрированная среда разработки доступна только для системы Windows.
- ✓ **Zend Studio** (`www.zend.com/store/products/zend-studio.php`). Zend Studio предназначена для использования в системах Windows и Linux. Эта интегрированная среда разработки была написана людьми, которые создали механизм Zend — ядро PHP. Цена Zend Studio — 195 долларов.

Web-страницу с описанием редакторов и интегрированных сред разработки, которые можно использовать для создания сценариев PHP, можно найти по адресу `phpeditors.linuxbackup.co.uk`. В настоящее время в списке содержится описание 111 редакторов.

Глава 3

Создание первого сценария PHP

В этой главе...

- Операторы языка PHP
- Включение фрагментов PHP в файлы HTML
- Операторы PHP для вывода данных
- Документирование сценариев

Оператор языка PHP — это инструкция, которая указывает интерпретатору PHP, какие действия нужно выполнить. PHP-сценарий — это последовательность операторов PHP. Теоретически сценарий может содержать один или несколько операторов, но вряд ли какой-либо сценарий будет состоять из единственного выражения. В большинстве случаев сценарии включают по несколько операторов в строке. Интерпретатор PHP выполняет операторы по-одному до тех пор, пока не достигнет конца сценария.

Как уже упоминалось в главе 1, PHP обеспечивает множество возможностей, а сценарии позволяют объяснить интерпретатору PHP, что вы хотите. Вы можете вывести текст на Web-страницу или сохранить данные, которые вводятся пользователем в форму на Web-странице. PHP также может выполнять действия, которые не свойственны Web-узлам, например записывать файлы в каталог на жестком диске. Можно написать простой сценарий, который выводит приветствие в окне Web-браузера. Можно создать сложный сценарий, который выводит в окне браузера разный текст для разных людей, запрашивает пароль у посетителей Web-узла или отказывает в доступе тем посетителям, которые не ввели нужный пароль. Приложение часто состоит из двух или нескольких сценариев, которые совместно выполняют требуемые действия. Большие, сложные приложения, такие как приложения электронной коммерции, могут состоять из многих сценариев.

В этой главе речь пойдет о том, как написать собственный сценарий. Мы обсудим операторы вывода, которые чаще всего встречаются в сценариях PHP. И, наконец, будет продемонстрирована важность документирования сценариев.

Написание операторов PHP

Оператор PHP определяет, какие действия необходимо выполнить. Одним из наиболее популярных операторов PHP является `echo`. Он предназначен для вывода информации. Рассмотрим следующий пример использования оператора `echo`:

```
echo "Привет";
```

Оператор `echo` выводит все, что находится между двойными кавычками (`""`). Поэтому в данном случае отобразится строка `Привет`.

Оператор `echo` является *простым* (simple statement). Все простые операторы в языке PHP заканчиваются точкой с запятой (`;`). Интерпретатор PHP считывает простые операторы до тех пор, пока не встретит символ `;` (или закрывающий дескриптор, который рассматривается далее в этой главе). При этом все пробелы игнорируются. Не имеет значения, сколько считано строк, каков смысл или синтаксис оператора. Все содержимое сценария просто считывается до точки с запятой и интерпретируется как один оператор.

Отсутствие точки с запятой является распространенной ошибкой, в результате которой выводится сообщение об ошибке. Оно может выглядеть следующим образом:

```
Parse error: expecting `','` or `';'` in file.php on line 6
```

Обратите внимание, что в сообщении указывается номер строки, где содержится ошибка. При этом строка, в которой пропущена точка с запятой, является предыдущей по отношению к указанной в сообщении, т.е. в данном случае точка с запятой отсутствует в пятой строке.

Для написания сценариев PHP можно воспользоваться специальным редактором, в котором будут отображаться номера строк. В этом случае будет проще анализировать код сценария. Иначе придется отсчитывать номера строк, чтобы найти ту, в которой содержится ошибка. Действительно, если сценарий состоит из 6 строк то пересчитать их не так уж и сложно. Но если в сценарии содержится 553 строки, то жизнь перестает быть веселой. Необходимо отметить, что многие редакторы позволяют указать номер строки, к которой необходимо перейти.

В PHP весь сценарий можно разместить в одной длинной строке, разделяя при этом простые операторы точкой с запятой. Но в этом случае анализ и чтение сценария существенно затруднятся. Таким образом, простые операторы лучшее размещать в разных строках.

Иногда несколько операторов помещают в один блок, заключенный в фигурные скобки ({ }). Операторы в блоке выполняются вместе, как один оператор. Очень часто блоки кода используются совместно с условным оператором, где набор инструкций выполняется при выполнении некоторого условия. Рассмотрим следующий пример:

```
if (время_суток == полночь)
{
      одеть пижаму;
      почистить зубы;
      лечь в кровать;
}
```

Фигурные скобки указывают, что набор операторов в блоке будет выполняться как одно целое. В полночь, т.е. когда `время_суток = полночь`, будут выполнены все три действия, указанные в блоке. В противном случае ничего делать не надо (не надо одевать пижаму, чистить зубы, идти ложиться спать).

Операторы, использующие блоки кода (как, например, оператор `if`), называются *сложными* (complex statement). В этом случае интерпретатор PHP считывает весь сложный оператор, не останавливаясь перед первым символом точки с запятой. Он допускает существование нескольких блоков кода и ищет последнюю фигурную скобку последнего блока. Следует также отметить, что перед закрывающей фигурной скобкой стоит точка с запятой, а после — нет.

Операторы внутри блока вводятся с отступом. Такое размещение влияет только на удобочитаемость кода, но никак не на результат выполнения. Человеку, анализирующему код сценария, будет намного легче определять, где начинается и заканчивается тот или иной блок. Отсутствие закрывающей фигурной скобки является распространенной ошибкой, особенно если используется несколько вложенных блоков. Использование отступов позволит быстро найти недостающую скобку.

Написание сценариев

Для написания полноценного сценария необходимо добавить несколько операторов в файл, который имеет расширение `.php`. Однако, если сценарий не так уж прост или если у вас нет опыта, будет разумным сначала написать код сценария на бумаге. Помните, что тщательное планирование позволит избежать многих ошибок при написании программного кода.

При создании сценария PHP для Web-узла его программный код необходимо каким-то образом вставить в HTML-код Web-страницы. Если же сценарий создается независимо от Web, то операторы PHP помещаются в отдельный файл, который вызывается напрямую. В следующих разделах будет показано, как следует поступать в обоих случаях.

Как сервер обрабатывает файлы PHP

Если пользователь обращается к обычному файлу HTML (с расширением `.html` или `.htm`), Web-сервер передает его код напрямую браузеру. Браузер, в свою очередь, обрабатывает полученный файл с дескрипторами HTML и выводит Web-страницу. Если же пользователь сгенерировал запрос к файлу PHP (с расширением `.php`), Web-сервер осуществляет обработку кода PHP (а не отправляет его сразу браузеру). При этом выполняются следующие шаги.

1. Web-сервер просматривает исходный файл в режиме HTML, т.е. он никак не обрабатывает дескрипторы HTML, а отправляет их напрямую браузеру.
2. Просмотр в режиме HTML выполняется до тех пор, пока не будет найден открывающий дескриптор PHP (`<?php`).
3. Найдя открывающий дескриптор `<?php`, Web-сервер переключается в режим PHP. Все, что находится после дескриптора `<?php`, интерпретируется Web-сервером как PHP-код и соответствующим образом обрабатывается. Например, если в коде PHP содержится оператор вывода, то соответствующая информация передается браузеру.
4. Web-сервер продолжает работу в режиме PHP до тех пор, пока не встретит закрывающий дескриптор PHP (`?>`).
5. Найдя закрывающий дескриптор `?>`, Web-сервер снова переключается в режим HTML и продолжает выполнять действия, начиная с п. 1.

Вставка операторов PHP в HTML-код

При использовании языка PHP для создания Web-узлов операторы PHP необходимо вставлять в HTML-код Web-страницы. При этом соответствующие файлы должны иметь расширение `.php`, чтобы Web-сервер мог знать о наличии кода PHP. (Администратор Web-сервера может использовать и другие расширения файлов, которые могут содержать операторы PHP, например `.php4` или `.phtml`. Однако расширение `.php` является наиболее популярным, поэтому оно и будет использоваться в данной книге.)

Для вставки кода PHP в файл HTML используются дескрипторы, наподобие тех, что применяются в файлах HTML. Общий синтаксис имеет следующий вид:

```
<?php
...
операторы PHP
...
?>
```

Можно также использовать сокращенную форму дескрипторов PHP `<?` и `?>`, что позволяет немного сократить код. Для того чтобы активизировать или отключить использование сокращенных дескрипторов, следует внести соответствующие изменения в конфигурационный файл `php.ini`.

Иногда использование сокращенных дескрипторов является не очень хорошей идеей. Действительно, если разместить Web-узел на сервере, который не поддерживает сокращенные дескрипторы, PHP-код будет обрабатываться некорректно, т.е. для переносимости кода лучше применять полные дескрипторы PHP.

Все операторы, находящиеся между открывающим и закрывающим дескрипторами PHP, передаются Web-сервером на обработку интерпретатору PHP. После выполнения всех действий код PHP опускается, а результат передается обратно Web-серверу. Тот, в свою очередь, отправляет на сторону пользователя (т.е. браузеру) код HTML и результаты обработки сценария PHP. При этом код PHP в браузере не отображается.

Например, добавим в файл HTML следующие строки кода PHP. При этом не забудьте сохранить этот файл с расширением .php:

```
<?php
   echo "Это привет от PHP";
?>
```

При получении Web-сервером запроса на файл с расширением .php он проверяет его на наличие дескрипторов PHP. Найдя открывающий дескриптор <?php, Web-сервер выполнит оператор echo, а не просто отправит его браузеру. Браузеру будет передан результат выполнения, а именно строка Это привет от PHP, которая и отобразится в его окне. Даже если просмотреть исходный код в браузере, в нем вы не увидите код PHP.

Не пытайтесь просматривать файл PHP, используя браузер напрямую, т.е. для перемещения к файлу не надо выбирать в меню браузера команду File⇨Open⇨Browse (Файл⇨Открыть⇨Обзор). К файлу нужно обращаться через полный URL-адрес сценария PHP, как описано в главе 2.

На Web-страницу можно добавить несколько фрагментов PHP-кода. В качестве примера приведем следующий фрагмент.

```
дескрипторы HTML
<?
     echo "Это привет от PHP";
?>
дескрипторы HTML
<?
     echo " Это тоже привет от PHP";
?>
```

В результате в окне браузера отобразятся две строки.

Использование PHP независимо от Web

PHP можно использовать в качестве самостоятельного языка написания сценариев, независимо от Web. Для этого в PHP предназначен специальный интерфейс CLI (Command Line Interface — интерфейс командной строки). Версия PHP CLI отлична от той, которая используется для работы с Web-серверами (обычно называемой PHP CGI), и устанавливается независимо. В приложении А приводятся более подробные инструкции по установке интерфейса CLI.

Если вы намерены использовать интерфейс командной строки CLI для PHP, вы, скорее всего, работаете на компьютере под управлением операционной системы Unix или Linux. Программисты, работающие в Windows, вряд ли будут использовать PHP в качестве самостоятельного языка написания сценариев (хотя в случае необходимости это можно сделать). В данном разделе речь пойдет об использовании версии PHP CLI в операционных системах Linux/Unix, но большая часть информации останется верной и для Windows. (В некоторых случаях будут указаны соответствующие расхождения.)

Рассмотрим следующий сценарий PHP:

```
<?php
     echo "Это привет от PHP";
?>
```

Выполнение сценариев PHP в операционных системах Linux/Unix

Если у вас уже имеется опыт запуска сценариев оболочки (shell script) или сценариев Perl в операционных системах Linux/Unix, то выполнение PHP-сценариев не составит особого труда. Нужно просто добавить строку, указывающую расположение интерфейса командной строки PHP CLI, и тогда вызов сценария можно осуществлять напрямую:

```
#! /usr/bin/php
<?php
     echo "Это привет от PHP";
?>
```

Первая строка указывает, что сценарий будет выполнен с использованием программы, расположенной по адресу `/usr/bin/php`. В операционной системе Windows эта строка не работает, хотя и не приводит к генерации ошибки. Строку `#! /usr/bin/php` можно спокойно включать при создании сценариев PHP для Linux/Unix, не особенно заботясь о том, что при переходе к Windows она не будет работать.

Для запуска сценария PHP необходимо ввести его имя. Это можно сделать в каталоге, в котором содержится исходный файл, или указать полный путь к нему. Например, если файл, содержащий приведенный выше фрагмент кода, называется `test.php`, то для его запуска следует ввести имя

```
test.php
```

или указать полный путь к нему:

```
/mystuff/test.php
```

Не забывайте также о том, что для выполнения этого файла, как и для запуска любого другого сценария, нужно иметь соответствующие разрешения.

Если назвать этот файл `testcli.php`, то для его запуска необходимо ввести следующую строку (при условии, что он находится в том же каталоге, где установлен PHP CLI):

```
php testcli.php
```

Можно также указать полный путь к интерпретатору PHP:

```
/usr/local/php/cli/php testcli.php
```

Если же сценарий PHP требуется запустить в операционной системе Windows, следует воспользоваться консольным режимом ввода команд. Для этого необходимо выбрать команду меню Start⇨Programs⇨Accessories⇨Command Prompt (Пуск⇨Программы⇨Стандартные⇨Командная строка).

Основные отличия версии CLI от CGI состоят в следующем.

- ✓ **Вывод заголовков HTTP**. Поскольку в версии CGI результат выполнения сценария передается сначала Web-серверу, а потом браузеру, к нему добавляются заголовки HTTP (информация, с помощью которой Web-сервер и браузер взаимодействуют друг с другом). Таким образом, в предыдущем примере версия PHP CGI выдаст следующий результат:

  ```
  Content-type: text/html
  X-Powered-By: PHP/5.0

  Это привет от PHP
  ```

  Первые две строки в окне браузера не отображаются, но они необходимы для корректной работы Web-сервера. В свою очередь, версия PHP CLI автоматически не передает заголовки HTTP, а просто выведет одну строку:

  ```
  Это привет от PHP
  ```

- ✓ **Формат сообщений об ошибках**. При использовании версии PHP CGI сообщения об ошибках будут отформатированы с помощью дескрипторов HTML, поскольку, вероятнее всего, они будут отображаться в браузере. Сообщения об ошибках в версии PHP CLI никак не форматируются.
- ✓ **Использование значений по умолчанию для параметров `argc` и `argv`**. Переменные `argc` и `argv` используются для передачи данных в сценарий из командной строки (как в C и других языках). Вряд ли вы будете использовать эти параметры в версии CGI, в то время как в CLI — скорее всего, будете. Поэтому по умолчанию эти параметры доступны в CLI и недоступны в CGI. (Более подробно переменные `argc` и `argv` описаны в главе 5.)

При использовании PHP для командной строки можно воспользоваться некоторыми дополнительными параметрами (ключами). Например, ключ `-v` позволяет вывести версию PHP. Для этого нужно ввести следующую строку:

```
php -v
```

В табл. 3.1 приведены наиболее полезные параметры командной строки.

Таблица 3.1. Параметры командной строки PHP

Параметр	Назначение
`-c`	Определяет путь к используемому файлу `php.ini`. Версией CLI может использоваться свой собственный файл `php.ini`, например `-c /usr/local/php/cli/php.ini` (Более подробную информацию о файле `php.ini` можно найти в приложении А.)
`-f`	Идентифицирует запускаемый сценарий. Например, `php -f /myfiles/testcgi.php`
`-h`	Выводит справочную информацию
`-i`	Выводит информацию о PHP, которая аналогична результатам функции `phpinfo()` (см. главу 2)
`-l`	Проверяет сценарий на наличие ошибок, но не выполняет его
`-m`	Выводит список скомпилированных модулей PHP. (Более подробно модули описаны в главе 14.)
`-r`	Выполняет код PHP, введенный в командной строке. Например, `php -r 'print('Привет');'`
`-v`	Выводит номер версии PHP

Создание первого сценария PHP

За многие годы сложилась такая традиция, что первая созданная программа должна выводить строку `Здравствуй, мир`. Если вы изучали HTML, то вам, наверное, доводилось осуществлять эту процедуру и вы использовали примерно следующий код:

```
<html>
<head><title>HTML-программа</title></head>
<body>
<p>Здравствуй, мир!</p>
</body>
</html>
```

Если отобразить содержимое этого файла в браузере, получим очевидный результат:

```
Здравствуй, мир!
```

Ваш первый сценарий на PHP будет выполнять абсолютно те же действия. Ниже приведен фрагмент кода, который содержит дескрипторы HTML и PHP-код и выводит в окне браузера строку `Здравствуй, мир!`

```
<html>
<head><title>Сценарий PHP</title></head>
<body>
<?php
   echo "<p>Здравствуй, мир!</p>";
?>
</body>
</html>
```

Результат выполнения этого сценария будет таким же, как и результат вывода файла HTML.

Нет смысла просматривать файл PHP, используя напрямую браузер, т.е. для перехода к файлу нет смысла выбирать в меню браузера команду File⇨Open⇨Browse (Файл⇨Открыть⇨Обзор). Для обращения к файлу необходимо указать полный URL-адрес сценария PHP в адресной строке браузера (см. главу 2). Если код PHP не выполняется, а просто отображается в окне браузера, значит, вы обратились к нему не через URL-адрес.

В приведенном выше примере фрагмент PHP состоит из следующих строк:

```
<?php
   echo "<p>Здравствуй, мир!</p>"
?>
```

Между открывающим и закрывающим дескрипторами PHP содержится только один оператор `echo`, который выводит строку, заключенную в двойные кавычки (`" "`).

Таким образом, в результате выполнения сценария браузеру будет передана следующая строка:

```
<p>Здравствуй, мир!</p>
```

Если фрагмент PHP-кода заменить этой строкой, то получим обычный HTML-файл. При этом в обоих случаях результат будет один и тот же, точно так же, как и исходный код, обрабатываемый браузером (чтобы его увидеть в меню браузера, следует выбрать команду View⇨Source (Вид⇨Просмотр HTML-кода)).

Детальнее об операторах вывода

В сценарии, созданном в предыдущем разделе, для отображения информации использовался оператор вывода `echo`. *Операторы вывода* (output statement) встречаются практически во всех сценариях. Очень редко приходится создавать приложения, в которых ничего не выводится. Действительно, PHP-сценарий может выполнять любые действия незаметно для пользователя, например проверять наличие файла на жестком диске. Но если клиенту ничего не будет выведено, то в чем тогда смысл? Как пользователь узнает, найден файл или нет? А если найден, то где он в таком случае расположен? Соответственно, практически во всех сценариях используются операторы вывода.

Общий синтаксис оператора `echo` выглядит следующим образом:

```
echo элемент_вывода1,элемент_вывода2,элемент_вывода3,...
```

При работе с этим оператором следует руководствоваться такими правилами.

- ✓ Параметр *элемент_вывода* может быть числом (например, таким как 1 или 250) либо текстовой строкой. Более подробно типы данных рассматриваются в главе 5.
- ✓ Текстовая строка должна быть заключена в одинарные или двойные кавычки. (В главе 5 обсуждается, какие кавычки следует использовать в тех или иных случаях.)
- ✓ Количество аргументов оператора `echo` может быть произвольным.
- ✓ Аргументы оператора `echo` разделяются запятой (`,`) без использования пробелов.
- ✓ Если необходимо добавить пробел при выводе различных значений, его следует добавить в строку вывода в качестве отдельного символа.

В табл. 3.2 приведены некоторые примеры использования оператора `echo`.

Таблица 3.2. Примеры использования оператора `echo`

Пример	Результат выполнения
`echo 123;`	`123`
`echo "Здравствуй, мир!";`	`Здравствуй, мир!`
`echo "Здравствуй,","мир!";`	`Здравствуй,мир!`
`echo "Здравствуй,", " ","мир!";`	`Здравствуй, мир!`
`echo Здравствуй, мир!;`	Выдаст сообщение об ошибке, поскольку строка не заключена в кавычки
`echo ''Здравствуй, мир!';`	`Здравствуй, мир!`

Обработка операторов вывода PHP

В сценарии, выводящем строку `Здравствуй, мир!`, как и в большинстве других сценариев, операторы вывода используются для генерирования кода HTML, который отображается в браузере. В этом случае необходимо помнить о двух этапах, которые при этом выполняются

1. Интерпретатор PHP выполняет операторы PHP, перенаправляет результат работы Web-серверу, который, в свою очередь, отправляет их клиентскому браузеру.

   PHP никак не обрабатывает HTML-код, а только передает информацию, полученную в результате выполнения PHP-кода.

2. Web-браузер интерпретирует полученную от интерпретатора PHP информацию как HTML-код и выводит ее на Web-странице.

   В отличие от PHP он обрабатывает только HTML-код и не "понимает" кода PHP. Таким образом, нужно позаботиться о том, чтобы Web-браузер корректно интерпретировал информацию, полученную в результате выполнения фрагмента кода на PHP.

В качестве примера рассмотрим оператор вывода `echo`, отображающий строку `Здравствуй, мир!`:

```
echo "<p>Здравствуй, мир!</p>";
```

Он выводит все, что находится в двойных кавычках (`" "`). Обработка этого PHP-кода будет выполнена в два этапа.

1. Результатом выполнения оператора `echo` будет следующая строка:

   ```
   <p>Здравствуй, мир!</p>
   ```

Поскольку интерпретатор PHP не понимает HTML-код, то дескрипторы и никак обработаны не будут, а будут восприниматься как символы текстовой строки.

2. Получив строку, Web-браузер, в свою очередь, интерпретирует символы и как открывающий и закрывающий дескрипторы HTML для форматирования текста и выводит его в окне браузера соответствующим образом:

```
Здравствуй, мир!
```

При этом сами дескрипторы не отображаются. Для того чтобы увидеть, что именно было передано браузеру Web-сервером, можно отобразить исходный код HTML-страницы. Для приведенной страницы он будет иметь следующий вид:

```
<p>Здравствуй, мир!</p>
```

Использование специальных символов в операторах вывода

Оператор echo специальным образом обрабатывает некоторые символы, которые существенно влияют на получаемый результат. Одним из них является символ перехода на новую строку \n. Рассмотрим следующий пример:

```
echo "<p>Здравствуй,\n мир!</p>"
```

Символ \n в операторе echo означает переход на новую строку, однако это не отображается в браузере. Для перехода к новой строке на Web-странице необходимо использовать внутри строки специальный дескриптор
. Рассмотрим следующий пример:

```
echo "<p>Здравствуй,<br> мир!</p>"
```

В этом случае в браузере будет выведено две строки.

В табл. 3.3 приводятся отличия результатов выполнения оператора echo на разных этапах: после его обработки интерпретатором PHP и в Web-браузере. Первый столбец содержит пример фрагмента кода с оператором echo. Во втором приводится результат его обработки интерпретатором PHP (т.е. то, что передается браузеру), а в третьем — результат, отображаемый в самом браузере.

Таблица 3.3. Этапы обработки страницы с PHP-кодом

Пример оператора echo	Результат обработки интерпретатором PHP	Результат отображения в браузере
`echo "Здравствуй, мир!";`	`Здравствуй, мир!`	`Здравствуй, мир!`
`echo "Здравствуй,";`	`Здравствуй,мир!`	`Здравствуй,мир!`
`echo "мир!";`		
`echo "Здравствуй,\nмир!";`	`Здравствуй,` `мир!`	`Здравствуй, мир!`
`echo "Здравствуй, мир!";`	`Здравствуй, мир!`	`Здравствуй,` `мир!`
`echo "Здравствуй, \nмир!";`	`Здравствуй, ` `мир!`	`Здравствуй,` `мир!`

Обратите внимание на наличие пробела в результирующих выражениях. Так, в первом примере пробел содержится в текстовой строке, выводимой оператором echo. Следовательно, этот символ будет содержаться и после обработки интерпретатором PHP, и в браузере. Во втором случае выполняются два разных вызова оператора echo. Причем, поскольку ни одна из выводимых строк не содержит пробела, в результирующих выражениях он также отсутствует. В третьем примере при отображении текста в окне браузера будет выведен пробел, хотя

в строке Здравствуй\nмир! его нет. Это связно с тем, что специальный символ \n никак не интерпретируется браузером (т.е. не осуществляется перевод на новую строку), а просто заменяется на пробел.

Свободно используйте символ перевода строки \n. В противном случае исходный HTML-код будет содержать очень длинные строки. Представим, например, что имеется форма, содержащая большое количество полей. Несмотря на то что она может иметь нормальный вид в окне браузера, ее HTML-код может быть размещен всего в нескольких строках. В общем-то, это не создает никаких проблем до тех пор, пока не понадобится поработать с кодом HTML самой Web-страницы. В этом случае придется серьезно попотеть, отыскивая ошибку в многокилометровых строках кода. Поэтому использование символа \n позволит привести HTML-код к более удобному для анализа виду. Кроме того, некоторые браузеры могут некорректно обрабатывать длинные строки.

PHP выводит в точности то, что ему указывают. Интерпретатору PHP не важно, где впоследствии будет отображаться информация — в окне браузера или просто на экране. Прежде всего, это ваша забота. Если разрабатывается приложение для Web, то выводимые строки должны содержать дескрипторы HTML. Если же язык PHP используется для написания сценария, независимого от Web, в этом случае информацию лучше выводить в виде обычного текста.

Документирование сценариев

Документирование сценариев является очень важным этапом при разработке приложений. Комментарии позволяют описать, что именно и как выполняет сценарий. Чем больше, сложнее и замысловатее код сценария, тем большее значение приобретает документирование. Проработав двадцать часов над сценарием, вы будете уверены, что он навеки останется в вашей памяти. Однако по прошествии двух лет, когда необходимо будет пересмотреть код сценария, вы его совсем не узнаете. Также вполне возможна ситуация, когда действительно кто-то другой захочет переделать код сценария, а в это время вы будете недоступны, отдыхая на Багамах.

Комментарии (comments) представляют собой заметки, вставляемые в код сценария. При этом они игнорируются интерпретатором PHP и предназначены для пользователей. Комментарии в сценарии могут быть произвольной длины. Их общий синтаксис имеет следующий вид:

```
/* текст комментариев
еще комментарии */
```

Когда интерпретатор PHP встречает в сценарии символы начала комментария (/*), весь последующий код игнорируется вплоть до символов */.

Очень часто разработчики добавляют комментарии в начало сценариев, чтобы дать о нем краткую информацию и указать, какие функции он выполняет. Рассмотрим следующий пример:

```
/*    имя сценария:     hello.php
      описание:         отображает строку "Здравствуй, мир!"
                        на Web-странице
      автор:            Программист Джон
      дата создания:    02/01/03
      дата изменения:   03/15/03
*/
```

В языке PHP комментарии можно использовать и в сокращенной форме. Для этого можно использовать символ решетки (#) либо два символа косой черты (//). Они позволяют закомментировать целую строку. Например:

```
# Строка комментариев 1
// Строка комментариев 2
```

Символы # и // можно также использовать в строке кода, указывая тем самым на начало комментария. Это особенно полезно для описания работы отдельного оператора. Рассмотрим следующий пример:

```
echo "Привет"; // это мой первый оператор вывода
```

Следует отметить, что интерпретатор PHP не включает комментарии в HTML-код, передаваемый впоследствии браузеру, так что пользователь их увидеть не сможет.

Иногда очень полезно использовать комментарии в качестве заголовка разделов кода, например:

```
/* Проверяет, исполнилось ли покупателю 18 лет */
/* Сохраняет информацию в базе данных */
/* Осуществляет поиск указанного файла */
```

Иногда следует выделить комментарии, чтобы заострить на них внимание. Это можно сделать следующим образом:

```
################################
# Дважды проверьте этот раздел #
################################
```

Используйте комментарии так часто, как того требует ситуация. Однако, с другой стороны, не стоит ими злоупотреблять: большое количество комментариев может резко снизить их ценность. Их лучше всего использовать для описания различных разделов кода сценария или в случае нестандартного (или неочевидного) подхода к написанию программы. Например, комментарий, поясняющий работу оператора echo, является лишним, поскольку и так понятно, для чего он предназначен. Если смысл кода очевиден, не следует загромождать его лишними комментариями.

Внимательно следите за тем, чтобы не использовать в сценарии вложенные комментарии. В этом случае код PHP будет работать некорректно. Рассмотрим следующий пример:

```
/* Первый комментарий
   /* Вложенный комментарий */
*/
```

Найдя символы начала комментария, /*, интерпретатор PHP будет игнорировать код до тех пор, пока не встретит символы */. Поэтому будет проигнорирован и второй набор символов /*. Таким образом, будет закомментировано все, что находится между первыми символами /* и */. Дойдя до второго набора символов */, интерпретатор PHP выдаст сообщение об ошибке, поскольку он не интерпретирует их как символы окончания комментария.

Часть II

Переменные и данные

"Извините, но доменные имена "ГАВ", "ВАФ" и "БРР" уже зарегистрированы в сети. Можем предложить "МЯУ", "КВА" или "МУУ".

В этой части...

В этой части речь пойдет о создании и использовании переменных в языке PHP. Будут описаны типы данных, которые могут храниться в переменных, а также особенности хранения различных типов. Вы узнаете также, как хранить взаимосвязанные данные в сложных переменных, получивших название *массивов*.

Глава 4

Использование переменных в сценариях PHP

В этой главе...

- Имена переменных
- Присваивание значений переменным
- Удаление переменных
- Использование констант
- Обработка ошибок

Переменные (variable) — это контейнеры, в которых содержится информация. Сначала задается имя переменной, а затем в этой переменной можно хранить информацию. Например, можно назвать переменную `$age` и записать в нее число 21. После присваивания переменной некоторого значения эту переменную можно использовать в сценарии.

При использовании языка PHP для Web переменные зачастую применяются для хранения информации, которую пользователи вводят в формах HTML, например имени пользователя. Тогда эту переменную можно использовать в сценарии или даже создать персональную Web-страницу с отображением имени пользователя, например *"Приветствую Вас, Сергей Петренко!"*

В этой главе речь пойдет о том, как описывать переменные и работать с содержащейся в них информацией. Вы также узнаете, как обрабатывать ошибки.

Имена переменных

Имена переменных, или *идентификаторы*, должны быть осмысленными. Автору приходилось встречать сценарии, где все переменные назывались `$var1`, `$var2`, `$var3` и т.д. Переменные, конечно, можно называть подобным образом, но если спустя какое-то время вы вернетесь к этому сценарию, у вас могут возникнуть проблемы с выяснением смысла каждой переменной. Интерпретатор PHP не будет затрудняться или путаться, но люди, которые попытаются разобраться в сценарии, вынуждены будут тратить лишнее время. Делайте ваши сценарии как можно проще для понимания, используя осмысленные имена переменных, например `$firstName`, `$directory_name` или `$DateOfBirth`.

Приведем основные правила именования переменных.

- ✓ **Все имена переменных начинаются с символа доллара** (`$`). Этот символ в языке PHP означает начало имени переменной.
- ✓ **Имена переменных могут быть любой длины.**
- ✓ **Имена переменных могут включать только буквы, цифры и символы подчеркивания.**
- ✓ **Имена переменных должны начинаться с буквы или символа подчеркивания.** Они не могут начинаться с цифры.

✓ **Символы верхнего и нижнего регистров различаются.** Имена `$favoritecity` и `$Favoritecity` означают разные переменные. Если информация хранится в переменной `$FavoriteCity`, вы не сможете позже в сценарии получить доступ к этой информации, используя имя переменной `$favoriteCity`.

Приведем примеры допустимых имен переменных:

```
$_name
$first_name
$name3
$name_3
```

Следующие имена переменных приведут к генерации сообщений об ошибках:

```
$3name
$name?
$first+name
$first.name
```

Первое имя недопустимо, потому что оно начинается не с буквы или символа подчеркивания, как требуют правила именования. Три остальных имени недопустимы, поскольку кроме цифр, букв и символа подчеркивания в них содержатся другие символы.

Выбор имени переменной зависит от личных предпочтений. Чаще всего для создания осмысленных имен переменных используется символ подчеркивания или прописные буквы. Эти наиболее популярные стили формирования имен переменных проиллюстрированы на следующем примере:

```
$first_name
$firstName
```

Описание переменных на основе одного из двух приведенных стилей облегчает чтение сценариев. Обычно имя начинается с символа нижнего регистра. Однако главное правило выбора имен переменных — их согласованность. Выберите стиль и используйте его во всем сценарии

Присваивание и отображение значений переменных

Переменные могут содержать числа или строки символов. Переменная может существовать или не существовать, содержать или не содержать информацию. Причем это не одно и то же. Даже если переменная в настоящее время не содержит никакой информации, она может существовать так же, как существует полка шкафа, даже если она пуста. Конечно, если переменная содержит информацию, она должна существовать.

В следующих разделах рассказывается о том, как создавать переменные и работать с их значениями.

Создание переменных

Переменная автоматически создается при сохранении в ней некоторой информации.

Чтобы задать значение переменной, нужно использовать знак равенства (=) . Например, следующие четыре оператора PHP присваивают значения переменным:

```
$age = 21;
$price = 20.52;
$temperature =-5;
$name = "Кларк Кент";
```

Обратите внимание, что в этих примерах числа не заключены в кавычки, а имя, которое является строкой символов, — заключено. Кавычки в PHP свидетельствуют о том, что символы входят в состав строки, обрабатываемой интерпретатором PHP как отдельная единица. Без кавычек интерпретатор PHP не будет знать, что символы образуют строку, и будет неправильно их обрабатывать. Различные типы данных и особенности их использования будут подробно обсуждаться в главе 5.

Всякий раз, когда значение присваивается несуществующей переменной, эта переменная создается. Например, предположим, что в начале сценария используются следующие операторы PHP:

```
$color = "синий";
$color = "красный";
```

Первый оператор создает переменную и присваивает ее значение `"синий"`. Следующая инструкция изменяет значение переменной `$color` на `"красный"`.

Значение одной переменной можно присваивать другой переменной, как показано в следующих примерах:

```
$name1 = "Салли";
$name2 = "Сюзан";
$favorite_name = $name2;
```

После выполнения этих операторов в переменной `$favorite_name` будет содержаться значение `"Сюзан"`.

Можно создать переменную, не присваивая ей никакого значения. Например, следующая инструкция создает переменную:

```
$city = "";
```

Теперь переменная существует, но не содержит никакого значения. Более подробная информация о типах данных и особенностях их использования содержится в главе 5.

Отображение значений переменных

Самый быстрый способ отображения значения переменной заключается в использовании функции `print_r`. Приведем пример отображения значения переменной:

```
$today = "Воскресенье";
print_r($today);
```

Результатом выполнения кода будет вывод слова `Воскресенье`.

Значение можно также отобразить и с помощью оператора `echo`. Результатом выполнения кода PHP

```
$age = 21;
echo $age;
```

будет вывод числа 21.

Использование оператора `echo` указанного вида приводит к тому же результату, что и применение функции `print_r`. Однако оператор `echo` позволяет сделать гораздо больше. С его помощью можно вывести несколько элементов, включая числа и строки. Например, предположим, что в переменной `$name` содержится значение `Кларк Кент`. Вы можете включить в HTML-файл следующую строку:

```
<p> Добро пожаловать, <?php echo $name?>/p>
```

Результат ее обработки на Web-странице будет выглядеть так:

```
Добро пожаловать, Кларк Кент
```

Если вы попытаетесь использовать несуществующую переменную, то будет сгенерировано предупреждение. Например, предположим, что вы хотите вывести значение переменной `$age`, но допустили ошибку:

```
echo $aeg;
```

В результате появится следующее предупреждение:

```
Notice: Undefined variable: aeg in c:\testvar.php on line 5
```

В этом сообщении (Notice) говорится о том, что вы используете неопределенную переменную. Такое сообщение в данном случае полезно, поскольку позволяет точно определить строку с опечаткой. Однако в некоторых случаях при написании кода может использоваться несуществующая переменная, и это не будет опечаткой (такая переменная может быть задана преднамеренно). Например, такой подход можно использовать в условном операторе (условные операторы будут описаны в главе 7). Сценарий при этом будет работать как положено, и предупреждение в этом случае будет совсем ни к чему. Эту проблему можно решить, добавив перед именем переменной символ @. Если вы хотите, чтобы предупреждение не отображалось, используйте следующий синтаксис:

```
echo @$aeg;
```

Таким образом, можно отключить вывод сообщения об ошибках. Поэтому, несмотря на то, что переменная не существует, команда echo не отобразит никакой информации.

Не отключайте сообщения об ошибках, которые вы не понимаете. Убедитесь, что вы понимаете сущность ошибки и уверены, что это не приведет к неправильной работе приложения. Сообщение об ошибке указывает на то, что в сценарии содержится некорректность, которую нужно исправить. Это может быть опечатка в имени переменной, как в предыдущем примере.

Многие языки программирования требуют, чтобы перед использованием переменные были описаны. В этих языках использование переменной без ее описания — неустранимая ошибка, при наличии которой сценарий прекращает выполняться. Однако PHP не требует этого, что может несколько смутить специалистов, имеющих опыт программирования на языке Java или C.

Создание первого сценария с переменными

Сценарий, приведенный в главе 3, отображает на Web-странице приветствие, используя простую инструкцию echo. В этом разделе мы тоже создадим сценарий, отображающий приветствие, но с использованием переменной. В сценарии из главы 3 для вывода приветствия используется следующий фрагмент кода на PHP:

```
<?php
   echo "<p>Здравствуй, мир! </p>";
?>
```

Приведем полный текст сценария, содержащий раздел PHP с переменной и отображающий приветствие "Здравствуй, мир!":

```
<html>
<head> <title>Сценарий с использованием переменной</title></head>
<body>
<?php

   $salutation = "Здравствуй, мир!";
   echo "<p> $salutation </p>";
?>
</body>
</HTML>
```

Если этот сценарий загрузить в браузер, на Web-странице отобразится приветствие

```
Здравствуй, мир!
```

Переменная сохраняет свое значение в процессе выполнения всего сценария, а не только в отдельном PHP-разделе. Если переменной присвоить значение 5 в начале сценария, то и в конце сценария значение этой переменной будет равно 5 (если, конечно, вы не присвоите ей нового значения). Например, следующий сценарий содержит два отдельных PHP раздела:

```
<html>
<head><title>Сценарий с использованием переменной</title></head>
<body>
<?php
   $salutation = "Здравствуй, мир!";
   echo "<p> $salutation </p>";
?>
<p>Это раздел HTML</p>
<?php
   echo "<p>$salutation еще раз</p>";
?>
</body>
</HTML>
```

Если URL-адрес этого сценария указать в браузере, то на Web-странице отобразится следующая информация:

```
Здравствуй, мир!
Это раздел HTML
Здравствуй, мир! еще раз
```

Более подробно об операторах вывода

Как упоминалось в главе 3, оператор `echo` имеет следующий формат:

```
echo элемент_вывода1,элемент_вывода2,элемент_вывода3,...
```

Вместо любого элемента вывода можно использовать переменную. Например, можно создать такой PHP-раздел:

```
<?php
   $first_name = "Кларк";
   $last_name = "Кент";
   echo "Меня зовут ",$first_name," ",$last_name;
?>
```

Приведем результат выполнения кода этого раздела:

```
Меня зовут Кларк Кент
```

Обратите внимание на пробел между переменными `$first_name` и `$last_name`. Если не добавить этот пробел, эти две переменные будут выводиться слитно:

```
Меня зовут КларкКент
```

Для получения желаемого вывода операторы, содержащие несколько переменных, должны удовлетворять определенным правилам форматирования. В табл. 4.1 показаны некоторые операторы `echo`, которые содержат переменные, а также результаты их использования. В таблице в операторах `echo` используются следующие значения переменных:

```
$number = 123;
$word1 = "Здравствуй,";
$word2 = "мир!";
```

Обратите внимание, что во второй строке таблицы между двумя переменными отсутствует пробел, поэтому нет никакого пробела и при выводе. В третьей строке между значениями переменных отображается пробел.

Таблица. 4.1. Операторы echo с переменными

Оператор echo	Результат вывода
`echo $number;`	`123`
`echo $word,$word2;`	`Здравствуй,мир!`
`echo $word1," ",$word2;`	`Здравствуй, мир!`
`echo $word1 $word2;`	Неверно, потому что имена переменных не отделены друг от друга запятыми. Это приведет к генерации сообщения об ошибке
`echo "$word1 $word2 снова";`	`Здравствуй, мир ! снова`

В некоторых операторах echo интерпретатор PHP не может отделить имя переменной, если сразу за ним следует текст. В подобных случаях имя переменной необходимо заключать в фигурные скобки. Например, предположим, что в сценарии используются следующие строки:

```
$type = "bird";
echo "Keep the $typecage clean";
```

Вместо желаемого результата будет получено следующее сообщение:

```
Notice: Undefined variable: typecage in testvar.php on line 6
```

После уведомления о проблеме будет выведен такой текст:

```
Keep the clean
```

Для того чтобы этот код работал правильно, необходимо использовать оператор echo следующего формата:

```
echo "Keep the {$type}cage clean";
```

тогда будет получен следующий результат:

```
Keep the birdcage clean
```

Использование переменных переменных

PHP позволяет использовать динамические имена переменных, получившие название *переменных переменных*. Переменную можно описать, используя значение, сохраненное в другой переменной, т.е. одна переменная может содержать имя другой переменной. Предположим, например, что вы хотите создать переменную $city со значением Лос-Анджелес. Для этого можно использовать следующий синтаксис:

```
$name_of_the_variable = "city";
```

В этой строке создается переменная, содержащая имя, которое можно назначить другой переменной. Для этого необходимо использовать операторы

```
$$name_of_the_variable = "Лос-Анджелес";
```

Обратите внимание на дополнительный символ доллара в начале имени переменной. Это и есть обозначение переменной переменной. Этот оператор создает новую переменную, именем которой является значение, содержащееся в переменной $name_of_the_variable:

```
$city = "Лос Анджелес";
```

Значение переменной $name_of_the_variable при этом не изменится.

Следующий пример иллюстрирует особенности работы с такими переменными. Операторы приведенного сценария вряд ли окажутся очень полезными, поскольку можно придумать лучший вариант решения данной задачи. Истинное преимущество переменных переменных становится понятным, когда они используются с массивами и циклами. Эти вопросы обсуждаются в главах 6 и 7.

Предположим, необходимо создать несколько переменных, имена которых совпадают с названиями городов, а их значениями является количество людей, населяющих соответствующий город. Для этого можно использовать такой код:

```
$Reno = 360000;
$Pasadena = 138000;
$cityname = "Reno";
echo "Население $cityname - ${$cityname}";
$cityname = "Pasadena";
echo "Население $cityname - ${$cityname}";
```

Результатом выполнения этого кода является следующий текст:

```
Население Reno - 360000
Население Pasadena - 138000
```

Обратите внимание, что в операторе `echo` имена переменных необходимо заключать в фигурные скобки. Без использования фигурных скобок вы получите совсем другой результат.

Удаление переменных

Удалить информацию из переменной можно с помощью оператора

```
$age =__;
```

Этот оператор удаляет значение переменной `$age`. Причем это не означает, что теперь переменная `$age` установлена в значение `0`. Это лишь значит, что переменная `$age` не содержит никакой информации. Технически это означает, что значением `$age` является строка нулевых символов. Если вы попытаетесь вывести значение `$age` на экран с помощью инструкции `echo`, то на экран ничего не выведется. (Сообщение об ошибке при этом тоже не выводится.)

Переменную можно удалить и следующим образом:

```
unset($age);
```

После выполнения этого оператора переменная `$age` больше не будет существовать. Если вы попробуете вывести ее с помощью оператора `echo`, то получите сообщение об ошибке `"undefined variable"` (неопределенная переменная). Таким образом, можно одновременно удалить несколько переменных:

```
unset($age, $name, $address);
```

Работа с константами

Константы подобны переменным. Константам дают имена и присваивают значения. Однако константы являются постоянными; их значения не могут быть изменены в сценарии. После того как значение константы установлено, оно уже не меняется. Если ввести константу для обозначения погоды и установить для нее значение "солнечная", то впоследствии изменить это значение нельзя.

Создание констант

Константы описываются с помощью оператора `define`. Общий формат этого оператора таков:

```
define("имя_константы", "значение_константы");
```

Например, чтобы создать константу `weather`, используйте следующий синтаксис:

```
define("weather", "солнечная");
```

Эта команда создает константу с именем `weather` и значением `солнечная`.

При описании констант, как и при объявлении переменных, нужно использовать осмысленные имена. Однако, в отличие от переменных, имена констант не начинаются со знака доллара. В соответствии с негласным соглашением константам присваивают имена, состоящие из прописных букв латинского алфавита, чтобы их можно было легко отличать от переменных. Однако в языке PHP можно использовать и символы нижнего регистра.

Константа может содержать строку или число. Следующий оператор создает константу с именем INTEREST и присваивает ей значение .01:

```
define("INTEREST",.01);
```

Константам нельзя давать имена, которые являются ключевыми словами PHP. *Ключевые слова* — это имена операторов PHP, например echo. Их нельзя использовать в качестве имен констант, потому что интерпретатор PHP обрабатывает их как команды PHP-сценария, а не как константы. PHP позволит описать константу с именем ECHO, не выдавая сообщения об ошибке, но при использовании такой константы возникнут проблемы. Например, если написать оператор вида

```
echo ECHO;
```

интерпретатор PHP запутается и выведет сообщение об ошибке. Он рассматривает константу как начало следующего оператора echo, но не может найти аргументы, чтобы завершить первую инструкцию echo.

В набор ключевых слов PHP включены следующие:

```
and             echo            list
as              else            new
break           empty           or
case            eval            print
class           exit            require
const           for             return
continue        foreach         switch
declare         function        use
default         global          var
die             if              while
do              include
```

Если вас беспокоит некоторый код, который выглядит хорошо, но отказывается корректно работать даже после многочисленных изменений, попробуйте изменить имя константы. Возможно, проблема состоит в том, что в качестве имени константы используется какое-то ключевое слово. Такое случается редко, но все же случается.

Следует отметить, что ключевые слова вполне можно использовать в качестве имен переменных, поскольку имена переменных начинаются с символа $. Однако лучше этим не пользоваться, так как такой стиль может запутать пользователей, которые будут работать с этим сценарием в дальнейшем.

Когда использовать константы

Если вы знаете, что значение какой-либо переменной не будет изменяться на протяжении всего сценария, рационально использовать для его хранения константу. Применение константы позволяет выбрать длинное осмысленное имя и сделать сценарий более ясным. Например, название PRODUCT_COST более понятно, чем число 20.50.

При использовании константы ее значение можно установить только один раз — в начале сценария. Если это значение когда-либо понадобится изменить, то при использовании константы это можно сделать только в одном месте, а не изменять значение переменной в двадцати различных местах сценария. Одна замена лучше, чем двадцать. Такой подход очень удобен и ускоряет работу.

Использование константы гарантирует, что данное значение не будет случайно изменено где-либо в сценарии.

Предположим, имеется сценарий, который переводит значения из одной валюты в другую, умножая количество долларов на обменный курс. Например, если курс обмена доллара США на канадский доллар составляет `1.52`, можно записать следующий код:

```
<?php
   $US_dollars = 20.00;
   $CA_dollars = $US_dollars * 1.52;
?>
```

Теперь предположим, что этот сценарий содержит 40 000 строк кода и что доллары США в нем конвертируются в канадские доллары в 50 различных местах. Значит, приведенный выше код встречается в 50 различных местах. Вы понимаете, что обменный курс изменяется практически каждую неделю, значит, каждую неделю вам придется редактировать этот сценарий, вручную изменяя число `1.52` на другое значение в 50 различных местах. Согласитесь, что это очень утомительно.

Гораздо лучше поместить обменный курс в переменную, чтобы изменять это значение только в одном месте. Тогда сценарий будет выглядеть следующим образом:

```
<?php
   $rate = 1.52;

   $US_dollars = 20.00;
   $CA_dollars = $US_dollars * $rate;
?>
```

Значение `$rate` можно установить в начале сценария, а затем использовать его в 50 необходимых местах сценария. Естественно, это более удобная реализация. Если потребуется изменить курс, то значение переменной потребуется поменять только в одном месте. Например, если на следующей неделе обменный курс изменится на `1.53`, вам придется изменить лишь первую строку сценария:

```
$rate = 1.53;
```

Такой подход вполне пригоден. Однако `$rate` — не очень осмысленное имя. Вспомните, что сценарий содержит 40 000 строк кода и 2 строки, определяющие процедуру конвертирования валюты, которые используются в 50 различных местах. Предположим, где-то в середине сценария необходимо добавить некоторый код, определяющий степень заинтересованности. Предположим, где-нибудь в середине сценария вы случайно используете следующий код:

```
$interest_rate = 20;
$rate = $interest_rate-1;
$amount = $principal * $rate;
```

Теперь всюду после этого фрагмента значение переменной `$rate` изменится. Значение `1.52`, установленное в начале сценария, будет заменено числом `19`, определяемым этим кодом. Этого можно избежать, выбирая более осмысленные имена переменных. А еще лучше использовать константу, как в следующем сценарии:

```
<?php
   define("RATE", 1.52);

   $US_dolars = 20;
   $CA_dollars = $US_dollars * RATE;
?>
```

Теперь вы используете константу `RATE`, которую нельзя изменить в сценарии. Если в середине сценария вы попробуете добавить строку

```
RATE = 20;
```

интерпретатор PHP не позволит изменить значение. Таким образом можно избежать ошибок, которые легко допустить при использовании переменных.

На следующей неделе, когда обменный курс изменится на 1.53, этот сценарий придется лишь слегка отредактировать следующим образом:

```
<?php
   define("RATE", 1.53);
   $US_dollars = 20
   $CA_dollars = $US_dollars * RATE;
?>
```

Конечно, лучше использовать более осмысленное имя, например:

```
define("US_TO_CA", 1.52);
```

Имейте в виду, что ошибки, которые невозможно допустить при работе с простыми сценариями, состоящими из десяти строк, вполне возможны при работе со сценариями, включающими тысячи строк кода, особенно, если такой сценарий разрабатывает целая команда программистов.

Если вы уверены, что некоторое значение не будет изменяться на протяжении всего сценария, используйте константу. Если значение изменяется в процессе работы сценария, используйте переменную.

Отображение значений констант

Значение константы можно вывести с помощью функции print_r следующим образом:

```
print_r(US_TO_CA);
```

Константу также можно вывести с помощью оператора echo:

```
echo US_TO_CA;
```

Если для вывода константы используется оператор echo, имя константы не нужно заключать в кавычки. Если имя константы все же заключить в кавычки, то на экране отобразится имя константы, а не ее значение. Константу можно вывести, как показано в предыдущем примере, или же с использованием круглых скобок. Можно формировать и более сложные команды вывода, разделяя выводимые объекты запятыми, как в следующем примере:

```
echo "Курс обмена на канадский доллар составляет $",US_TO_CA;
```

В результате выполнения этой команды будет выведен следующий текст:

```
Курс обмена на канадский доллар составляет $1.52
```

Использование встроенных констант

В языке PHP определено множество встроенных констант, которые можно использовать в сценариях. Для примера, значением константы __LINE__ является номер текущей строки, а константа __FILE__ содержит имя соответствующего файла. (Встроенные константы начинаются с двух символов подчеркивания и заканчиваются тоже двумя символами подчеркивания.) Например, можно использовать команду

```
echo __FILE__;
```

ее результат выглядит следующим образом:

```
c:\program files\apache group\apache\htdocs\testvar2.php
```

В языке PHP имеется и множество других встроенных констант. Например, константы E_ALL и E_ERROR можно использовать для управления процессом обработки ошибок в PHP. Эти константы описаны в следующем разделе.

Обработка сообщений об ошибках

При возникновении проблем с выполнением сценария интерпретатор PHP выдает сообщения об ошибках. Существуют следующие типы сообщений.

- ✓ **Сообщение об ошибке** (error message). Это сообщение выдается при возникновении проблем, препятствующих выполнению сценария. Сценарий отображает сообщение об ошибке и прекращает свою работу. Чтобы помочь идентифицировать проблему, такое сообщение должно содержать как можно больше информации. Вот пример типичного сообщения об ошибке:

  ```
  Parse error: parse error in c:\test.php on line 6
  ```

  Такие сообщения об ошибках часто выдаются при отсутствии символа-разделителя (точки с запятой, кавычек или скобок).

- ✓ **Предупреждение** (warning message). Предупреждающее сообщение выдается в том случае, когда интерпретатор PHP встречает проблему, но эта проблема не препятствует выполнению сценария. Предупреждающие сообщения указывают на вероятную некорректность кода. Необходимо идентифицировать источник возникновения сообщения, а затем решить, нужно ли изменять сценарий. Например, если не указать имя переменной в качестве параметра функции `print_r` — написать `print_r()`, а не `print_r($varname)`, — появится следующее сообщение:

  ```
  Warning: print_r () expects at least 1 parameter, 0 given in
        d:\test1.php on line 9
  ```

  Поскольку это предупреждение, а не сообщение об ошибке, то после выполнения функции `print_r` работа сценария продолжится. Предупреждение обычно указывает на более серьезную проблему, а не содержит простое уведомление. В этом случае необходимо устранить проблему.

- ✓ **Уведомление** (notice). Такое сообщение выводится в том случае, когда в сценарии PHP встречается конструкция, которая может быть и ошибочной, и правильной. Обычно уведомление выводится в ответ на попытку отображения на экране несуществующих переменных. Рассмотрим пример:

  ```
  Notice: Undefined variable: age in testing.php on line 9
  ```

Сообщения об ошибках, предупреждения и уведомления содержат имя проблемного файла и номер строки, в которой возникла проблема.

Типы отображаемых сообщений об ошибках зависят от установленного уровня проверки ошибок. Необходимо видеть все сообщения об ошибках, но далеко не все предупреждения и уведомления. (Зачастую единственная проблема с уведомлениями — это сами уведомления; при этом код работает правильно.) Можно сделать так, чтобы предупреждающие сообщения и уведомления отображались только во время отладки сценария, но не после того, как клиенты начнут использовать приложение. Все сообщения об ошибках можно записывать в журнал, чтобы они не выводились для пользователей, а вы могли их потом просмотреть.

Изменение уровня проверки ошибок для Web-узла

Уровень проверки ошибок для Web-узла определяется в файле `php.ini`. Если вы являетесь администратором модуля PHP и имеете доступ к файлу `php.ini`, то сможете изменить уровень проверки ошибок. Если вы не являетесь администратором, то можете изменить уровень обработки ошибок для каждого сценария в отдельности, как будет описано в следующем

разделе. (Более детальную информацию о настройках, содержащихся в файле php.ini, можно найти в приложении А.)

Для того чтобы определить текущий уровень обработки ошибок, откройте файл php.ini в текстовом редакторе и найдите строку, подобную следующей:

```
error_reporting = E_ALL; display all errors, warnings and notices
(выводить все ошибки, уведомления и предупреждения)
```

Эта инструкция включает обработку всех типов ошибок и вывод сообщений о них. Этот режим полезен при разработке сценария. Однако после передачи сценария пользователям предупреждения и уведомления отображать не стоит.

Обратите внимание на то, что в приведенном выше примере точка с запятой находится после значения E_ALL, а не в начале строки. Точка с запятой — это символ, обозначающий комментарий в файле php.ini. Поэтому текст в строке после точки с запятой — только комментарий, а не часть команды. Если бы точка с запятой располагалась в начале строки, то комментарием считалась бы вся строка и данная команда не выполняла бы никаких действий.

При внимательном изучении файла php.ini вы, вероятно, найдете несколько закомментированных команд. Эти инструкции включены как примеры, а не как выполняемые команды. Инструкции файла php.ini без точки с запятой в начале строки являются действующими.

E_ALL — это встроенная константа PHP, которая определяет все ошибки, предупреждения и уведомления; E_NOTICE — встроенная константа, определяющая уведомления. Эти две константы можно использовать в следующей конструкции:

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

Наличие E_ALL требует отображения всех сообщений об ошибках, предупреждений и уведомлений. Однако использование второго параметра ~E_NOTICE приводит к тому, что уведомления отображаться не будут. В результате будут отображаться только сообщения об ошибках и предупреждения. Этот метод позволяет очень легко определить уровень проверки ошибок, не задавая перечень всех типов ошибок, которые вы хотите отобразить.

Две инструкции, приведенные в этом разделе, используются наиболее часто. Чтобы установить уровень обработки ошибок, можно использовать и другие константы. Однако констант E_ALL и E_NOTICE обычно достаточно для большинства сценариев. Список всех констант можно найти в конфигурационном файле php.ini. Для получения более детальной информации об установке уровня обработки ошибок прочтите интерактивное руководство по языку PHP.

Можно отказаться от вывода сразу всех сообщений об ошибках. Иногда нежелательно показывать пользователям сгенерированные интерпретатором PHP сообщения об ошибках или предупреждающие сообщения, поскольку они могут содержать компрометирующую информацию. Обычно при тестировании сценария сообщения об ошибках сохраняются в файле журнала. Эта процедура будет описана в этой главе ниже.

Чтобы отключить вывод сообщений об ошибках, найдите в файле php.ini строку display_errors = On и измените значение параметра On на Off.

Для того чтобы внесенные изменения вступили в силу, после любой модификации файла php.ini необходимо перезапустить Web-сервер.

Изменение уровня проверки ошибок в сценарии

Если вы хотите установить уровень проверки ошибок для конкретного сценария, добавьте в начало сценария команду

```
error_reporting(ОПЦИИ);
```

ОПЦИИ — это одна из встроенных констант, рассмотренных в предыдущем разделе. Например, добавив в начало сценария следующую инструкцию, можно обеспечить вывод всех сообщений об ошибках, предупреждений и уведомлений:

```
error_reporting(E_ALL);
```

Предположим, что в файле `php.ini` уровень обработки ошибок установлен в значение `E_ALL`. Такой уровень обработки ошибок может вполне подходить для разработки сценариев, но после передачи сценария пользователям желательно отменить вывод уведомлений. Чтобы отменить настройки, заданные в файле `php.ini`, можно добавить к готовому сценарию инструкцию

```
error_reporting(E_ALL & ~E_NOTICE);
```

Можно так определить уровень обработки ошибок, чтобы не отображались никакие сообщения. Для этого используйте команду

```
error_reporting(0);
```

В готовых отлаженных сценариях вывод предупреждений и сообщений об ошибках обычно отключают. Не нужно, чтобы пользователи видели сообщения об ошибках, которые выдает интерпретатор PHP, потому что информация в таких сообщениях может представлять угрозу для безопасности сервера. Однако вы можете захотеть самостоятельно просматривать любые сообщения об ошибках. Используя данную функцию с параметром `0`, можно отключить отображение сообщений об ошибках, но одновременно записывать такие сообщения в файл журнала. Пользователи не увидят сообщений, а вы сможете их просматривать. Запись сообщений в файл журнала описана в следующем разделе.

Запись сообщений в файл журнала

Сообщения об ошибках и предупреждения интерпретатора PHP можно записывать в файл журнала. При этом наряду с записью сообщений в журнал можно отображать или не отображать их на экране для обычных пользователей.

В журнал можно записывать сообщения об ошибках для всего узла. Для этого следует внести изменения в файл `php.ini`, если вы имеете к нему доступ. Откройте файл `php.ini` и найдите строку

```
log_errors = Off
```

Значение `Off` необходимо изменить на `On`. Нужно также указать, куда записывать сообщения об ошибках. Чтобы сделать это, найдите строку

```
;error_log = имя_файла
```

Теперь удалите точку с запятой в начале строки, превратив комментарий в действующую инструкцию. Замените значение *имя_файла* на путь к файлу, в который вы хотите сохранять сообщения об ошибках. Например, можно использовать команду

```
error_log = c:\temp\php_error_log
```

Необходимо указывать существующий каталог (в Windows он часто называется также папкой). Для того чтобы сообщения об ошибках можно было записывать в указанный файл, необходимо создать каталог `c:\temp`. При этом файл создавать не обязательно. Если интерпретатор PHP сможет найти указанный каталог, то он самостоятельно создаст требуемый файл.

Чтобы внесенные в файл `php.ini` изменения вступили в силу, необходимо перезапустить Web-сервер.

Расширенная обработка ошибок

В этом разделе описываются дополнительные возможности обработки ошибок. Новичкам совсем не обязательно читать этот раздел. Его следует изучить только после того, как вы освоите методы программирования на языке PHP, описанные в оставшейся части книги.

Стандартных сообщений PHP об ошибках может оказаться недостаточно. Например вы чувствуете, что сценарий работает неправильно, хотя интерпретатор PHP не обнаруживает никаких ошибок. Допустим, вы пишете сценарий проектирования дома. В таком случае, если высота двери, определяемая переменной `$height_of_door`, больше высоты дома `$height_of_house`, — это явно неправильно. Вам это известно, а интерпретатору PHP — нет. Он не распознает эту ситуацию как ошибку. Чтобы интерпретатор PHP проверял эту ошибку в сценарии, можно написать следующий код:

```
if ($height_of_door > $height_of_house)
{
   trigger_error("Impossible condition", E_USER_ERROR);
}
```

Оператор `if` более подробно рассматривается в главе 7.

Параметр `E_USER_ERROR` свидетельствует о том, что условие является ошибочным. Строка `Impossible condition` является сообщением, которое будет отображаться при возникновении данной ошибки. Если условие истинно, будет выводиться следующее сообщение:

```
Fatal error: Impossible condition in d:\testerr.php on line 9
```

Сценарий остановится в этой точке, поскольку интерпретатору PHP было указано, что это ошибка, а не предупреждение или уведомление. Если в качестве параметра вместо `E_USER_ERROR` указать `E_USER_WARNING` или `E_USER_NOTICE`, то PHP будет обрабатывать данную ситуацию как предупреждение или уведомление соответственно.

Если вы хотите определить свой собственный способ обработки ошибок, а не использовать стандартные процедуры PHP, то можно написать собственные операторы передачи сообщений, записи в журнал, отправки электронной почты или остановки сценария. Например, с помощью следующего кода можно вывести пользователю сообщение об ошибке и прервать выполнение сценария:

```
if ($height_of_door > $height_of_house)
{
   echo "Это невозможно<br>";
   exit();
}
```

Если значение `$height_of_door` больше значения переменной `$height_of_house`, будет выведено сообщение об ошибке и функция `exit()` прервет выполнение сценария. Никакие операторы больше выполняться не будут.

Сообщение об ошибке можно записать в файл журнала, воспользовавшись функцией

```
error_log(message, 3, имя_файла_журнала);
```

Например, можно использовать следующий блок `if`:

```
if ($height_of_door > $height_of_house)
{
   error_log ("Дверь выше дома", 3, "/temp/err_log");
   exit();
}
```

После добавления в сценарий этого оператора, если значение `$height_of_door` превышает `$height_of_house`, в файл журнала `/temp/err_log` будет записано сообщение `Дверь выше дома`. Значение `3` в этом операторе означает запись сообщения в указанный

файл журнала. При этом каталог /temp должен существовать. Если файла журнала не существует, интерпретатор PHP создаст его самостоятельно.

В качестве альтернативы в случае ошибки можно отправить себе почтовое сообщение. Для этой цели, так же как и для записи сообщения об ошибке в файл журнала, можно использовать оператор error_log. Значение 1 в списке параметров функции error_log означает передачу сообщения по указанному адресу электронной почты:

```
error_log ("Дверь выше дома", 1, "me@mymail.com");
```

Использование этого оператора предполагает, что сообщение электронной почты можно отправить из PHP-сценария. Более подробно этот вопрос рассматривается в главе 13.

С другой стороны, стандартные ошибки PHP можно перехватывать и обрабатывать по-своему. Можно определить операторы, которые будут выполняться при возникновении ошибки. Например, можно потребовать, чтобы интерпретатор PHP отображал созданное вами сообщение или выполнял требуемые инструкции. Допустим, вам захотелось получать сообщения об ошибках по электронной почте или открывать и закрывать некоторые файлы перед завершением выполнения сценария.

Для обработки ошибок можно написать собственный код и потребовать от интерпретатора PHP использовать его всякий раз при возникновении ошибок. Чтобы это реализовать, необходимо написать код обработки ошибок и сохранить его как *функцию*. Потом можно вызывать эту функцию всякий раз при возникновении ошибок. (Принципы создания функций описаны в главе 8.) Чтобы при обработке ошибок интерпретатор PHP использовал созданную функцию, а не стандартную процедуру, воспользуйтесь функцией

```
set_error_handler(имя_функции);
```

Например, можно использовать следующую команду:

```
set_error_handler(my_error_handler);
```

Детальные рекомендации по созданию обработчика ошибок my_error_handler содержатся в главе 8.

Еще один метод обработки ошибок средствами языка PHP состоит в использовании функции die, которая выводит сообщение при некорректной работе функции. Этот вопрос подробно обсуждается в главе 8.

Глава 5

Работа с данными

В этой главе...

- Что такое типы данных
- Выполнение арифметических операций
- Обработка текстовых строк
- Использование даты и времени

Переменные могут содержать значения различных типов. В свою очередь, над данными различных типов можно выполнять различные операции. Например, можно складывать два числа — `1+2`. Однако сложение двух символов (`a+b`) выполняется совсем по-другому. В этой главе вы узнаете, какие типы данных существуют в языке PHP и как их можно использовать.

Типы данных

В переменных PHP можно хранить данные следующих простых типов.

- ✓ **Целый тип (integer)** позволяет оперировать с целыми числами (без дробной части), такими как `-43`, `0`, `1`, `27` или `5438`. Допустимый диапазон целочисленных значений в общем случае зависит от операционной системы. Однако обычно допустимые значения переменных целочисленного типа лежат в пределе от –2 до 2 миллиардов.
- ✓ **Тип с плавающей точкой (floating point)** позволяет манипулировать числами с дробной частью, например `5.24` или `123.456789`. Такие числа часто называются *действительными* (real number) или *числами с плавающей точкой* (float).
- ✓ **Строковый тип (string)** обеспечивает хранение последовательности символов, например `привет`. На длину текстовой строки практически не накладывается каких-либо ограничений.
- ✓ К **булевому (логическому) типу (boolean)** относятся два значения: `TRUE` (истина) и `FALSE` (ложь). Данный тип более подробно рассматривается ниже.

Присваивание типов данных

В большинстве языков программирования требуется, чтобы перед использованием каждая переменная была предварительно связана с типом. Однако язык PHP является гораздо менее формальным. В нем нет необходимости явно определять тип переменных. Интерпретатор PHP самостоятельно проверяет значение, присвоенное некоторой переменной, и после этого связывает с ней соответствующий тип данных. В общем случае это является очень полезным. При этом следует отметить, что процедура автоматического определения типа переменной работает достаточно точно.

Истина или ложь: булевы значения

Переменные логического типа могут содержать два значения: `TRUE` (истина) и `FALSE` (ложь). В основном они используются в условных выражениях. Например, результатом обработки выражения `$a>$b` будет либо `TRUE` (истина), либо `FALSE` (ложь).

В языке PHP ложными считаются следующие значения.

- строка `"FALSE"` (состоящая из символов как в верхнем, так и в нижнем регистрах);
- целое значение `0`;
- значение с плавающей точкой `0.0`;
- пустая строка;
- строка, содержащая единственный символ `"0"`;
- константа `NULL`.

Все остальные значения соответствуют логическому значению `TRUE`. При выводе булевской переменной с помощью функции `echo` отобразится пустая строка, если в этой переменной содержится значение `FALSE`, и `1` — в противном случае. Логические значения зачастую применяются в качестве возвращаемых значений функций. Это позволяет определить, успешно ли было завершено ее выполнение. Вопросы совместного использования функций и логических переменных более подробно рассматриваются в главе 8.

По мере необходимости в языке PHP автоматически выполняется и преобразование типов. Например, в следующем фрагменте требуемое преобразование выполняется без малейших проблем:

```
$firstNumber = 1;    # Хранится как целое
$secondNumber = 1.1;# Хранится как число с плавающей точкой
$sum = $firstNumber+$secondNumber;
```

С формальной точки зрения, третье выражение лишено смысла, поскольку в нем используются операнды различных типов. Однако интерпретатор PHP выполняет преобразование целого значения в значение типа `float`, и суммирование двух переменных происходит без каких бы то ни было проблем. Следует заметить, что вся эта процедура выполняется автоматически.

Приведение типов

Иногда интерпретатору PHP не удается правильно определить тип переменной. Тогда при ее использовании будет выдана ошибка об использовании неверного типа. В этом случае нужно самостоятельно определить тип переменной. Такая операция называется *приведением типов* (type casting). Явно задать тип можно следующим образом:

```
$newint = (int)$var1;
$newfloat = (float)$var1;
$newstring = (string)$var1;
```

Значение переменной, расположенной справа от символа равенства, присваивается переменной с указанным типом, находящейся слева. Так, значение `$var1` присваивается переменной `$newint` с типом `integer` (`(int)`).

При приведении типов будьте очень внимательны. Иногда это может привести к непредсказуемым результатам. Например, при преобразовании действительного значения в целое теряется дробная часть. Например, если в переменной `$number` содержится значение `1.8`, то после его преобразования в целое — `$newnumber = (int)$number` — в переменной `$newnumber` будет содержаться значение `1`.

Определить тип переменной можно с помощью функции

```
var_dump($myvariable);
```

Например, следующее выражение позволяет определить тип переменной $checkvar:

```
var_dump($checkvar);
```

В качестве результата будет получено выражение int(27), что свидетельствует о том, что в переменной $checkvar содержится целочисленное значение 27.

Работа с числами

Типы данных float и integer являются числовыми. Инициализацию переменных этих типов можно осуществить таким образом:

```
$intvar =3;
$floatvar =9.3;
```

При этом интерпретатор PHP автоматически разместит заданные значения в оперативной памяти с учетом их типа.

Выполнение математических операций

Язык PHP позволяет выполнять над числовыми переменными различные операции. При этом необходимо задать два операнда и соответствующий символ математической операции. Например, операцию сложения (+) можно выполнить следующим образом:

```
1+2
```

То же самое можно осуществить и с переменными:

```
$var1+$var2;
```

При использовании чисел в математических операциях их не следует заключать в кавычки. В противном случае они будут рассматриваться как строковые переменные, над которыми нельзя выполнять необходимые арифметические операции. Однако в отличие от других языков программирования в PHP по необходимости строки автоматически преобразуются в числовой формат. Например:

```
$var1 = "1";
$var2 = 2;
$total = $var1+$var2;
```

Формально переменные $var1 и $var2 просуммировать нельзя, поскольку в $var1 содержится текстовая строка. Однако при обработке этого выражения интерпретатор PHP автоматически преобразует строку "1" в числовое значение 1 и корректно выполнит эту операцию.

В следующем фрагменте также выполняется автоматическое преобразование текстовой строки в число, однако конечный результат уже не так очевиден:

```
$var1 = "x";
$var2 = 2;
$total = $var1+$var2;
```

Поскольку PHP не может преобразовать символ "x" в числовое значение, то при выполнении сложения вместо него используется значение 0. Таким образом, результат сложения переменных $var1 и $var2 будет равен 2. Преобразование типов далеко не всегда приводит к получению желаемых результатов. Безусловно, автоматическое приведение типов является очень удобным и позволяет сэкономить массу усилий. Однако нужно соблюдать осторожность, поскольку оно может привести к получению непредвиденных результатов, как показано в предыдущем примере.

Иногда интерпретатору PHP не удается корректно обработать выражения, которые понятны человеку. Например:

```
$var1 = "2,000";
$var2 = 2;
$total = $var1+$var2;
```

Хотя человеку и понятно назначение запятой в значении первой переменной, интерпретатору PHP — нет. Он преобразует строку `"2,000"` в числовое значение 2, и в результате после выполнения сложения в переменной `$total` будет содержаться значение 4.

В табл. 5.1 приведены основные математические операции языка PHP.

Таблица 5.1. Математические операции PHP

Операция	Описание
+	Сложение двух чисел
-	Вычитание второго числа из первого
*	Умножение двух чисел
/	Деление первого числа на второе
%	Остаток от деления (или *деление по модулю*). Например, в результате вычисления выражения `$a = 13%4` в переменной `$a` будет содержаться значение 1

Порядок выполнения операций

Несколько математических операций можно выполнять одновременно. Например, в следующем выражении используется сразу три операции:

```
$total = 1+2*3+1;
```

Порядок выполнения операций очень важен, поскольку от этого зависит конечный результат. В языке PHP сначала выполняются операции умножения и деления и лишь потом сложение и вычитание. При равном приоритете операций действия выполняются слева направо. Так, в приведенном выше примере значение переменной `$total` становится равным 8.

```
$total = 1+2*3+1 #сначала выполняется умножение
$total = 1+6+1   #затем - левая операция сложения
$total = 7+1     #затем - сложение справа
$total = 8
```

Изменить последовательность выполнения арифметических операций можно с помощью круглых скобок. Тогда математические операции в скобках будут выполняться в первую очередь. Например, предыдущее выражение можно переписать в следующем виде:

```
$total = (1+2)*3+1;
```

После выполнения всех математических преобразований в переменной `$total` будет содержаться значение 10:

```
$total = (1+2)*3+1 #сначала выполняется сложение в скобках
$total = 3*3+1     #затем умножение
$result = 9+1      #и наконец сложение
$result = 10
```

Порядок вычислений в скобках соответствует общим правилам выполнения арифметических операций. Например, в выражении `(3+2*5)` сначала выполняется умножение, а затем — сложение. Естественно, внутри скобок можно использовать другие скобки и таким образом изменять последовательность выполнения операций.

Лучше руководствоваться поговоркой "Семь раз отмерь — один раз отрежь" и использовать скобки во всех случаях, когда арифметическое выражение выглядит неоднозначным.

Инкрементирование и декрементирование

Увеличить значение переменной на 1 можно следующим образом:

```
$counter = $counter+1;
```

Однако для этого можно воспользоваться и сокращенной конструкцией языка PHP:

```
$counter++;
```

Например, рассмотрим выражения

```
$counter=0;
$counter++;
echo $counter;
```

В результате в переменной `$counter` будет содержаться значение 1 (что и будет выведено на экран). Точно так же можно выполнить и вычитание:

```
$counter--;
```

Для увеличения значения переменной на 1 можно воспользоваться еще одним выражением: `"+=1"`. С помощью такой конструкции значение переменной можно изменить произвольным образом. Например:

```
$counter+=2;
$counter-=3;
$counter*=2;
$counter/=3;
```

В приведенных примерах выполняется увеличение значения переменной `$counter` на 2, его уменьшение на 3, умножение на 2 и деление на 3 соответственно.

Использование встроенных математических функций

Более сложные математические вычисления можно выполнять с помощью встроенных функций. (Функции более подробно рассматриваются в главе 8.) Например, если нужно найти квадратный корень некоторого значения, нет необходимости создавать новую функцию, так как она уже существует. Например:

```
$rootvar = sqrt(91);
$rootvar = sqrt($number);
```

В первом выражении вычисляется квадратный корень числа 91, а во втором — значения, содержащегося в переменной `$number`.

Для округления действительного числа к ближайшему большему целому можно воспользоваться функцией

```
$upnumber = ceil(27.63);
```

Результатом этого выражения будет 28. Существует аналогичная функция, которая округляет действительное значение к ближайшему меньшему целому. Например:

```
$downnumber = floor(27.63);
```

В этом случае функция `floor()` вернет значение 27.

В языке PHP определено и много других математических функций: для выполнения простых операций, например поиска максимума, минимума или генерирования случайных чисел, или более сложных — вычисления синуса, тангенса или преобразования чисел в двоичный или восьмеричный формат. Перечень математических функций вы найдете в приложении Б.

Форматирование чисел для вывода

Зачастую числа необходимо отображать в каком-то определенном формате, например использовать запятую для разделения тысяч или использовать два знака после запятой для обозначения

денежной суммы. Однако при использовании PHP числа хранятся и отображаются в наиболее эффективном формате. Так, если в переменной содержится число `10.00`, он будет выведено как `10`. Если же нужно вывести это число в другом виде, придется явно указать соответствующий формат.

Для определения формата в PHP предназначена функция `number_format()` с синтаксисом

`number_format(`*`число, количество_десятичных_знаков, "разделитель1", "разделитель2"`*`)`

и со следующими параметрами:

- ✓ *число*. Форматируемое число, которое является обязательным аргументом.
- ✓ *количество_десятичных_знаков*. Определяет количество знаков после запятой. Если этот параметр отсутствует, то по умолчанию его значение равно 0 и, таким образом, *число* округляется до ближайшего целого числа (т.е. отображается без дробной части). Если используются аргументы *разделитель1* и *разделитель2*, то *количество_десятичных_знаков* является обязательным параметром.
- ✓ *разделитель1*. Определяет символ, который будет использоваться в качестве символа-разделителя целой и дробной частей. По умолчанию таким символом является точка. Параметр *разделитель1* является обязательным, если используется параметр *разделитель2*.
- ✓ *разделитель2*. Определяет символ, который будет разделителем в целой части числа. По умолчанию в качестве такого символа используется запятая.

В табл. 5.2 приведено несколько примеров использования функции `number_format()`.

Таблица 5.2. Примеры использования функции `number_format()`

$number	Формат	Результат
12321	`number_format($number)`	12,321
12321.66	`number_format($number, 2)`	12,321.66
12321.66	`number_format($number)`	12,322
12321.6	`number_format($number, 3)`	12,321.600
12321	`number_format($number, 0, ".", ".")`	12.321
12321.66	`number_format($number, 2, ".", "")`	12321.66

После форматирования число конвертируется в текстовую строку. Поэтому все арифметические операции необходимо выполнить до форматирования.

Для более сложного форматирования в языке PHP предназначены функции `printf()` и `sprintf()`.

- ✓ Функция `printf()` позволяет напрямую выводить число в заданном формате.
- ✓ Функция `sprintf()` сохраняет форматируемое число в переменной.

Функции `printf()` и `sprintf()` одновременно позволяют форматировать как строки, так и числовые значения. Более подробно эти функции рассматриваются ниже, в разделе "Форматирование текстовых строк".

Работа со строками символов

К *символам* (character) относятся буквы, числа и знаки пунктуации. *Строка символов* (или *текстовая строка*) (character string) является последовательностью символов. Если числа используются в качестве символов, они сохраняются так же, как и буквы, т.е. над ними невозможно выполнять арифметические операции. Например, номер телефона обычно хранится как строка, поскольку над ним не нужно выполнять никаких математических действий.

Для присваивания переменной строкового значения используются одинарные или двойные кавычки. Это позволяет указать интерпретатору PHP начало и конец последовательности символов. Например, следующие два выражения идентичны:

```
$string = "Здравствуй, мир!";
$string = 'Здравствуй, мир!';
```

Работа с длинными текстовыми строками

В языке PHP имеется *heredoc*-механизм, позволяющий сохранять в переменных длинные последовательности символов, которые могут занимать несколько строк. Этот механизм позволяет указать начало и конец строки символов с использованием следующего синтаксиса:

$имя_переменной = <<<*МЕТКА*

текст

МЕТКА;

Здесь *МЕТКА* представляет собой произвольное имя. Если в переменной *$имя_переменной* необходимо сохранить некоторый текст, его нужно заключить между метками *МЕТКА*. Тогда при обработке этого выражения интерпретатор PHP сможет определить, где находится начало и конец строки, и разместит *текст* в переменной *$имя_переменной*.

В строке, созданной с помощью *heredoc*-механизма, могут содержаться переменные и специальные символы, как и в обычной строке в двойных кавычках. (Такие строки более подробно будут рассматриваться ниже, в разделе "Сравнение строк в одинарных и двойных кавычках".)

Ниже приведен пример создания строки с использованием *heredoc*-механизма.

```
$distance = 10;
$herevariable = <<<ENDOFTEXT
Расстояние между
Лос-Анджелесом и Пасаденой
составляет $distance миль.
ENDOFTEXT;
echo $herevariable;
```

При выводе значения переменной `$herevariable` на экран будет получено следующее:

```
Расстояние между
Лос-Анджелесом и Пасаденой
составляет 10 миль.
```

Однако при использовании таких строк нужно соблюдать осторожность. К выбору меток интерпретатор PHP предъявляет достаточно жесткие требования. При первом появлении *МЕТКА* (в данном примере ENDOFTEXT) должна находиться в конце первой строки так, чтобы за ней не было ни одного символа, даже пробела. То же самое касается и завершающей метки: за ней не должно быть никаких символов, кроме точки с запятой. Если эти правила не выполняются, символ конца строки не будет распознан, интерпретатор PHP продолжит поиск этого символа в оставшейся части сценария и в конечном счете будет сгенерировано сообщение об ошибке.

Использование в строках специальных символов

В языке PHP определены специальные символы, \n и \t, которые можно использовать в строках. Символ \n является символом перехода на новую строку. Например:

```
$string = "Здравствуй, \nмир";
echo $string;
```

В результате будет выведено следующее:

```
Здравствуй,
мир
```

Символ \t является символом табуляции. Например, использование фрагмента

```
$string = "Строка 1 \n\tстрока 2";
echo $string;
```

приведет к тому, что вторая строка будет выведена с отступом:

```
Строка 1
     строка 2
```

Специальные символы можно использовать только в строках, заключенных в двойные кавычки. При использовании одинарных кавычек с этими символами *не будет связано никакого особого смысла. Различия между этими двумя представлениями* строк рассматриваются в следующем разделе.

Сравнение строк в одинарных и двойных кавычках

Строки в одинарных и двойных кавычках обрабатываются интерпретатором PHP по-разному.

- ✓ Последовательность символов, заключенная в одинарные кавычки, хранится "как есть", за исключением символа \', который хранится как символ апострофа. (Более подробно этот вопрос рассматривается в следующем разделе.)
- ✓ В строках в двойных кавычках обрабатываются переменные и специальные символы, и лишь после этого может использоваться сама строка.

В следующих примерах проиллюстрированы различия между строками в одинарных и двойных кавычках.

Если имя переменной разместить в двойных кавычках, интерпретатор PHP будет использовать ее значение, а если в одинарных — ее имя. Например:

```
$name = "Сэм";
$output1 = "$name";
$output2 = '$name';
echo $output1;
echo $output2;
```

При выполнении этого выражения будет получен следующий результат:

```
Сэм
$name
```

Аналогично обрабатываются и специальные символы. Если их разместить в строке с двойными кавычками, то специальные символы будут интерпретироваться, а если в строке с оди-нарными кавычками — нет. Эти отличие проиллюстрированы в следующем примере:

```
$string1 = "Строка в \n\tдвойных кавычках";
$string2 = 'Строка в \n\tодинарных кавычках';
```

Первая строка отобразится так:

```
Строка в
    двойных кавычках
```

а вторая — так:

```
Строка в \n\тодинарных кавычках
```

Наличие разных кавычек определяет, каким образом будут обрабатываться другие кавычки в текстовой строке. Например:

```
$number = 10;
$string1 = "В очереди находится '$number' людей.";
$string2 = 'В очереди находится "$number" людей.';
echo $string1, "\n";
echo $string2;
```

При обработке этого выражения будет получен следующий результат:

```
В очереди находятся '10' людей.
В очереди находятся "$number" людей.
```

Обратите внимание, что, несмотря на то, что в строке `$string1` имя переменной `$number` заключено в одинарные кавычки, двойные кавычки приводят к интерпретации ее значения, а не имени. При выводе второй строки происходит обратное. Поскольку вся строка заключена в одинарные кавычки, то она выводится без интерпретации переменных, имена которых включены в нее.

Сокрытие символов

Иногда необходимо, чтобы в строках с двойными кавычками символы использовались "как есть", т.е. без учета их специального назначения. Например, может понадобиться, чтобы символ доллара интерпретировался как обычный символ, а не как первый символ имени переменной. Такого эффекта можно добиться с помощью символа обратной косой черты (backslash) (\). Например:

```
$string = 'Имя переменной — $var1';
$string = "Имя переменной — \$var1";
```

При выводе обе строки будут выглядеть одинаково:

```
Имя переменной — is $var1
```

Пусть, например, необходимо присвоить строковой переменной следующее значение:

```
$string = 'Где находится дом О'Хары?';
echo $string;
```

Это выражение будет интерпретировано неправильно, поскольку после символа О следует символ кавычки ('), который будет рассматриваться как символ конца строки. При выводе строки будет получен следующий результат:

```
Где находится дом О
```

Поэтому интерпретатору PHP необходимо указать, что символ кавычки нужно отображать как символ апострофа, а не считать его концом строки. Для этого символ обратной косой черты нужно поместить перед символом кавычки, т.е. для корректного отображения строки нужно внести следующие изменения:

```
$string = 'Где находится дом О\'Хары?';
```

Точно так же, если в строке, заключенной в двойные кавычки, необходимо разместить символ ", то перед ним нужно разместить символ \.

Объединение текстовых строк

Процедура объединения текстовых строк называется *конкатенацией* (concatenation). В языке PHP для выполнения конкатенации используется операция точки (.). Рассмотрим следующий пример:

```
$string1 = "Здравствуй,";
$string2 = "мир!";
$stringall = $string1.$string2;
echo $stringall;
```

При выполнении этого фрагмента будет получен следующий результат:

```
Здравствуй,мир!
```

Обратите внимание, что в результирующей строке не содержится символ пробела (" "), поскольку он отсутствует в обеих исходных строках. Для добавления символа пробела между отдельными словами нужно объединить три строки: две переменные и непосредственно строку с символом пробела. Например:

```
$stringall = $string1." ".$string2;
```

Для добавления символов к существующей строке можно воспользоваться операцией .=. Например, предыдущие примеры можно модифицировать таким образом:

```
$stringall = "Здравствуй,";
$stringall .= " мир!";
echo $stringall;
```

В результате получим следующее:

```
Здравствуй, мир!
```

Манипуляция строками

Для обработки текстовых строк язык PHP предоставляет множество встроенных функций. (Более подробно функции рассматриваются в главе 8.) С помощью стандартных функций в строке можно находить подстроки или отдельные символы, заменять часть строки новыми символами, разбивать строку, находить длину строки и т.д.

Зачастую в начале или в конце строки нужно удалить ненужные пробелы. Это можно осуществить следующим образом:

```
$string = trim($string)  # удаляет пробелы в начале и в конце строки
$string = ltrim($string) # удаляет пробелы только в начале строки
$string = rtrim($string) # удаляет пробелы только в конце строки
```

С использованием функции `str_word_count()` строку можно легко разбить на отдельные слова:

```
str_word_count("строка", формат)
```

Параметр *формат* может иметь два значения: 1 или 2. При использовании значения 1 результатом выполнения функции `str_word_count()` будет простой массив, элементами которого будут отдельные слова. При использовании значения 2 эта функция возвращает ассоциативный массив, ключами которого будут позиции первых символов слов в строке. (Более подробно массивы рассматриваются в главе 6.) Если параметр *формат* отсутствует, возвращаемым значением функции `str_word_count()` будет количество символов в строке. Рассмотрим следующий пример:

```
$string = "Подсчет слов";
$numberOfWords = str_word_count($string);
```

```
$word1 = str_word_count($string, 1);
$word2 = str_word_count($string, 2);
```

При выполнении этого фрагмента будут получены такие результаты:

```
$numberOfWords = 2
$word1[0] = Подсчет
$word1[1] = слов
$word2[0] = Подсчет
$word2[9] = слов
```

Обратите внимание, что первое слово начинается с позиции с 0 (а не с 1, как многие могли бы предположить), а следующее — с позиции 9. Все эти вопросы более подробно рассматриваются в главе 6 при обсуждении массивов.

Некоторые полезные функции, которые можно использовать для обработки текстовых строк, приведены в табл. 5.3. Рассматривая примеры, не забывайте о том, что нумерация символов в строке начинается с нуля.

Форматирование текстовых строк

В языке PHP все значения всегда выводятся в строковом формате. Другими словами, результатом функции `echo` является строка, даже если среди ее параметров содержатся числовые значения. Рассмотрим такой пример:

```
$number = 4;
echo "У Салли $number детей.";
```

В результате выполнения этого фрагмента получим следующую строку:

```
У Салли 4 детей.
```

Несмотря на то что в переменной `$number` содержится число 4, функция `echo` отображает его как часть символьной строки.

Форматирование данных является одним из важных этапов написания сценариев. Однако функция `echo` оказывается недостаточно гибкой. В разделе "Форматирование чисел для вывода" этой главы уже рассматривались возможности функции `number_format()`, связанные с форматированием чисел. Для форматирования строк в языке PHP также имеются дополнительные возможности. Функции `printf()` и `sprintf()` позволяют форматировать строки, числа, а также их произвольную комбинацию.

Функции `printf()` и `sprintf()` имеют следующий общий синтаксис:

```
printf("формат", $имя_переменной1, $имя_переменной2, ...);
$newvar = sprintf("формат", $имя_переменной1, $имя_переменной2, ...);
```

Функция `printf()` предназначена для вывода уже отформатированной строки, а функция `sprintf()` сохраняет ее в переменной. При этом обе функции позволяют обрабатывать комбинацию чисел, строк и значений переменных. Параметр *формат* позволяет задать шаблон, в соответствии с которым будут форматироваться переменные *$имя_переменной*. Например, следующее выражение является абсолютно корректным:

```
$newvar = sprintf("Здравствуй, мир!");
```

В этом выражении не выполняется никакого форматирования, и строка `Здравствуй, мир!` сохраняется в переменной `$newvar`. Однако строковые литералы можно объединять с переменными с текстовыми символами, как показано в следующем примере:

```
$nboys = 3;
$ngirls = 2;
printf("%s парней и %s девушек", $nboys, $ngirls);
```

Таблица 5.3. Функции для обработки текстовых строк

Функция	Описание	Пример	Результат
`str_repeat("строка", n)`	Повторяет *строку n* раз	`$x = str_repeat("x", 5);`	`$x = xxxxx`
`str_replace ("a", "b", "строка")`	В *строке* заменяет все фрагменты *a* фрагментами *b*	`$a = "abc abc"; $s = str_replace("b", "i", $a);`	`$s = aic aic`
`strchr ("строка", "символ")`	Возвращает часть *строки*, начиная с *символа* и до конца строки	`$str = "aBc abc"; $sub = strchr($str, "b");`	`$sub = bc`
`stristr ("строка", "символ")`	Аналогична функции `strchr()`, но не учитывает регистр	`$str = "aBc abc"; $sub = stristr($str, "b");`	`$sub = Bc abc`
`strlen("строка")`	Возвращает длину *строки*	`$n = strlen("привет");`	`$n = 6`
`strpos ("строка", "подстрока")`	Возвращает позицию первого вхождения *подстроки* в *строку*	`$str = "привет"; $n = strpos($str, "и");`	`$n = 2`
`strrchr ("строка", "символ")`	Аналогична функции `strchr()`, но осуществляет поиск последнего фрагмента *символа*	`$str = "abc abc"; $sub = strrchr($str, "b");`	`$sub = bc`
`strrev("строка")`	Возвращает *строку* в обратном порядке	`$n = strrev("abcde");`	`$n = edcba`
`strrpos ("строка", "подстрока")`	Возвращает позицию последнего вхождения *подстроки* в *строку*	`$str = "abc abc"; $n = strrpos($str, "bc");`	`$n = 5`
`strtolower("строка")`	Преобразует все символы *строки* в нижний регистр	`$str = strtolower("ДА");`	`$str = да`
`strtoupper("строка")`	Преобразует все символы *строки* в верхний регистр	`$str = strtoupper("да");`	`$str = ДА`
`strtr("строка", "стр1", "стр2")`	Заменяет все вхождения подстроки *стр1* в *строке* значением *стр2*	`$str = "aa bb cc"; $new = strtr($str, "bb", "xx");`	`$new = aa xx cc`
`substr ("строка", n1, n2)`	Возвращает часть *строки*, начиная с позиции *n1* и заканчивая *n2*	`$sstr = substr("привет", 2, 4);`	`$sstr = иве`
`substr_count ("строка", "sub")`	Возвращает количество вхождений подстроки *sub* в *строку*	`$str = "abc ab abc"; $s = "bc"; $n = substr_count($str, $s);`	`$n = 2`
`substr_replace ("s", "r", n, l)`	Заменяет в строке `s` фрагмент из `l` символов, начиная с позиции `n`, фрагментом `r`	`$s = "abc abc"; $t = substr_replace($s, "x", 2, 3);`	`$t = abxbc`
`ucfirst("строка")`	Преобразует первый символ *строки* в верхний регистр	`$str = "a B c"; $str2 = ucfirst($str);`	`$str2 = A B c`
`ucwords("строка")`	Преобразует первый символ каждого слова *строки* в верхний регистр	`$str = "aa Bb cc"; $str2 = ucwords($str);`	`$str2 = Aa Bb Cc`

Строка %s представляет собой инструкцию форматирования, которая указывает функции printf() на необходимость вставки значения переменной как строки. Таким образом, результатом выполнения приведенного выше примера будет строка 3 парней и 2 девушек. Символ % сообщает функции printf() о том, что дальше следует инструкция форматирования. Инструкции форматирования имеют следующий общий формат:

%заполнитель-длина.точностьтип

В инструкции форматирования используются следующие параметры.

- ✓ Символ процента (%) является первым символом инструкции форматирования.
- ✓ *Заполнитель*. Символ-заполнитель, используемый для дополнения строки до заданной *длины* (этот параметр описывается ниже). Параметр *заполнитель* может содержать пробел (по умолчанию), 0 или любой другой символ, которому предшествует символ одинарной кавычки. Часто в качестве символа-заполнителя используется значения 01 или 0001.
- ✓ Символ выравнивания (-). При его использовании результат будет выравниваться по левой стороне, в противном случае — по правой (режим по умолчанию).
- ✓ *Длина* определяет ширину выводимого значения. Если количество выводимых символов меньше *длины*, то недостающее пространство заполняется *символом-заполнителем*. Например, если *длина* равна 5, *заполнитель* — 0, а форматируемое значение — 1, то в результате будет получена строка 00001.
- ✓ .*точность*. Определяет, сколько десятичных знаков после точки должно выводиться при форматировании действительных чисел.
- ✓ *тип* значения. Обычно используется строковый тип — s (string). Для вывода чисел с плавающей точкой используется тип f (float).

Ниже приведено несколько примеров использования функции sprintf().

```
$money = 30;
$pet = "Kitten";
$new = sprintf("It costs $%03.2f for a %s.\n", $money, $pet);
$new2 = sprintf("%'.-20s%3.2f", $pet, $money);
echo $new;
echo $new2;
```

В результате выполнения этого фрагмента получим следующий результат.

```
It costs $030.00 for a Kitten.
Kitten.............. 30.00
```

Обратите внимание на то, что для форматирования значения переменной $money используется шаблон 3.2f (три цифры до десятичной точки и две после нее). Этот шаблон используется при формировании обеих строковых переменных, $new и $new2. В то же время в переменной $new значение переменной $money дополняется символом 0, а в $new2 — пробелом.

При добавлении переменной $pet в переменную $new2 использовался шаблон '.-20. Значение 20 определяет количество символов, которые будут использоваться для переменной $pet. Поскольку значение этой переменной Kitten занимает шесть символов, оставшееся пространство дополняется символом-заполнителем ('.), т.е. дополнительно будет выведено 14 точек (.). Инструкция выравнивания (-) определяет, что строка Kitten будет выравниваться по левому краю. В противном случае использовалось бы выравнивание по правому краю.

Зачастую необходимо вывести столбцы чисел. Пусть, например, имеется три числа: 12.3, 1 и 234.55. При использовании функции echo будет получен следующий результат:

```
12.3
1
234.55
```

Даже при использовании функции `number_format()` для задания двух десятичных знаков после точки нужный результат не будет достигнут:

```
12.30
1.00
234.55
```

Для вывода чисел в упорядоченном столбце лучше всего воспользоваться функцией `printf()`:

```
printf("%5.2f\n", $number1);
printf("%5.2f\n", $number2);
printf("%5.2f\n", $number3);
```

Только теперь получен необходимый результат:

```
 12.30
  1.00
234.55
```

Для вывода числовых значений в данном случае использовался шаблон форматирования `%5.2f\n`. Рассмотрим его составные части более подробно.

- ✓ `%`. Первый символ инструкции форматирования.
- ✓ `5`. Определяет ширину, т.е. количество знаков, которое будет использоваться для вывода чисел. Если количество цифр в числе меньше 5, то спереди добавляются символы-заполнители (в данном случае пробелы). Поскольку выравнивание справа используется по умолчанию, то символ выравнивания не используется.
- ✓ `2`. Определяет количество знаков после десятичной точки.
- ✓ Тип `f` определяет, что числа будут выводиться как числа с плавающей точкой.
- ✓ `\n`. Символ перевода строки.

Для того чтобы вывести число в денежном формате (со знаком $), можно воспользоваться функцией `sprintf()`. Например:

```
$newvariablename = sprintf("$%.2f", $oldvariablename);
```

В этом выражении форматируется значение переменной `$oldvariablename`, а результат сохраняется в переменной `$newvariablename`. В следующем примере денежное значение выводится в правильном формате:

```
$price = 25;
printf("$%.2f", $price);
```

В результате получим следующее:

```
$25.00
```

Использование даты и времени

Дата и время являются важными элементами сценариев. Поэтому язык PHP предоставляет широкие возможности для их обработки. Компьютер хранит значения даты и времени в формате *timestamp* в секундах. Поскольку этот формат использовать неудобно, в языке PHP имеются различные встроенные функции для его преобразования.

Формат *timestamp* используется в операционной системе Unix и отсчитывает количество секунд, прошедших с 1 января 1970 года, `00:00:00` GMT. Этот формат чрезвычайно удобен при вычислении временной разницы между двумя событиями. Для этого достаточно просто вычесть одно значение из другого.

Форматирование даты

Для форматирования даты чаще всего используется функция `date()`. Она возвращает строковое значение даты и времени в специальном формате. Функция `date()` имеет следующий общий синтаксис:

```
$mydate = date("формат", $timestamp);
```

Параметр *$timestamp* содержит количество секунд. Как описывается в следующем разделе, это значение можно получить с помощью функции `time()` или `mktime()`. Если параметр *$timestamp* не указан, то используется текущая дата и время. Например:

```
$today = date("Y/m/d");
```

Если бы сегодня было 10 марта 2004 года, то в результате было бы выведено следующее значение:

```
2004/03/10
```

Строковый параметр *формат* определяет формат отображения даты и времени. Например, при использовании формата `y-m-d` функция `date()` вернула бы значение `04-3-10`, а при использовании `M.d.Y` — `Mar.10.2004`. В табл. 5.4 приведены различные символы, которые можно использовать в строковом параметре *формат*. (Их полный перечень можно найти на Web-узле `www.php.net`.) Различные элементы шаблона даты могут быть отделены друг от друга дефисом (`-`), точкой (`.`), косой чертой (`/`) или пробелом.

Таблица 5.4. Символы форматирования даты

Символ	Значение	Пример
`M`	Трехбуквенное английское сокращение месяца	`Jan`
`F`	Английское название месяца	`January`
`m`	Месяц (две цифры с нулями)	`02` или `12`
`n`	Месяц (одна-две цифры без нулей)	`1` или `12`
`d`	День месяца (две цифры с нулями)	`01` или `14`
`j`	День месяца (две цифры без нулей)	`3` или `30`
`l`	Английское название дня недели	`Friday`
`D`	Трехбуквенное английское сокращение дня недели	`Fri`
`w`	Порядковое число дня недели, от `0` (воскресенье) до `6` (суббота)	`5`
`Y`	Год (четыре цифры)	`2004`
`y`	Год (две цифры)	`04`
`g`	Часы (12-часовой формат без нулей, от `1` до `12`)	`2` или `10`
`G`	Часы (24-часовой формат без нулей, от `0` до `23`)	`2` или `15`
`h`	Часы (12-часовой формат с нулями, от `01` до `12`)	`01` или `10`
`H`	Часы (24-часовой формат с нулями, от `00` до `23`)	`00` или `23`
`i`	Минуты	`00` или `59`
`s`	Секунды	`00` или `59`

Окончание табл. 5.4

Символ	Значение	Пример
a	до или после полудня: am или pm (в нижнем регистре)	am
A	ДО или ПОСЛЕ полудня: AM или PM (в верхнем регистре)	AM
U	Целое число секунд, прошедших с 1 января 1970 года, 00:00:00 GMT	1056244941

Хранение значений в формате *timestamp*

Для присваивания текущего значения даты и времени в формате *timestamp* предназначена функция `time()`:

```
$today = time();
```

То же самое можно осуществить с помощью выражения

```
$today = strtotime("today");
```

Для получения любого другого значения даты и времени в формате *timestamp* предназначена функция `mktime()`. Она имеет следующий синтаксис:

```
$importantDate = mktime(h, m, s, mo, d, y);
```

где аргумент `h` соответствует часам; `m` — минутам; `s` — секундам; `mo` — месяцу; `d` — дню; `y` — году. Например, для вычисления секунд, прошедших к 15 января 2004 года, необходимо использовать следующее выражение:

```
$importantDate = mktime(0, 0, 0, 1, 15, 2003);
```

Для получения значений в формате *timestamp* можно использовать различные английские ключевые слова или сокращения. Например, приведенное выше выражение можно переписать следующим образом:

```
$importantDate = strtotime("January 15 2003");
```

Функция `strtotime()` может распознавать следующие слова и сокращения.

- ✓ **Название месяцев**: 12 месяцев и соответствующие сокращения.
- ✓ **Название дней недели**: 7 дней и соответствующие сокращения.
- ✓ **Название единиц времени**: `year` (год), `month` (месяц), `fortnight` (две недели), `week` (неделя), `day` (день), `hour` (час), `minute` (минута), `second` (секунда), am (до полудня), pm (после полудня).
- ✓ **Некоторые английские слова**: `ago` (тому назад), `now` (сейчас), `last` (длиться), `next` (следующий); `this` (этот), `tomorrow` (завтра), `yesterday` (вчера).
- ✓ **Знаки "плюс" и "минус"**.
- ✓ **Все числа**.
- ✓ **Временные зоны**: например, gmt (Greenwich Mean Time — среднее время по Гринвичу) или pdt (Pacific Daylight Time — дневное тихоокеанское время).

Слова и сокращения можно по-разному комбинировать. Например:

```
$importantDate = strtotime("tomorrow");          # 24 часа от сегодня
$importantDate = strtotime("now + 24 hours");
$importantDate = strtotime("last Saturday");
$importantDate = strtotime("8pm + 3 days");
$importantDate = strtotime("2 weeks ago");       # две недели назад
$importantDate = strtotime("next year gmt");     # на один год вперед
$importantDate = strtotime("tomorrow 4am");
```

Для нахождения временной разницы между событиями достаточно просто вычесть одно значение из другого. Например, если переменная `$importantDate` описывает прошедшее событие, то для определения его "давности" соответствующее значение нужно отнять от значения переменной `$today` (которая должна быть определена ранее). Например:

```
$timeSpan = $today-$importantDate;
```

Полученное значение и будет временной разницей в секундах между двумя событиями. Перевести это значение в часы можно следующим образом:

```
$timeSpan = (($today-$importantDate)/60)/60;
```

Глава 6

Объединение данных с помощью массивов

В этой главе...

- Создание массивов
- Присваивание значений элементам массивов
- Сортировка массивов
- Обработка значений в массивах
- Создание многомерных массивов

Массивы представляют собой составные переменные, позволяющие объединять данные под одним именем. Например, в одной переменной `$FordInfo` можно хранить такую информацию о машине, как название модели, цвет или стоимость. При этом массивы можно легко обрабатывать, получать к ним доступ или изменять информацию. Например, в PHP предусмотрено несколько методов сортировки данных в массивах.

В этой главе предлагается описание методов создания, изменения, копирования и использования массивов.

Создание массивов и работа с ними

Массивы являются важным элементом программирования на языке PHP. В этом разделе описывается, как создавать, изменять и удалять массивы.

Создание массивов

Для создания переменной в PHP ей можно присвоить значение. Аналогично, чтобы создать массив, необходимо присвоить ему соответствующие значения. Например, пусть `$customers` — переменная, которая ранее не использовалась в сценарии. Тогда для создания массива с именем `$customers` воспользуемся выражением

```
$customers[1] = "Сэм Смит";
```

Теперь `$customers` — имя массива с единственным значением `"Сэм Смит"`. Рассмотрим следующий фрагмент кода:

```
$customers[2] = "Сью Джонс";
$customers[3] = "Мэри Хуанг";
```

Теперь массив `$customers` содержит три значения: `"Сэм Смит"`, `"Сью Джонс"` и `"Мэри Хуанг"`.

Массив можно рассматривать как набор пар *ключ–значение* (key/value), т.е.

```
$arrayname['ключ1'] = значение1;
$arrayname['ключ2'] = значение2;
$arrayname['ключ3'] = значение3;
```

и т.д. по всем элементам массива.

Ключ (key) также называют *индексом* (index).

В качестве индекса в массивах могут использоваться числа или строки. В массиве `$customers` индексами являются 1, 2 и 3. Однако можно использовать и текстовые строки. Рассмотрим пример создания массива, в котором индексами являются прописные буквы некоторых штатов:

```
$capitals['CA'] = "Сакраменто";
$capitals['TX'] = "Остин";
$capitals['OR'] = "Салем";
```

Для создания массива можно также использовать сокращенную запись, а не писать отдельное выражение для каждого значения индекса. Одним из возможных вариантов является

```
$streets[] = "ул. Вязов";
$streets[] = "ул. Дубов";
$streets[] = "7-ая авеню";
```

В таком случае создается массив с автоматической нумерацией индексов — числами, начиная с 0. Например, для отображения первого элемента массива можно воспользоваться следующим выражением:

```
echo "$streets[0]";
```

В результате получим:

```
ул. Вязов
```

Отсчет значений индекса массивов начинается с 0, если только специально не присвоить другое значение. Одной из часто совершаемых ошибок является использование в качестве первого значения индекса массива 1, а не 0.

Еще более сокращенной записью создания массива является

```
$streets = array("ул. Вязов", "ул. Дубов", "7-ая авеню");
```

В результате выполнения этого оператора создается массив, аналогичный созданному в предыдущем примере. Если, например, необходимо, чтобы отсчет в массиве начинался с 12, а не с 0, можно воспользоваться выражением

```
$streets = array(12 => "ул. Вязов", "ул. Дубов", "7-ая авеню");
```

В результате будет создан следующий массив:

```
$streets[12] = "ул. Вязов";
$streets[13] = "ул. Дубов";
$streets[14] = "7-ая авеню";
```

Аналогичным образом можно создавать массивы с текстовыми индексами. Например, с помощью следующего выражения можно создать массив сокращенных названий штатов:

```
$capitals = array("CA" => "Сакраменто",
                  "TX" => "Остин",
                  "OR" => "Салем"));
```

Обратите внимание на структуру этого выражения. PHP не учитывает пробелы и перевод строки. Все выражение можно было бы написать в одну длинную строку. Однако приведенная выше структура позволяет программистам облегчить анализ кода сценария, который стоит создавать настолько прозрачным и четким, насколько это возможно. При тестировании кода сценария вы с удовольствием сможете оценить затраченные усилия.

PHP также позволяет создавать массивы, значения которых будут находиться в некотором промежутке. Например:

```
$years = range(2001, 2010);
```

В результате будет создан следующий массив:

```
$years[0] = 2001
$years[1] = 2002
...
$years[8] = 2009
$years[9] = 2010
```

Можно также воспользоваться выражением

```
$reverse_letters = range("z", "a");
```

и в результате будет создан массив из 26 элементов:

```
$reverse_letters[0] = z
$reverse_letters[1] = y
...
$reverse_letters[24] = b
$reverse_letters[25] = a
```

Вывод элементов массивов

Для просмотра структуры и значений элементов массива можно воспользоваться одной из двух функций: `var_dump()` или `print_r()`. Однако функция `print_r()` не так полезна. Рассмотрим следующее выражение для вывода массива `$customers`:

```
print_r($customers);
```

В результате получим:

```
Array
(
      [1] => Сэм Смит
      [2] => Сью Джонс
      [3] => Мэри Хуанг
)
```

Таким образом, выводятся только индекс и значение каждого элемента массива. Для получения более подробной информации о массиве следует использовать функцию `var_dump()`:

```
var_dump($customers);
```

В результате получим следующее:

```
array(3){
      [1]=>
      string(9) "Сэм Смит"
      [2]=>
      string(9) "Сью Джонс"
      [3]=>
      string(10) "Мэри Хуанг"
}
```

В этом случае вместе с индексом и значением будет также выводиться тип данных каждого элемента массива (например, текстовая строка из девяти символов). Так, массив, содержащий информацию об имени покупателя и его возрасте, отобразится с помощью функции `var_dump()` следующим образом:

```
array(2){
      ["name"]=>
      string(9)"Сэм Смит"
      ["age"]=>
      int(12)
}
```

Целочисленный тип элемента массива определяется ключевым слова `int`. В данном примере элемент с индексом `age` имеет значение `12`.

Необходимо помнить, что в таком виде информация о массиве возвращается интерпретатором PHP. Однако при использовании PHP для Web результат выполнения функций `print_r()` и `var_dump()` будет отображаться с помощью HTML, т.е. будет выведен в одну длинную строку. Для форматирования выводимой в браузер информации о массивах следует воспользоваться соответствующими HTML-дескрипторами, например:

```
echo "<pre>";
var_dump($customers);
echo "</pre>";
```

Изменение массивов

Массивы можно изменять таким же образом, как и переменные. PHP позволяет изменять значения, добавлять или удалять элементы и производить сортировку. Например, если имеется массив `$capitals`, значение его элемента можно легко изменить следующим образом:

```
$capitals['TX'] = "Биг Спрингс";
```

Это выражение присваивает элементу с индексом `TX` новое значение `"Биг Спрингс"`, хотя жители Остина (Austin) могут возразить (Остин — столица штата Техас). Для создания нового элемента массива `$capitals` можно воспользоваться следующим кодом:

```
$capitals['RI'] = "Провиденс";
```

Это выражение добавляет новый элемент, не изменяя существующие.

Предположим, имеется созданный ранее массив, в котором индексами являются целые числа:

```
$customers[1] = Сэм Смит
$customers[2] = Сью Джонс
$customers[3] = Мэри Хуанг
```

При использовании выражения

```
$customers[] = "Хуан Лопес";
```

массив `$customers` будет уже состоять из четырех элементов:

```
$customers[1] = Сэм Смит
$customers[2] = Сью Джонс
$customers[3] = Мэри Хуанг
$customers[4] = Хуан Лопес
```

PHP позволяет легко копировать существующий массив в новый, например:

```
$customerCopy = $customers;
```

Удаление значений из массива

Иногда необходимо полностью удалить значение из массива. Например, предположим, что имеется массив

```
$colors = array("красный", "зеленый", "синий", "розовый", "желтый"));
```

который содержит пять значений. А теперь предположим, что вам больше не нравится розовый цвет и вы хотите от него избавиться. Этого можно добиться таким образом:

```
$colors[3] = "";
```

Однако это выражение присваивает элементу `$colors[3]` пустое значение, но не удаляет его. Массив `$colors` все еще содержит пять элементов, значение одного из которых — пустая строка. Для того чтобы полностью удалить его из массива, необходимо воспользоваться следующим выражением:

```
unset($colors[3]);
```

Теперь массив $colors содержит четыре значения:

```
$colors[0] = красный
$colors[1] = зеленый
$colors[2] = синий
$colors[4] = желтый
```

Следует заметить, что при удалении элемента с индексом 3 все остальные остались неизменными.

После того как массив создан, он не перестает существовать до тех пор, пока его специально не удалить. При этом удаление его элементов по отдельности не влечет за собой удаление массива как такового, точно так же, как извлечение всей одежды из шкафа не говорит о ее исчезновении. Для того чтобы удалить массив, необходимо использовать выражение

```
unset($colors);
```

Сортировка массивов

Одной из самых замечательных возможностей, предоставляемых языком PHP при работе с массивами, является их сортировка. Исходно интерпретатор PHP размещает значения элементов массива в указанном вами порядке и позволяет соответствующим образом выводить их. Очень часто необходимо изменить порядок следования элементов. Например, нужно отсортировать элементы массива в алфавитном порядке по индексу или по значению.

Язык PHP предоставляет широкие возможности при сортировке массивов. Например, чтобы изменить порядок следования элементов массива с числами в качестве индексов, следует воспользоваться функцией sort():

```
sort($имя_массива);
```

Это выражение сортирует элементы массива по значению и присваивает им новые индексы в порядке следования. При этом сначала идут значения, начинающиеся с чисел, потом — с прописных и строчных букв. Например, рассмотрим массив $streets:

```
$streets[0] = "ул. Вязов";
$streets[1] = "ул. Дубов";
$streets[2] = "7-ая авеню";
```

Использование выражения

```
sort($streets);
```

приведет к следующему:

```
$streets[0] = "7-ая авеню";
$streets[1] = "ул. Вязов";
$streets[2] = "ул. Дубов";
```

Если же использовать функцию sort() для сортировки массива, индексами которого являются текстовые строки, PHP преобразует их в числа.

Поэтому для таких целей необходимо воспользоваться функцией asort():

```
asort($capitals);
```

Это выражение сортирует значения столиц по значению, при этом не изменяя индексы массива. Рассмотрим массив $capitals, созданный в предыдущем разделе:

```
$capitals['CA'] = "Сакраменто";
$capitals['TX'] = "Остин";
$capitals['OR'] = "Салем";
```

При использовании выражения

```
asort($capitals);
```

получим следующее:

```
$capitals['TX'] = "Остин"
$capitals['CA'] = "Сакраменто"
$capitals['OR'] = "Салем"
```

Заметим, что порядок следования индексов массива изменяется в соответствии со значениями. Теперь столицы расположены в алфавитном порядке. Если бы в качестве индексов массива использовались числа, то они бы тоже поменяли свой порядок. Например, пусть имеется массив

```
$capitals[1] = "Сакраменто";
$capitals[2] = "Остин";
$capitals[3] = "Салем";
```

Результатом выполнения функции `asort()` будет

```
$capitals[2] = "Остин"
$capitals[1] = "Сакраменто"
$capitals[3] = "Салем"
```

Вряд ли эта функция будет полезна для массивов с числовыми индексами.

Язык PHP предоставляет также и другие возможности для сортировки массивов различными способами. В табл. 6.1 представлены необходимые для этого функции.

Таблица 6.1. Сортировка массивов

Сортировка массивов	Описание
`sort(`*$имя_массива*`)`	Сортирует значения элементов массива по возрастанию, присваивая им при этом новые числовые индексы
`asort(`*$имя_массива*`)`	Сортирует значения элементов массива по возрастанию, не изменяя при этом индексы
`rsort(`*$имя_массива*`)`	Сортирует значения элементов массива по убыванию, присваивая при этом новые числовые индексы
`arsort(`*$имя_массива*`)`	Сортирует значения элементов массива по убыванию, не изменяя при этом индексы
`ksort(`*$имя_массива*`)`	Сортирует индексы элементов массива по возрастанию
`krsort(`*$имя_массива*`)`	Сортирует индексы элементов массива по убыванию
`usort(`*$имя_массива, имя_функции*`)`	Пользовательская сортировка, определяемая функцией *имя_функции* (функции рассматриваются в главе 8)
`natsort(`*$имя_массива*`)`	Сортирует смешанные значения (текстовые и числовые) элементов массива и устанавливает "естественный" порядок. Например, массив со значениями `day1, day5, day11, day2` будет отсортирован следующим образом: `day1, day2, day5, day11`. В то время как результатом выполнения функции `sort()` будет: `day1, day11, day2, day5`

Использование массивов в выражениях

Массивы можно использовать в выражениях точно так же, как и переменные. Этот раздел как раз и посвящен этому вопросу.

Значения отдельных элементов массива можно легко получать путем прямого доступа к ним, как, например, в следующем выражении:

```
$CAcapital = $capitals['CA'];
echo $CAcapital;
```

Результатом выполнения будет

```
Сакраменто
```

Если попытаться получить доступ к несуществующему элементу массива, PHP выдаст сообщение об ошибке:

```
$CAcapital = $capitals['CAx'];
```

Поскольку в массиве `$capitals` нет элемента с индексом `CAx`, будет выведено следующее:

```
Notice: Undefined index: CAx in d:\testarray.php on line 9
(Предупреждение: Неизвестный ключ: CAx в файле d:\testarray.php в строке 9)
```

При этом сообщение об ошибке не прекращает выполнения последующих строк кода сценария. Но поскольку переменной `$CAcapital` не было присвоено никакого значения, при использовании функции `echo` будет выводиться пустая строка. Для предотвращения сообщения об ошибке необходимо использовать символ `@`:

```
@$CAcapital = $capitals['CAx'];
```

Использование массивов в операторе `echo`

Для вывода элементов массива с помощью функции `echo` следует воспользоваться выражением

```
echo $capitals['TX'];
```

В результате получим:

```
Остин
```

Если необходимо поместить массив в длинную строку в двойных кавычках, его название нужно заключить в фигурные скобки:

```
echo "Столица штата Техас – {$capitals['TX']}";
```

Результатом выполнения этого выражения будет следующее:

```
Столица штата Техас – Остин
```

Использование массивов в функции `list`

PHP позволяет получать значения и индексы нескольких элементов массива одновременно. С помощью функции `list()` можно копировать значения массивов в переменные. Пусть, например, имеется массив

```
$shoeInfo = array("мокасины", "черные", 22.00);
```

Информацию о нем можно получить, используя функцию `print_r()`:

```
print_r($shoeInfo);
```

Результатом выполнения будет следующее:

```
Array
(
    [0] => мокасины
    [1] => черные
    [2] => 22
)
```

Следующий пример иллюстрирует использование функции `list()`:

```
list($first, $second) = $shoeInfo;
echo $second, " ",$first;
```

В результате PHP создаст две переменные, `$first` и `$second`, которым будут присвоены значения первых двух элементов массива `$shoeInfo`, что эквивалентно выполнению операторов:

```
$first = $shoeInfo[0];
$second = $shoeInfo[1];
```

Значение третьего элемента массива, `$shoeInfo`, не будет скопировано, поскольку функция `list()` содержит только два аргумента. В результате функция `echo` выведет

```
черные мокасины
```

В некоторых случаях необходимо получить индекс элемента массива, а не его значение. Для этих целей предназначена функция `key()`. Предположим, что первым элементом массива является

```
$shoeInfo['style'] = "мокасины";
```

В следующем выражении с помощью `echo` выводятся индекс и значение первого элемента массива `$shoeInfo`:

```
$value = $shoeInfo['style'];
$key = key($shoeInfo);
echo "$key: $value";
```

Результатом выполнения будет

```
стиль: мокасины
```

В первой строке кода переменной `$value` присваивается значение `"мокасины"`, во второй с помощью функции `key()` переменной `$key` присваивается индекс элемента. В данном случае мы получаем информацию о первом элементе, поскольку указатель массива, который отвечает за перемещение по массиву, установлен в текущей позиции. В следующем разделе более подробно рассматриваются вопросы, связанные с указателями на массивы.

Перемещение по массивам

Часто необходимо производить различные операции над каждым элементом массива. Например, нужно вывести все значения, сохранить их в базе данных или прибавить к каждому элементу число 6. Формально выражаясь, процесс перебора всех элементов массива является *итеративным*, т.е. состоящим из итераций, или *проходом* (traversing). В данном разделе описаны два таких способа.

- ✓ **Перебор вручную.** Использование указателя для перехода от одного элемента массива к другому;
- ✓ **С помощью оператора `foreach`.** Поэлементный автоматический проход по массиву от его начала до конца.

Перебор элементов массива вручную

Для перебора элементов массива вручную можно воспользоваться указателем. При этом массив представляется как список, содержащий указатель на его элементы. Указатель не изменяет свое положение до тех пор, пока его не переместить. Для этого в языке PHP предназначены следующие функции.

- ✓ `current(`*`$имя_массива`*`)`. Возвращает текущее значение элемента массива, не перемещая указатель.
- ✓ `next(`*`$имя_массива`*`)`. Перемещает указатель вперед на один элемент.
- ✓ `previous(`*`$имя_массива`*`)`. Перемещает указатель назад на один элемент.
- ✓ `end(`*`$имя_массива`*`)`. Перемещает указатель в конец массива.
- ✓ `reset(`*`$имя_массива`*`)`. Перемещает указатель в начало массива.

С помощью следующего выражения осуществляется проход по всем элементам массива, содержащего названия столиц штатов:

```
$value = current($capitals);
echo "$value<br>";
$value = next($capitals);
echo "$value<br>";
$value = next($capitals);
echo "$value<br>";
```

Поскольку указатель ранее не перемещался, по умолчанию он установлен на первый элемент массива. Если же положение указателя изменялось в предыдущих строках кода или нет уверенности, что отсчет начнется с первого элемента, то следует предварительно воспользоваться функцией `reset()`:

```
reset($capitals);
```

Использование этого метода для прохода по массиву предполагает выполнение оператора присваивания и функции `echo` для каждого элемента (в общем случае для каждого из 50 штатов). В результате будет выведен список названий штатов.

Этот метод достаточно гибок. Можно просто перемещаться по массиву в произвольном порядке и обрабатывать его элементы. Можно перемещаться назад, непосредственно перейти в конец, пропускать элементы, используя подряд два выражения `next`, или применять любой другой полезный прием. Однако, если необходимо пройти поэлементно массив от начала до конца, язык PHP позволяет использовать более простой метод, а именно оператор `foreach`, который выполняет те же действия намного эффективнее. Этот оператор рассматривается в следующем разделе.

Использование оператора `foreach` для прохода по массиву

Оператор `foreach` осуществляет поэлементный проход по массиву и позволяет выполнить набор инструкций со значениями и индексами. Общий синтаксис имеет вид:

```
foreach($имя_массива as $ключ => $значение)
{
    набор операторов;
}
```

Параметрами выражения `foreach` являются следующие.

- ✓ *$имя_массива*. Имя массива.
- ✓ *$ключ*. Имя переменной, которой будут присваиваться значения индексов элементов массива. Этот параметр является необязательным, т.е. без его использования сохранится только значение элемента массива в переменной *$значение*.
- ✓ *$значение*. Имя переменной, которой будут присваиваться значения элементов массива.

Рассмотрим следующий пример, в котором оператор `foreach` используется для вывода численности населения по штатам:

```
$state_population = array ("CA" => 34501130,
                           "WY" => 494423,
                           "OR" => 3472867);
ksort($state_population);
foreach($state_population as $state => $population)
{
    $population = number_format($population);
    echo "$state: $population.<br>";
}
```

В результате на Web-странице отобразится следующее:

```
CA: 34,501,130
OR: 3,472,867
WY: 494,423
```

Можно воспользоваться и следующей строкой кода:

```
foreach($state_population as $population)
```

Однако в этом случае мы будем иметь доступ только к значениям (численности населения) массива, но не к индексам.

При использовании оператора `foreach` указатель автоматически перемещается в начало, т.е. нет необходимости использовать функцию `reset()`.

Определение размера массива

Для получения информации о структуре и значениях массива можно использовать функцию `var_dump()` или `print_r()` (см. выше раздел "Вывод элементов массивов"). Но иногда необходимо просто знать размер массива. Для этого в языке PHP предусмотрены две функции: `count()` и `sizeof()`. Их общий синтаксис имеет следующий вид:

```
$n = count($имя_массива);
```

```
$n = sizeof($имя_массива);
```

В результате выполнения переменной `$n` будет присвоено значение количества элементов в массиве.

Преобразование массивов в текстовые строки (и наоборот)

Иногда необходимо выполнять операции над данными в специальном формате. Например, надо вывести слова предложения в отдельных строках. Одним из вариантов решения данной задачи является добавление перед выводом после каждого слова символа `'\n'`. Это легко можно осуществить с помощью оператора `foreach`, если предложение представляет собой массив, а не текстовую строку. PHP как раз позволяет создавать массив, элементами которого являются слова некоторого предложения.

Для того чтобы переписать содержимое текстовой строки в массив, можно воспользоваться функцией

```
$arrayname = explode("символ", строка);
```

Первым аргументом данной функции является *символ*, который используется для разбиения *строки* и представления ее в виде массива. Рассмотрим следующий пример:

```
$string1 = "Это: новый: дом";
$testarray = explode(":", $string1);
print_r($testarray);
```

Функция `explode()` используется для разделения строки `$string1` символом двоеточия (`:`) и создания массива `$testarray`, содержащего подстроки. Результатом ее выполнения будет

```
Array
(
      [0] => Это
      [1] => новый
      [2] => дом
)
```

При этом переменная `$string1` не изменяет свое значение.

И наоборот, аналогичным образом массивы можно преобразовывать в строки. Для выполнения этой операции предназначена функция

```
$resString = implode("символ", $массива);
```

В результате будет создана строка $resString, состоящая из элементов массива *$массива*, разделенных *символом*. Рассмотрим следующий пример:

```
$arrayIn = array("красный", "синий", "зеленый");
$stringOut = implode(";", $arrayIn);
echo $stringOut;
```

Результат выполнения функции implode() будет присвоен переменной $stringOut. Так же как и в случае с функцией explode(), implode() не изменяет значения входных аргументов, а только считывает их. Если какая-то функция будет преобразовывать входные параметры, об этом будет сказано отдельно.

В результате выполнения последнего выражения получим:

```
красный;синий;зеленый
```

Между вставленными в строку элементами массива нет пробелов, поскольку они не были указаны в параметрах функции implode(). Следующее выражение позволяет выполнить эту операцию:

```
$stringOut = implode("; ", $arrayIn);
```

Таким образом, в результате выполнения этой функции получим:

```
красный; синий; зеленый
```

Преобразование переменных в массивы (и наоборот)

Иногда данные элементов массива необходимо сохранить в переменных, которые впоследствии можно использовать в выражениях PHP. Либо может возникнуть потребность в обратной операции — преобразовать переменные в элементы массива. Например, нужно выполнить одну и ту же операцию (прибавление 1) над несколькими переменными. Если преобразовать переменные в массив, с помощью одного оператора foreach можно получить доступ ко всем значениям и выполнить требуемые действия. В противном случае пришлось бы отдельно для каждой переменной писать свою строку кода.

Для решения этих задач в языке PHP предусмотрены функции extract() и compact(). Функция extract() позволяет получать значения элементов массива и присваивать их переменным, именами которых являются соответствующие индексы. Другими словами, значение каждого элемента копируется в переменную с именем индекса. Рассмотрим следующий пример, в котором с помощью функции extract() выводятся все данные массива:

```
$testarray = array("pink" => "гвоздика", "red" => "роза");
extract($testarray);
echo "Моим любимым красным цветком является $red.\n";
echo "Моим любимым розовым цветком является $pink.";
```

В результате будет выведено:

```
Моим любимым красным цветком является роза.
Моим любимым розовым цветком является гвоздика.
```

Функция compact() предназначена для выполнения обратного действия — преобразования набора переменных в массив. При этом значения переменных копируются в массив. Рассмотрим следующий пример:

```
$color1 = "красный";
$color2 = "синий";
$a = "фиолетовый";
$b = "оранжевый";
```

```
$arrayIn = array("a", "b");
$arrayOut = compact("color1", "color2", $arrayIn);
```

В результате получим следующий массив:

```
$arrayOut[color1] = красный
$arrayOut[color2] = синий
$arrayOut[a] = фиолетовый
$arrayOut[b] = оранжевый
```

Видно, что имена переменных использовались в качестве индексов.

Следует заметить, что в функции `compact()` использовались два метода для создания массива.

- ✓ **Первый** состоит в использовании имен переменных напрямую для индексов. Так происходит с переменными `$color1` и `$color2`.
- ✓ **Второй** состоит в использовании массива, содержащего имена переменных. В предыдущем примере таковым является `$arrayIn` с переменными `$a` и `$b`. Функция `compact()` добавляет эти имена в результирующий массив `$arrayOut`.

Язык PHP позволяет использовать оба метода. Если имеется несколько переменных, которые необходимо сгруппировать в массив, первый метод является более предпочтительным. В противном случае лучше применять второй метод, объединив сначала имена переменных в некоторый массив, а затем использовать его в качестве аргумента функции `compact()`.

Разбиение и слияние массивов

Часто необходимо объединять или разделять массивы. Предположим, например, что имеются два класса студентов и два соответствующих массива, содержащих их имена. Если нужно было бы объединить оба класса, это же действие пришлось бы выполнять и для массивов.

PHP позволяет разбить массив и создать новый, содержащий подмножество элементов исходного. Общий синтаксис функции, выполняющей это действие, имеет следующий вид:

```
$subArray = array_slice($массив, n1, n2);
```

Параметр *n1* указывает позицию элемента, с которого начнется разбиение исходного *$массива* (0 — с первого элемента, 1 — со второго), а *n2* — количество элементов нового массива. Например:

```
$testarray = array("красный", "зеленый", "синий", "розовый");
$subArray = array_slice($testarray, 1, 2);
```

В результате массив `$subArray` будет иметь вид

```
[0] => зеленый
[1] => синий
```

Разбиение массива `$testarray` начинается с позиции номер 1 и содержит два элемента.

По умолчанию отсчет начинается с 0, а не с 1, поэтому первый элемент результирующего массива имеет значение `"зеленый"`, а не `"красный"`.

Аналогичным образом можно объединить различные массивы, используя следующее выражение:

```
$bigArray = array_merge($массив1, $массив2, ...);
```

Рассмотрим пример объединения двух массивов:

```
$array1 = array("красный", "синий");
$array2 = array("зеленый", "желтый");
$bigArray = array_merge($array1, $array2);
```

В результате будет создан массив `$bigArray`:

```
$bigArray[0] = красный
$bigArray[1] = синий
$bigArray[2] = зеленый
$bigArray[3] = желтый
```

Таким же образом можно объединять массивы не только с числовыми индексами, но и с текстовыми. Однако, если индексы для некоторых элементов совпадают, последний из них заменит первоначальный. Рассмотрим следующий пример:

```
$array1 = array("color1" => "красный", "color2" => "синий");
$array2 = array("color1" => "зеленый", "color3" => "желтый");
$bigArray = array_merge($array1, $array2);
```

В результате получим:

```
$bigArray[color1] = зеленый
$bigArray[color2] = синий
$bigArray[color3] = желтый
```

Если необходимо объединить массивы с одинаковыми индексами, следует воспользоваться функцией `array_merge_recursive()`, а не `array_merge()`. В этом случае будет создан многомерный массив, а не заменены исходные значения. Многомерные массивы более подробно рассматриваются в разделе "Многомерные массивы" далее в этой главе.

Сравнение массивов

Очень часто необходимо знать, совпадают ли массивы. Для этого надо найти одинаковые и различные элементы. Эта операция может быть выполнена с помощью функции

```
$diffArray = array_diff($массив1, $массив2, ...);
```

В результате выполнения функции `array_diff()` массив `$diffArray` будет содержать элементы из `$array1`, отсутствующие в `$array2`. При этом в результирующем массиве ключи сохранят свои значения. Рассмотрим следующий пример:

```
$array1 = array("a" => "яблоко", "b" => "апельсин", "c" => "банан");
$array2 = array("слива", "апельсин", "банан");
$diffArray = array_diff($array1, $array2);
```

В результате выполнения этого фрагмента кода получим:

```
$diffArray[a] = яблоко;
```

Поскольку элемент со значением `"яблоко"` находится в массиве `$array1`, но не в `$array2`, он записывается в результирующий `$diffArray` с индексом `"a"`.

Порядок следования аргументов в функции `array_diff()` является очень важным. Так, если последнюю строку кода в предыдущем примере переписать как

```
$diffArray = array_diff($array2, $array1);
```

то результирующий массив `$diffArray` будет иметь вид

```
$diffArray[0] = слива;
```

Единственным элементом, который находится в массиве `$array2`, но не в `$array1`, будет элемент со значением `"слива"`. Именно поэтому он и записывается в `$diffArray`.

Если необходимо найти элементы, отличающиеся либо своим значением, либо ключом, следует воспользоваться следующей функцией:

```
$diffArray = array_diff_assoc($array1, $array2);
```

Подставляя в качестве аргументов функции `array_diff_assoc()` массивы, которые использовались в предыдущем примере, получим следующий результат:

```
$diffArray[a] = яблоко
$diffArray[b] = апельсин
$diffArray[c] = банан
```

В этом случае ни один из элементов массива `$array1` не содержится в `$array2`, поскольку индексы у них различны.

Чтобы найти одинаковые элементы в двух и более массивах, в PHP предусмотрена функция

```
$simArray = array_intersect($массив1, $массив2, ...);
```

Рассмотрим пример использования этой функции для массивов из предыдущих примеров:

```
$simArray = array_intersect($array1, $array2);
```

В результате массив `$simArray` будет иметь такие элементы:

```
$simArray[b] = апельсин
$simArray[c] = банан
```

Функция `array_intersect()` считает элементы одинаковыми, если их значения равны. Для того чтобы учитывать равенство как ключа, так и значения элемента, в языке PHP предназначена функция `array_intersect_assoc()`, которая имеет следующий синтаксис:

```
$simArray = array_intersect_assoc($array1, $array2);
```

В результирующий массив `$simArray` будут записаны только те элементы `$array1` и `$array2`, которые имеют одинаковые ключ и значение. Используя в качестве аргументов все те же массивы, результатом будет пустой массив, поскольку исходные не содержат одинаковых индексов.

Другие операции с массивами

В этом разделе мы рассмотрим другие операции над массивами.

- ✓ Суммирование массивов.
- ✓ Удаление повторяющихся элементов.
- ✓ Перестановка ключей и значений в массиве.

Суммирование массивов

Для сложения всех значений массива следует использовать функцию

```
$sum = array_sum($array);
```

Например:

```
$arrayAdd = array(3, 3, 3);
$sum = array_sum($arrayAdd);
echo $sum;
```

В результате будет получено значение 9.

Просуммировать можно только числовые массивы. Но, как уже упоминалось в главе 5, при использовании (в данном случае при попытке их "сложения") текстовые строки автоматически преобразуются в 0.

Удаление повторяющихся элементов

Очень часто необходимо удалить повторяющиеся элементы в массиве. Например, имеется массив с именами покупателей, который нужно вывести на экран без повторений. Это можно осуществить, используя выражение

```
$names = array("Мэри", "Салли", "Салли", "Сэм");
$names2 = array_unique($names);
```

В результате массив `$names2` будет иметь такой вид:

```
$names2[0] => Мэри
$names2[1] => Салли
$names2[3] => Сэм
```

Как можно увидеть, в результирующем массиве продублированный элемент был удален.

Перестановка ключей и значений в массиве

PHP позволяет менять местами индексы (а точнее, ключи) и значения элементов массива. Пусть, например, имеется следующий массив:

```
$testarray['rose'] = red
$testarray['iris'] = purple
```

Для перестановки значений и ключей следует воспользоваться функцией

```
$arrayFlipped = array_flip($testarray);
```

В результате будем иметь:

```
$testarray['red'] = rose
$testarray['purple'] = iris
```

Многомерные массивы

В предыдущих разделах речь шла об одномерных списках пар ключ–значение. Однако в некоторых случаях необходимо сохранять данные с несколькими ключами. Пусть, например, имеется информация о ценах на некоторые продукты:

- ✓ лук (onion) — 0,50;
- ✓ яблоко (apple) — 2,50;
- ✓ апельсин (orange) — 2,00;
- ✓ бекон (bacon) — 3,50;
- ✓ картофель (potato) — 1,00;
- ✓ ветчина (ham) — 5,00.

Ее можно сохранить в одномерном массиве следующим образом:

```
$foodPrices['onion'] = 0.50;
$foodPrices['apple'] = 2.50;
$foodPrices['orange'] = 2.00;
$foodPrices['bacon'] = 3.50;
$foodPrices['potato'] = 1.00;
$foodPrices['ham'] = 5.00;
```

Тогда в любой момент можно просмотреть информацию о ценах на продукты путем прямого доступа к элементам массива `$foodPrices`. Но представим, что список товаров содержит 3000 наименований. В этом случае необходимо будет просмотреть 3000 элементов массива, чтобы найти информацию о конкретном продукте, например о *луке* (onion) или *ветчине* (ham).

Однако список товаров можно разделить на три группы: овощи (vegetable), фрукты (fruit) и мясные изделия (meat). И тогда необходимо будет просматривать информацию о продуктах только в рамках одной категории, а не весь список, что является более эффективным. Это можно осуществить, используя многомерные массивы:

```
$foodPrices['vegetable']['onion'] = 0.50;
$foodPrices['vegetable']['potato'] = 1.00;
$foodPrices['fruit']['apple'] = 2.50;
$foodPrices['fruit']['orange'] = 2.00;
$foodPrices['meat']['bacon'] = 3.50;
$foodPrices['meat']['ham'] = 5.00;
```

Такой массив называется *многомерным* (multidimensional), поскольку представляет собой массив массивов. На рис. 6.1 показана структура массива `$foodPrices`. На рисунке видно, что массив `$foodPrices` содержит три пары ключ–значение. При этом значение каждого элемента с ключами `vegetable`, `fruit` и `meat` — массив с двумя элементами. Например, элемент с ключом `meat` — это массив с двумя парами ключ–значение: `bacon-3.50` и `ham-5.00`.

$foodPrices	ключ	значение	
		ключ	*значение*
	vegetable	onion	0,50
		potato	1,00
	fruit	orange	2,00
		apple	2,50
	meat	bacon	3,50
		ham	5,00

Рис. 6.1. Структура массива `$foodPrices` – массив массивов

Массив `$foodPrices` является двумерным. Язык PHP также позволяет работать с четырех-, пяти-, шестимерными или массивами большей размерности. Однако их использование может внести большую путаницу в код сценария.

Создание многомерных массивов

PHP позволяет создавать многомерные массивы точно так же, как и одномерные. Например, можно воспользоваться прямым присваиванием значений элементам массива:

```
$foodPrices['vegetable']['potato'] = 1.00;
$foodPrices['fruit']['apple'] = 2.50;
```

Можно также воспользоваться сокращенной записью:

```
transportation['car'][] = "Форд";
transportation['car'][] = "Джип";
```

По умолчанию в качестве индексов будут использоваться целые числа, начиная с 0:

```
transportation[car][0] = Форд;
transportation[car][1] = Джип;
```

Создать многомерный массив можно с помощью выражения `array`:

```
$foodPrices = array(
     "vegetable" => array("potato" =>1.00, "onion" => .50),
     "fruit" => array("apple" => 2.50, "orange" => 2.00));
```

Следует заметить, что массив $foodPrices с индексами vegetable и fruit создается с помощью первого вызова функции array. Его элементы также представляют собой массивы, что приводит к созданию массива массивов. В результате получим следующий многомерный массив:

```
$foodPrices[vegetable][potato] = 1.00
$foodPrices[vegetable][onion] = .50
$foodPrices[fruit][apple] = 2.50
$foodPrices[fruit][orange] = 2.00
```

Вывод многомерных массивов

Для вывода информации о многомерном массиве можно воспользоваться теми же функциями, что и для одномерного, — print_r() или var_dump(). Рассмотрим следующий пример:

```
array(2) {
      ["vegetable"] =>
      array(2){
            ["potato"] =>
            float(1)
            ["onion"] =>
            float(0.5)
      }
      ["fruit"] =>
      array(2){
            ["apple"] =>
            float(2.5)
            ["orange"] =>
            float(2)
      }
}
```

Первая строка указывает на массив, содержащий два элемента. Его первый элемент с индексом vegetable — массив с двумя элементами типа float с индексами potato и onion и значениями 1 и 0.5 соответственно. Второй элемент основного массива с индексом fruit также состоит из двух элементов.

Использование многомерных массивов в выражениях

Для получение значений элементов многомерного массива используются те же процедуры, что и для одномерного. Рассмотрим пример, в котором доступ к элементам осуществляется напрямую:

```
$hamPrice = $foodPrices['meat']['ham'];
```

Для вывода значений можно использовать функцию echo:

```
echo $foodPrices['meat']['ham'];
```

Если же значение элемента необходимо вывести в выражении с двойными кавычками, имя массива следует заключить в фигурные скобки. При этом символ $ (начало имени переменной) должен следовать сразу же за открывающей фигурной скобкой без пробелов:

```
echo "Ветчина стоит \${$foodPrices['meat']['ham']}";
```

Заметим, что перед первым символом $ стоит символ \. Он используется для того, чтобы PHP интерпретировал $ не как символ начала имени переменной, а буквально. В результате будет выведено:

```
Ветчина стоит $5
```

Ранее в этой главе приводилось описание различных функций для преобразования текстовых строк в массивы (и наоборот), массивов в отдельные переменные и осуществления других

операций. Многие из этих операций не имеет смысла использовать для многомерных массивов, так как они будут работать некорректно. Однако это можно осуществить для отдельных элементов, которые сами по себе являются массивами. Рассмотрим, например, функцию `implode()`, которая преобразует массив в текстовую строку. Ее нельзя напрямую использовать для многомерного массива, поскольку его значения — массивы. Но, как уже упоминалось, эту операцию можно выполнить для отдельных элементов:

```
$resString = implode(":", $foodPrices['vegetable']);
```

Это выражение преобразует элементы массива `$foodPrices['vegetable']` в строку, в которой в качестве символа-разделителя используется двоеточие и пробел `": "`. Если результат выполнения функции `implode()` вывести на экран, получим следующее:

```
1: 0.5
```

Будут выведены значения элементов `1` и `0.5` массива `$foodPrices['vegetable']` с индексами `potato` и `onion` соответственно, разделенные двоеточием и пробелом, как указано в функции `implode()`.

Проход по многомерному массиву

Проход по многомерному массиву можно осуществить с помощью функции `foreach` (ее описание см. выше). Поскольку двумерный массив, такой как `$foodPrices`, содержит в качестве элементов два массива, для прохода необходимо дважды использовать оператор `foreach`: один внутри другого. (Использование оператора внутри другого называется *вложением* (nesting).)

Нижеприведенный фрагмент кода позволяет вывести значения элементов многомерного массива.

```
foreach ($foodPrices as $category )
{
      foreach ($category as $food =>$price )
      {
            $f_price =sprintf("%01.2f",$price);
            echo "$food: \$$f_price \n";
      }
}
```

В результате получим:

```
onion: $0.50
potato: $1.00
apple: $2.50
orange: $2.00
bacon: $3.50
ham: $5.00
```

Рассмотрим поэтапно, как интерпретируется использование выражения `foreach`.

1. Обрабатывается первая пара индекс–значение массива `$foodPrices`, и значение элемента записывается в переменную `$category` (которая сама по себе является массивом).
2. Обрабатывается первый элемент массива `$category`: его индекс сохраняется в переменной `$food`, а значение — в `$price`.
3. Значение переменной `$price` преобразуется в денежный формат.
4. Выводятся в одну строку наименование продукта и его цена.
5. Обрабатывается следующий элемент массива `$category`.
6. Формат переменной `$price` преобразуется в денежный, и выводится последующая строка с наименованием продукта и его цены.

7. По достижении конца массива `$category` заканчивается внутренний цикл выражения `foreach`.
8. Обрабатывается следующий элемент массива `$foodPrices` во внешнем цикле `foreach`, в котором значение присваивается переменной `$category`.
9. Шаги 1–8 повторяются до тех пор, пока не будет достигнут конец массива `$category`. Таким образом, будут завершены внутренние и внешние циклы выражения `foreach`.

Другими словами, внешний цикл `foreach` начинается с первого элемента с индексом `vegetable` и значением-массивом, которое присваивается переменной `$category`. В свою очередь, внутренний цикл `foreach` предназначен для прохода по всем элементам массива `$category` и заканчивается по достижении последнего элемента. Затем осуществляется переход во внешний цикл `foreach`, где обрабатывается вторая пара индекс–значение массива `$foodPrices`, и т.д. до конца.

Массивы, встроенные в PHP

В языке PHP имеется несколько встроенных массивов, которые можно использовать при написании сценариев. При этом в них содержится различная полезная информация. Например, данные об используемом сервере (заголовках, настраиваемых путях и размещении сценариев) сохраняются в массиве `$_SERVER`. Так, информация об имени текущего запущенного сценария содержится в переменной `$_SERVER['PHP_SELF']`.

Использование суперглобальных массивов

На данный момент два набора встроенных массивов содержат одну и ту же информацию. Одни из них были включены в PHP 4.1.0 и назывались *суперглобальными* (superglobal) или *автоглобальными* (autoglobals), поскольку могли использоваться в любом фрагменте кода, даже внутри функций. (Функции и использование переменных внутри них описываются в главе 8.) Более ранние версии массивов с длинными именами, такими как `$HTTP_SERVER_VARS`, предварительно необходимо было объявить глобальными, чтобы иметь возможность использовать их в функциях (подробнее об этом — в главе 8). В общем случае рекомендуется использовать новые версии массивов, имена которых начинаются с символа подчеркивания (_). К старым версиям необходимо прибегать только в тех случаях, когда использование версий PHP ниже 4.1.0 неизбежно.

Обновленный конфигурационный файл `php.ini`, используемый в PHP 5, позволяет предотвратить автоматическое создание старых массивов с длинными именами. Вряд ли есть необходимость их использования, однако некоторые старые сценарии все еще могут их использовать. Следующая строка в файле `php.ini` позволяет управлять этими установками:

```
register_long_arrays = On
```

На данный момент по умолчанию это значение равно `On`. Если старые сценарии не используются, это значение стоит изменить на `Off`.

И хотя в данный момент это значение по умолчанию равно `On`, в будущем ситуация может измениться, и это значение станет равным `Off`. Если, например, запускается сценарий и выдается ошибка в строке кода, содержащей переменную типа `$HTTP_GET_VARS`, то следует проверить настройки в файле `php.ini`, чтобы иметь возможность использовать массивы с длинными именами. Если это значение равно `Off`, значит, данные массивы не были созданы вообще и их использование запрещено.

Список встроенных массивов и их краткое описание представлены в табл. 6.2. Детальное описание возможностей специальных массивов приведено в соответствующих главах данной книги. Например, встроенные массивы, содержащие информацию о данных формы, описываются в главе 10.

Таблица 6.2. Встроенные массивы

Массив	Описание
$GLOBALS	Содержит все глобальные переменные. Например, если переменной $testvar присваивается значение 1, к ней можно получить доступ следующим образом: $GLOBALS['testvar']
$_POST	Содержит все переменные формы, переданные с помощью метода POST: method = "post"
$HTTP_POST_VARS	То же, что и $_POST
$_GET	Содержит все переменные, переданные с предыдущей страницы как часть URL-адреса, а также переменные формы, переданные с помощью метода GET: method = "get"
$HTTP_GET_VARS	То же, что и $_GET
$_COOKIE	Содержит все переменные cookie
$HTTP_COOKIE_VARS	То же, что и $_COOKIE
$_SESSION	Содержит все переменные сеанса
$HTTP_SESSION_VARS	То же, что и $_SESSION
$_REQUEST	Содержит все переменные, которые содержатся в массивах $_POST, $_GET и $SESSION
$_FILES	Содержит имена всех загруженных файлов
$HTTP_FILES_VARS	То же, что и $_FILES
$_SERVER	Содержит информацию об используемом сервере. Поскольку Web-серверы бывают разные, то и информация может быть различной
$HTTP_SERVER_VARS	То же, что и $_SERVER
$_ENV	Содержит информацию об операционной системе: имя, системный диск, путь к временному каталогу. Очевидно, что эта информация зависит от используемой операционной системы
$HTTP_ENV_VARS	То же, что и $_ENV

Использование массивов $_SERVER и $_ENV

Массивы $_SERVER и $_ENV позволяют получить информацию об используемом сервере и операционной системе. Для вывода данных, которые содержатся с этих массивах, можно воспользоваться следующим фрагментом

```
foreach($_SERVER as $key => $value)
{
      echo "Ключ = $key, Значение = $value\n";
}
```

В результате получим:

```
Ключ=DOCUMENT_ROOT, Значение=c:/program files/apache
                    group/apache/htdocs
Ключ=PHP_SELF, Значение=/test.php
```

Элемент с ключом DOCUMENT_ROOT содержит путь к каталогу, в котором Web-сервер Apache ищет требуемые Web-страницы. Элемент с индексом PHP_SELF содержит имя сценария, который выполняется в данный момент.

Для просмотра информации в массиве $_ENV можно также воспользоваться функцией phpinfo() с аргументом 16:

```
phpinfo(16);
```

Встроенные массивы можно использовать только тогда, когда включен режим track-vars. По умолчанию он включен начиная с версии PHP 4.0.3, кроме тех случаев, когда администратор намеренно не выключает его при установке PHP. Однако это происходит редко. Но если есть подозрение, что режим track-vars отключен, это следует проверить с помощью функции phpinfo(). И, если это так, необходимо переустановить модуль PHP.

Использование переменных $argv и $argc

Иногда информацию для сценария необходимо передать извне. Одним из возможных способов является передача данных посредством командной строки при запуске сценария. Однако для Web такой метод используется редко, в отличие от запуска PHP CLI из командной строки. Пусть, например, необходимо написать программу, суммирующую два числа, значения которых нужно передать при запуске сценария. Это можно осуществить с помощью следующей команды:

```
php add.php 2 3
```

В этом выражении add.php — имя сценария, а 2 и 3 — числа, которые необходимо сложить. Эти значения можно получить в сценарии, используя массив $argv. В нем содержится следующая информация:

```
$argv[0] = add.php
$argv[1] = 2
$argv[2] = 3
```

Таким образом, массив $argv содержит по крайней мере одно значение — имя сценария.

Затем в сценарии можно воспользоваться операцией суммирования двух чисел:

```
$sum = $argv[1]+$argv[2];
echo $sum;
```

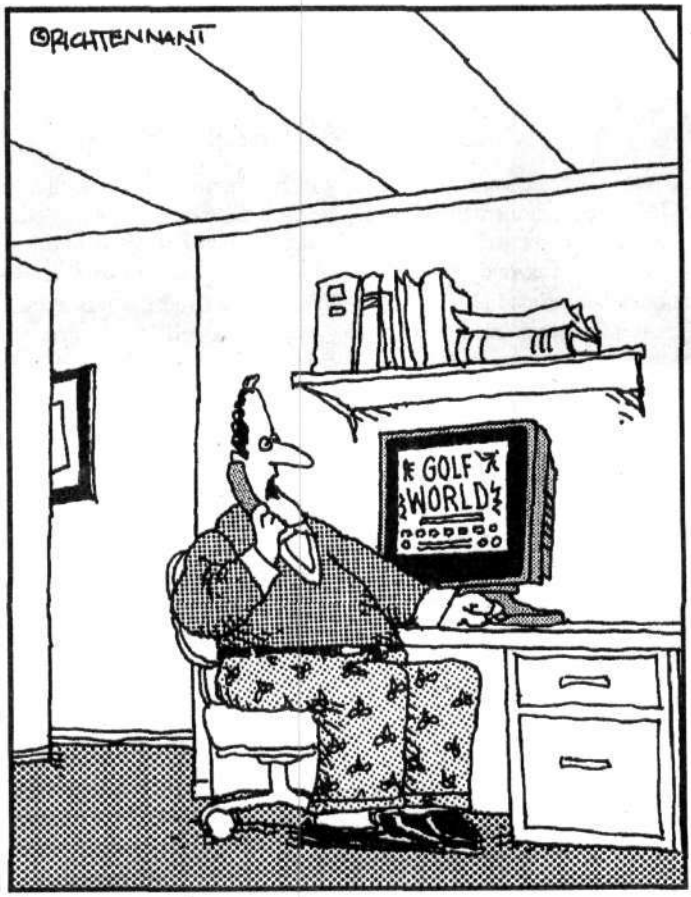
В результате получим значение

```
5
```

Можно воспользоваться также и переменной $argc. Она используется для подсчета количества элементов в массиве $argv. Минимальное значение переменной $argc — 1, которое соответствует имени сценария. В предыдущем примере ее значение равно 3.

Часть III

Основы программирования на PHP



"Я не могу объяснить этот факт, но каждый раз, когда я запускаю игру в гольф, небольшой фрагмент кода исчезает с главной страницы".

В этой части...

В этой части рассказывается о том, как написать полноценный сценарий на языке PHP и объединить разные операторы в окончательный код сценария. Речь пойдет о сложных операторах, направленных на решение сложных задач. Эта часть также будет посвящена вопросам, связанным с повторным использованием кода PHP. После ее изучения вы будете иметь все навыки, необходимые для написания полезных и сложных PHP-сценариев.

Глава 7

Управление ходом выполнения сценария

В этой главе...

- Изменение порядка выполнения операторов в сценарии
- Проверка условий
- Создание сложных условий путем объединения простых
- Использование условий в условных выражениях и циклах
- Использование конструкции `if`
- Создание и использование циклов
- Прерывание циклов

PHP-сценарий представляет собой набор инструкций в файле. Выполнение инструкций осуществляется в порядке их следования в файле, начиная с самого начала. Однако в большинстве случаев необходимо создавать более сложные сценарии. Например, нужно разработать сценарий, который будет отображать одну страницу для новых покупателей и совсем другую для уже существующих, или необходимо вывести на экран список телефонных номеров с помощью повторного использования одной и той же функции `echo`. В этой главе вы узнаете, как изменять порядок выполнения простых операторов с помощью более сложных инструкций, таких как условные операторы и циклы.

Изменение порядка выполнения операторов в сценарии

Простые операторы PHP выполняются по очереди, один за другим, с начала и до конца. Рассмотрим следующий фрагмент кода:

```
$a = "Доброе утро";
echo $a;
$a = "Добрый день";
echo $a;
```

Для того чтобы изменить порядок выполнения операторов, необходимо изменить порядок их следования в файле, например:

```
$a = "Добрый день";
echo $a;
$a = "Доброе утро";
echo $a;
```

Предположим, что нужно вывести приветственное сообщение в зависимости от времени суток, т.е. до полудня необходимо отображать `Доброе утро`, а после полудня — `Добрый день`. Другими словами, нужно сделать следующее:

```
if (время до полудня)
{
      $a = Доброе утро;
```

```
    echo $a;
}
or else if (время после полудня)
{
    $a = Добрый день;
    echo $a;
}
```

Таким образом, для вывода соответствующего приветственного сообщения следует воспользоваться сложным оператором, в котором проверяется время суток. Язык PHP предоставляет два возможных способа, которые позволяют выполнять такие сложные действия.

- ✓ **Условные операторы.** В некоторых случаях необходимо выполнять операторы или какие-либо действия только при выполнении определенных условий. Например, каталог с продукцией необходимо показывать только покупателям, которые оплатили свои счета, и не делать этого для тех, кто имеет задолженность. Такие операторы называются *условными* (conditional statement). В PHP существуют условные операторы `if` и `switch`.
- ✓ **Циклы.** Часто требуется выполнить некоторый блок кода несколько раз подряд. Например, нужно передать сообщения по электронной почте всем покупателям. Чтобы осуществить это, необходимо выполнить две операции: первая из них будет извлекать информацию об электронных адресах покупателей из базы данных, а вторая — отправлять соответствующие сообщения. Причем эти действия нужно осуществить для всех покупателей из базы данных. Операция, которая позволяет выполнять операторы несколько раз подряд, называется *циклом* (loop). В PHP циклы реализуются с помощью операторов `for`, `while` и `do..while`.

Оба рассмотренных типа сложных операторов выполняют некий фрагмент кода в зависимости от условий, т.е. если условие верно, то оператор выполняется. Причем в условных операторах код выполняется один раз. Например, если текущее время является послеобеденным, то будет выведена строка `Добрый день`. В циклах блок кода выполняется несколько раз до тех пор, пока условие не станет ложным. Например, если какой-то покупатель еще не получил электронное сообщение, то его необходимо отправить. И этот цикл повторяется до тех пор, пока сообщения не будут отправлены всем покупателям.

Проверка условий

Условия (condition) — это выражения, которые могут принимать два значения: истина или ложь. Условия применяются в сложных операторах для того, чтобы определить, выполнять последующий блок кода или нет. Для проверки условий необходимо сравнить некоторые значения. При этом возможны следующие варианты вопросов.

- ✓ **Равны ли два значения?** Имеет ли Бобби ту же фамилию, что и Салли? Достиг ли Ник 15-летнего возраста?
- ✓ **Больше или меньше одно значение другого?** Младше ли Ник, чем Бобби? Стоит ли дом Салли больше одного миллиона долларов?
- ✓ **Соответствует ли строка шаблону?** Начинается ли имя Бобби с буквы C? Состоит ли почтовый индекс из пяти цифр?

Можно проверять условия, состоящие из нескольких вопросов. Рассмотрим такие примеры. Верно ли, что Ник старше Бобби и младше Салли? Верно ли, что сегодня воскресенье и солнечный день? Верно ли, что сегодня воскресенье или понедельник?

Использование операций сравнения

В PHP имеется набор операций, предназначенных для сравнения значений (табл. 7.1).

Таблица 7.1. Операции сравнения

Операция	Описание
==	Равнозначны ли значения двух переменных?
===	Одинаковы ли как значения, так и типы двух переменных?
>	Больше ли первое значение, чем второе?
>=	Верно ли, что первое значение не меньше второго?
<	Меньше ли первое значение, чем второе?
<=	Верно ли, что первое значение не больше второго?
!=, <>	Не равны ли значения двух переменных?
!==	Не одинаковы ли значения или типы данных двух переменных?

Сравнивать можно как числовые значения, так и строковые. Текстовые строки ранжируются в алфавитном порядке, причем строки, которые начинаются с прописных букв, предшествуют строкам, начинающимся со строчных, например `SS` идет перед `Sa`. Символы пунктуации также имеют свой порядок следования, и можно говорить о том, что один символ "больше" другого. Однако такое сравнение, например запятой и точки, не имеет большого практического значения.

Текстовые строки можно сравнивать на основе ASCII-кодов. Каждый символ имеет свой ASCII-код, который соответствует целому числу от 0 до 127. И соответственно сравнение текстовых строк осуществляется на основе этого кода. Например, код, соответствующий запятой, равен 44, а точке — 46. Таким образом, можно сказать, что точка "больше" запятой.

Рассмотрим несколько примеров правильных выражений проверки на истинность в PHP.

- ✓ `$a == $b`
- ✓ `$age != 21`
- ✓ `$ageNick < $ageBobby`
- ✓ `$house_price >= 1000000`

Операция сравнения, проверяющая равенство двух значений, состоит из двух символов равенства (==). Одной из наиболее частых ошибок является использование для сравнения одного знака равенства (=). В этом случае происходит присваивание значения переменной. Так, в выражении `if ($weather = "дождливая")` значение переменной `$weather` станет равным `"дождливая"`, сравнение производиться не будет, и результат этого оператора всегда будет истинным.

PHP проверяет условие и возвращает булево значение: `TRUE` (истина) или `FALSE` (ложь). Рассмотрим следующее выражение:

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

Если, например, `$a = 1` и `$b = 1`, то это выражение будет истинным, а в случае `$a = 1` и `$b = 2` — ложным.

Для отрицания при сравнении используется операция восклицания (`!`). Рассмотрим следующий фрагмент кода:

```
$age != 21
```

В этом выражении проверяется, не равно ли значение переменной `$age` числу 21. Если, например, `$age = 20`, это условие истинно.

Проверка содержимого переменной

Иногда необходимо проверить, существует ли переменная или какое она имеет значение. Ниже приведены функции, позволяющие выполнить такие действия.

```
isset($имя_переменной) #Истина, если переменная
                       #объявлена даже без присваивания значения.
empty($имя_переменной) #Истина, если значение переменной
                       #равно нулю или пустой строке,
                       #либо переменная не объявлена.
```

PHP также позволяет проверить тип переменной. Например, для того чтобы проверить, является ли переменная целочисленной, следует воспользоваться функцией

```
is_int($number)
```

Результатом выполнения этой функции является `TRUE`, если переменная `$number` имеет тип `integer`. Рассмотрим подобные функции.

- ✓ `is_array($var2)` проверяет, является ли переменная `$var2` массивом.
- ✓ `is_float($number)` проверяет, является ли переменная `$number` числом с плавающей точкой.
- ✓ `is_null($var1)` проверяет, равно ли значение переменной `$var1` нулю
- ✓ `is_numeric($string)` проверяет, является ли переменная `$string` числовой строкой.
- ✓ `is_string($string)` проверяет, является ли переменная `$string` строкой.

Для проверки обратных условий следует воспользоваться символом восклицания (`!`). Например, при обработке следующего выражения будет получено значение `TRUE`, если переменная не объявлена:

```
!isset($имя_переменной)
```

Использование регулярных выражений

Зачастую необходимо проверить, обладает ли текстовая строка определенными характеристиками, а не просто совпадает с конкретным значением. Например, нужно найти строки, начинающиеся с символа `S` или содержащие числа. В этом случае строки сравниваются с неким шаблоном. Такие выражения называются *регулярными* (regular expression).

Возможно, вам уже приходилось использовать регулярные выражения. Например, при поиске файлов символ звездочки (`*`) часто используется для обозначения любого символа или нескольких символов (`dir ex*.doc` или `ls ex*.txt`). В этом случае `ex*.txt` является шаблоном, которому соответствуют такие строки, как `exam.txt`, `ex33.txt` и `ex3x4.txt`. Проверка регулярных выражений, конечно же, посложнее, чем просто использование символа звездочки.

Одним из наиболее частых применений регулярных выражений является проверка данных, введенных пользователями с помощью форм. Если информация не удовлетворяет определенному формату (шаблону), ее не следует сохранять в базе данных. Например, если пользователь вводит почтовый индекс, он должен состоять либо из пяти цифр, либо еще включать четыре дополнительные цифры через дефис. Другими словами, с помощью регулярных выражений можно проверить, соответствует ли введенная информация нужному шаблону. Если нет — необходимо заставить пользователя ввести ее заново, в корректной форме.

Использование в шаблоне специальных символов

Обычно шаблоны состоят из буквенных (литеральных) и специальных символов. Буквенные символы не имеют какого-то специального назначения. Например, символ e определяет букву e как одну из 26 букв алфавита. В то же время специальные символы имеют в шаблонах свое функциональное назначение. Например, символ звездочки заменяет любой другой символ или несколько вхождений этого символа. В табл. 7.2 приведен список и краткое описание специальных символов, использующихся в шаблонах.

Таблица 7.2. Специальные символы, используемые в шаблонах

Символ	Назначение	Пример	Соответствует шаблону	Не соответствует шаблону
^	Начало строки	^э	экзамен	математический экзамен
$	Конец строки	н$	экзамен	экзамены
.	Любой одиночный символ	..	до, от Более длинные слова также соответствуют этому шаблону, поскольку включают текстовую строку из двух символов	а, 2
?	Предшествующий символ является не обязательным	ger?m	germ, gem	geam
( )	Группирует буквенные символы в последовательность, которую в точности должна содержать строка	g(er)m	germ	Gem, grem
[ ]	Включает набор необязательных символов	g[er]m	gem, grm	germ, gel
[^]	Определяет все символы, кроме указанных	g[^er]m	gym, gum	gem, grem, germ
-	Включает диапазон всех символов, заключенных между двумя (ряд возможных символов)	g[a-c]m	gam, gbm, gcm	gdm, gxm, gal
+	Один или несколько наборов предшествующих символов	bldg[1-3]+	bldg111, bldg132	bldg, bldg555
*	Один, ни одного или несколько вхождений предшествующего символа	ge*m	gm, geeem	germ, grm
{*n*}	Повторение *n* раз	ge{5}m	geeeeem	geeeem, geeeeeem
{*n1*,*n2*}	Определяет диапазон повторений символа (символов)	a{2,5}	aa, aaa, aaaa, 145aaaaa	1, a3
\	Определяет буквенный символ	g*m	g*m	gem, germ
(| |)	Набор альтернативных строк	(Сэм|Салли)	Салли	Сара, Салмон

Некоторые примеры шаблонов

Для создания достаточно сложных шаблонов используются различные комбинации буквенных и специальных символов. Текстовая строка сравнивается с созданным шаблоном, и в случае их взаимного соответствия возвращается истинное значение (TRUE). Рассмотрим различные примеры шаблонов, а также соответствующих и несоответствующих им текстовых строк.

Пример 1

```
^[А-Яа-я].*
```

Этот шаблон определяет строки, начинающиеся с букв, и состоит из двух частей.

- ✓ `^[А-Яа-я]`. Первая часть определяет, что строки должны начинаться с букв (как верхнего, так нижнего регистра).
- ✓ `.*`. Вторая часть указывает на то, что строка должна состоять из одного или нескольких символов.

Этому шаблону удовлетворяют следующие текстовые строки: `играй в футбол`, `Сэм и Я`. Такие строки, как `123` и `?`, ему не соответствуют.

Пример 2

```
Уважаемый (Ким|Рикки)
```

Этот шаблон определяет две различные строки и состоит из двух частей.

- ✓ `Уважаемый`. Первая часть соответствует простому набору букв.
- ✓ `(Ким|Рикки)`. Вторая часть соответствует либо значению `Ким`, либо `Рикки`.

Шаблону `Уважаемый (Ким|Рикки)` соответствуют текстовые строки `Уважаемый Ким` и `Уважаемый Рикки`.

Такие выражения, как `Уважаемый Боб` и `Ким`, не удовлетворяют этому шаблону.

Пример 3

```
^[0-9]{5}(\-[0-9]{4})?$
```

Этот шаблон определяет общий формат почтового индекса и состоит из нескольких частей.

- ✓ `^[0-9]{5}`. Первая часть определяет строку, состоящую из пяти цифр.
- ✓ `\-`. Символ обратной косой черты (`\`) указывает на то, что дефис (`-`) в данном случае используется в качестве буквенного, а не специального символа.
- ✓ `[0-9]{4}`. Эта часть шаблона определяет строку, состоящую из четырех цифр.
- ✓ `( )?`. Этот набор символов включает в себя последние две части шаблона и указывает на то, что они являются необязательными.
- ✓ `$`. Символ доллара (`$`) является символом конца строки (т.е. после шаблона не должны стоять никакие другие символы).

Этому шаблону удовлетворяют текстовые строки `90001` и `90002-4323`.

Такие строки, как `9001` и `12-4321`, ему не соответствуют.

Пример 4

```
^.+@.+\.com$
```

Этот шаблон определяет любую строку, содержащую символ `@` и имеющую окончание `.com`. Другими словами, он задает адрес электронной почты. Этот шаблон состоит из следующих частей.

- ✓ `^.+`. Первая часть определяет строку, которая начинается с любого символа и предшествует символу `@`.
- ✓ `@`. Этот символ является буквенным, а не специальным, поэтому нет необходимости ставить перед ним символ обратной косой черты (`\`).
- ✓ `.+`. Этот набор специальных символов определяет строку, состоящую из одного или нескольких символов.

- ✓ `\.` . Символ \ указывает на то, что символы точки (.) должны интерпретироваться как буквенные символы.
- ✓ `com$`. Эта часть шаблона определяет, что строка должна обязательно заканчиваться символами com.

Такому шаблону удовлетворяют текстовые строки `you@yourcompany.com` и `johndoe@somedomain.com`.

Такие строки, как `you@yourcompany.net`, `you@.com` и `@you.com`, ему не соответствуют.

Сравнение строк с шаблоном

Для сравнения строк с шаблоном можно воспользоваться функцией `ereg()`. Ее синтаксис имеет следующий вид:

```
ereg("шаблон", значение);
```

Например, для того чтобы проверить правильность имени, введенного пользователем в форме, необходимо сравнить переменную $name (в которой оно хранится) с шаблоном:

```
ereg("^[A-Яа-я' -]+$", $name)
```

Данный шаблон интерпретируется следующим образом.

- ✓ Символы ^ и $ определяют начало и конец текстовой строки соответственно. Это означает, что строка должна полностью соответствовать созданному шаблону.
- ✓ Все разрешенные буквенные символы строки заключаются в квадратные скобки. К ним относятся буквы верхнего и нижнего регистров, апостроф, пробел и дефис.

  В пределах квадратных скобок можно задавать область изменения символов (например, А-Я). При этом дефис интерпретируется не как буквенный, а как специальный символ. Для того чтобы изменить это правило, необходимо, чтобы дефис не был заключен между двумя символами, т.е. его следует разместить в конце, после набора букв.
- ✓ После квадратных скобок следует символ "плюс". Он указывает на то, что строка может содержать любое количество символов, заданных в квадратных скобках, но не меньше одного.

Объединение условий

Часто бывает необходимо проверить несколько условий одновременно. Предположим, например, что компания имеет каталог продуктов на различных языках. Необходимо определить, описаниями каких продуктов интересуется покупатель и на каком языке. Для этого следует объединить несколько условий. Общий синтаксис имеет следующий вид:

```
условие1 and|or|xor условие2 and|or|xor условие3 and|or|xor ...
```

Условия можно объединять с помощью следующих трех выражений (логических операторов).

- ✓ `and` (логическое И). Результирующее условие истинно, если истинны оба условия.
- ✓ `or` (логическое ИЛИ). Результирующее условие истинно, если хотя бы одно из условий истинно.
- ✓ `xor` (логическое исключающее ИЛИ). Результирующее условие истинно, если истинно одно из условий, но не оба одновременно.

В табл. 7.3 приведены примеры сложных условий, состоящих из более простых.

Таблица 7.3. Сложные условия

Условие	Истина, если...
`$ageBobby == 21 or $ageBobby == 22`	Бобби 21 или 22 года
`$ageSally > 29 and $state == "OR"`	Салли больше 29 лет, и она живет штате Орегон
`$ageSally > 29 or $state == "OR"`	Салли больше 29 лет или она живет штате Орегон
`$city == "Рено" xor $state == "OR"`	Значение переменной `$city` равно `Рено` или значение переменной `$state` — `OR`, но не одновременно
`$name != "Сэм" and $age < 13`	Любое имя, кроме `Сэм`, и возраст не старше 13 лет

Можно объединить любое количество условий. При этом порядок проверки условий следующий: наибольший приоритет имеют выражения, содержащие `and`, затем `xor` и `or`. Рассмотрим следующее составное условие, состоящее из трех простых:

```
$resCity == "Рено" or $resState == "NV" and $name == "Салли"
```

Если имя покупателя `Салли` и она живет в штате Невада (NV), то результирующее выражение истинно. Тот же результат будет получен и в том случае, если покупатель живет в городе `Рено`, независимо от его имени. Это условие ложно, если покупатель обитает в Неваде, но его именем является не `Салли`. Все эти результаты получаются в результате следующего порядка проверки условий.

1. Обрабатывается выражение, содержащее логическое И (`and`). Интерпретатор PHP проверяет, равны ли значения переменных `$resState` и `$name` соответственно NV и `Салли`. Если оба значения совпадают, то условие верно и нет необходимости проверять условие, содержащее `or`. В противном случае PHP переходит к обработке следующего (по приоритету) выражения.
2. Обрабатывается выражение, содержащее логическое ИЛИ (`or`). Интерпретатор PHP проверяет значение переменной `$resCity`. Если оно равно `Рено`, то условие истинно, иначе — ложно.

Для того чтобы изменить порядок обработки логических операторов, следует воспользоваться скобками (`()`). Тогда условия, заключенные в скобки, будут иметь больший приоритет. Например, предыдущий пример можно переписать следующим образом:

```
($resCity == "Рено" or $resState == "NV") and $name == "Салли"
```

В этом случае скобки позволяют изменить порядок проверки условий, т.е. сначала будет обрабатываться выражение, содержащее логическое ИЛИ (`or`). Результирующее условие будет истинным в том случае, если имя покупателя `Салли` и она живет либо в городе `Рено` либо в штате Невада (NV). Этот результат получен благодаря следующему порядку проверки условий.

1. Обрабатывается выражение, содержащее логическое ИЛИ (`or`). Интерпретатор PHP проверяет, равны ли значения переменных `$resCity` и `$resState` соответственно `Рено` и NV. Если обе эти переменные имеют другие значения, то результирующее условие ложно и проверка прекращается. В противном случае PHP переходит к обработке следующего (по приоритету) выражения.
2. Обрабатывается выражение, содержащее логическое И (`and`). PHP проверяет значение переменной `$name`. И если оно равно `Салли`, то результирующее условие истинно, иначе — ложно.

Скобки стоит использовать даже в тех случаях, когда вы уверены в порядке обработки условий. Лишние скобки не помешают. В противном случае результат проверки условий может оказаться самым неожиданным.

Если вы знакомы с другими языками программирования, например C, то вам, наверное, приходилось использовать такие операторы, как `||` (в качестве логического ИЛИ `or`) и `&&` (в качестве логического И `and`). Эти операторы можно применять и в языке PHP. Например, выражение `$a < $b && $c > $b` эквивалентно `$a < $b and $c > $b`. При этом оператор `||` проверяется перед `or`, а `&&` — перед `and`.

Использование условных операторов

Условный оператор (conditional statement) выполняет фрагмент кода только при соблюдении определенного условия. Рассмотрим два наиболее полезных вида условных операторов.

- ✓ **Оператор** `if`. Проверяет условие и, если оно истинно, выполняет соответствующий фрагмент кода.
- ✓ **Оператор** `switch`. Проверяет набор альтернативных условий и, если одно из условий истинно, выполняет соответствующий фрагмент кода.

Использование оператора `if`

Оператор `if` предполагает проверку некоторого условия и, если оно истинно, выполнение соответствующего блока инструкций. Общий синтаксис оператора `if` имеет следующий вид:

```
if (условие)
{
     фрагмент кода
}
elseif (условие)
{
     фрагмент кода
}
else
{
     фрагмент кода
}
```

Оператор `if` состоит из следующих элементов.

- ✓ `if`. Обязательная часть, проверяющая условие.
  - **Если условие истинно,** выполняется первый фрагмент кода. После выполнения операторов этого блока интерпретатор PHP переходит к обработке следующего за `if` оператора. Если условный оператор содержит также блок `elseif` или `else`, они пропускаются.
  - **Если условие ложно,** первый фрагмент кода не выполняется и интерпретатор PHP переходит к обработке следующей за этим фрагментом инструкции, которой может оказаться блок `elseif`, `else` или любой другой оператор, следующий за `if`.

- ✓ `elseif`. Необязательный раздел оператора `if`. При желании можно использовать несколько разделов `elseif`. В этих разделах тоже проверяется истинность некоторого условия.
  - **Если условие истинно,** выполняется соответствующий фрагмент кода. После выполнения всех операторов этого фрагмента PHP переходит к обработке следующего за `if` оператора. Если условный оператор содержит несколько блоков `elseif` и `else`, они игнорируются.
  - **Если условие ложно,** соответствующий фрагмент кода не выполняется и PHP переходит к обработке следующего блока оператора `if` (т.е. блока `elseif` или `else`), а при его отсутствии — к следующему за `if` оператору.
- ✓ `else`. Этот раздел тоже является необязательным. При этом в рамках одного оператора `if` блок `else` можно использовать только один раз. В блоке `else` условие не проверяется, а выполняется фрагмент кода, если все условия, содержащиеся в предыдущих блоках оператора `if`, ложны.

Рассмотрим пример использования оператора `if`. Представьте, что вы — учитель. Следующий оператор передает студенту оценку и небольшое текстовое сообщение, в зависимости от количества баллов, набранных при выполнении теста:

```
if ($score > 92)
{
     $grade = "A";
     $message = "Отлично";
}
elseif ($score <= 92 and $score > 83)
{
     $grade = "B";
     $message = "Хорошо";
}
elseif ($score <= 83 and $score > 74)
{
     $grade = "C";
     $message = "Удовлетворительно";
}
elseif ($score <= 74 and $score > 62)
{
     $grade = "D";
     $message = "Неудовлетворительно";
}
else
{
     $grade = "F";
     $message = "Хуже некуда!";
}
echo $message. "\n";
echo "Ваша оценка $grade\n";
```

Оператор `if` обрабатывается следующим образом.

1. Значение переменной `$score` сравнивается с `92`.

   Если оно больше этого числа, переменной `$grade` присваивается `A`, а переменной `$message` — значение `Отлично`, и интерпретатор PHP переходит к строкам, в которых информация выводится с использованием функции `echo`. Если `$score` меньше

или равно 92, переменные $grade и $message не инициализируются и интерпретатор PHP переходит к инструкции elseif.

2. Значение переменной $score сравнивается с числами 92 и 83.

   Если оно меньше или равно 92 и больше 83, выполняется инициализация переменных $grade и $message, и интерпретатор PHP переходит к строкам, в которых информация выводится с использованием функции echo. Если $score меньше или равно 83, переменные $grade и $message не инициализируются и интерпретатор PHP переходит ко второй инструкции elseif.

3. Значение переменной $score сравнивается с 83 и 74.

   Если оно меньше или равно 83 и больше 74, выполняется инициализация переменных $grade и $message и интерпретатор PHP переходит к строкам, в которых информация выводится с использованием функции echo. Если $score меньше или равно 74, переменные $grade и $message не инициализируются и интерпретатор PHP переходит к следующей инструкции elseif.

4. Значение переменной $score сравнивается с 74 и 62.

   Если оно меньше или равно 74 и больше 62, осуществляется инициализация переменных $grade и $message и интерпретатор PHP переходит к выводу данных с использованием функции echo. Если $score меньше или равно 62, переменные $grade и $message не инициализируются и интерпретатор PHP переходит к инструкции else.

5. Переменной $grade присваивается значение F, а $message — "Хуже некуда!", и интерпретатор PHP переходит к выводу информации с использованием функции echo.

Если фрагмент кода, выполняемый после проверки условия, состоит из одного выражения, фигурные скобки можно опустить. Допустим, в предыдущем примере инициализировалась только одна переменная $grade, т.е.

```
if ($grade > 92)
{
  $grade = "A";
}
```

Теперь это выражение можно переписать следующим образом:

```
if ($grade > 92)
   $grade = "A";
```

Такая сокращенная запись позволяет избежать написания лишнего кода. Однако, если имеется несколько выражений, это может привести к путанице.

Отрицание в операторе if

Если необходимо, чтобы блок операторов выполнялся в случае невыполнения некоторого условия, перед этим условием следует поставить восклицательный знак (!). Рассмотрим для примера следующее выражение:

```
if (ereg("^C[a-я]*", $string))
{
   $list[] = $string."\n";
}
```

Этот условный оператор проверяет, начинается ли строка $string с символа C. Другими словами, если строка $string соответствует шаблону, определяющему строки, которые начинаются с символа C, за которым следуют буквы нижнего регистра, — блок операторов выполняется. Если перед условием поставить восклицательный знак, ситуация кардинально изменится:

```
if (!egreg("^C[а-я]*", $string)
{
   $list[] = $string."\n";
}
```

В этом случае из-за наличия восклицательного знака (!) перед условным выражением массив `$list` будет содержать все строки, кроме тех, которые начинаются с символа С, т.е. условие истинно, если строка `$string` не начинается с символа С.

Вложенные операторы `if`

Условный оператор `if` можно использовать в другом условном операторе. Такое действие называется *вложением* (nesting). Пусть, например, необходимо связаться со всеми клиентами, которые живут в штате Айдахо (Idaho, ID). При этом клиентам, имеющим адрес электронной почты, нужно отправить электронное сообщение, а тем, кто его не имеет, — письмо по обычной почте. Это можно сделать с помощью вложенного оператора `if`:

```
if ($custState == "ID")
{
      if ($EmailAdd = "")
      {
            $contactMethod = "по обычной почте";
      }
      else
      {
            $contactMethod = "по электронной почте";
      }
}
else
{
      $contactMethod = "нет необходимости";
}
```

Первое условие позволяет узнать, проживает ли клиент в штате Айдахо. Если это так, сценарий PHP проверит наличие адреса электронной почты. Если у клиента его нет, переменной `$contactMethod` (которая определяет вид связи) присваивается значение `по обычной почте`, иначе — `по электронной почте`. Если же клиент проживает не в штате Айдахо, ему ничего не отправляется.

Оператор `switch`

Во многих ситуациях оператор `if` работает отлично. Однако иногда имеется список условий, для каждого из которых необходимо выполнить свой набор операторов. Например, нужно написать сценарий, который будет вычислять налог с оборота. Как в этом случае учесть разную налоговую ставку для различных штатов? Именно для таких ситуаций и предназначен оператор `switch`.

Оператор `switch` проверяет значение некоторой переменной, и если оно совпадает с имеющимся в перечне, выполняются соответствующие операторы. Общий синтаксис инструкции `switch` имеет следующий вид:

```
switch ($имя_переменной)
{
      case значение:
            фрагмент кода;
            break;
      case значение:
            фрагмент кода;
```

```
            break;
      ...
      default:
            фрагмент кода;
            break;
}
```

При выполнении оператора `switch` интерпретатор PHP пытается найти строку `case`, содержащую *значение*, равное значению переменной *$имя_переменной*. Если такая строка обнаружена, выполняется соответствующий фрагмент кода до оператора `break`. Если же найти такую строку не удалось, выполняется фрагмент кода, следующий после ключевого слова `default`. В операторе `switch` можно задавать любое количество вариантов. При этом использование ключевого слова `default` не обязательно, но при его наличии строку `default` обычно ставят в конце оператора `switch`. Однако для интерпретатора PHP это не имеет значения.

Рассмотрим следующий пример, учитывающий различные ставки на налог с оборота для различных штатов:

```
switch ($custState)
{
      case "OR":
            $salestaxrate = 0;
            break;
      case "CA":
            $salestaxrate = 1.0;
            break;
      default:
            $salestaxrate = .5;
            break;
}
$salestax = $orderTotalCost*$salestaxrate;
```

В этом примере налог с оборота в штате Орегон равен 0%, в Калифорнии — 100%, а для всех остальных штатов — 50%. При выполнении оператора `switch` интерпретатор PHP пытается найти строку `case`, содержащую значение, равное значению переменной `$custState`. Если такая строка обнаружена, переменной `$salestaxrate` присваивается определенное значение. Например, если `$custState` равно `TX` (штат Техас), осуществляется переход к строке с ключевым словом `default` и переменной `$salestaxrate` присваивается значение `.5`. После выполнения оператора `switch` значение переменной `$salestax` будет равно половине (`.5`) от суммы заказа (`$orderTotalCost`).

Необходимо обратить внимание на операторы `break`, расположенные в конце каждого блока `case`. Если бы не они, интерпретатор PHP продолжал бы выполнять операторы либо до следующего оператора `break`, либо до конца инструкции `switch`. Это касается всех блоков `case`, кроме последнего.

Последний блок `case` оператора `switch` не требует оператора `break`. Однако его *стоит* использовать для большей ясности и согласованности кода.

Повторение действий с помощью циклов

Циклы очень часто используются в сценариях для многократного выполнения некоторых действий. При этом операции цикла повторяются либо некоторое наперед заданное количество раз, либо до выполнения определенных условий. Например, цикл, который выводит со-

кращенные названия штатов с помощью функции echo, необходимо повторить 50 раз, а цикл, который выводит имена файлов в каталоге, — пока не будут обработаны все файлы (независимо от их количества). В языке PHP существуют три типа циклов.

- ✓ for. Проверяет значение счетчика цикла и повторяет действия до тех пор, пока счетчик цикла не достигнет определенного значения.
- ✓ while. Проверяет условие и выполняет действия, пока условие не станет ложным.
- ✓ do..while. Выполняет действия и только потом проверяет условие. Если оно истинно — операторы цикла повторяются, в противном случае цикл прекращается.

В следующих разделах эти виды циклов рассматриваются более подробно.

Цикл for

Традиционный цикл for основан на счетчике. Сначала осуществляется инициализация переменной-счетчика, потом указывается ее конечное значение и определяется способ изменения состояния счетчика после выполнения набора действий. Общий синтаксис цикла for имеет следующий вид:

```
for (начальное_значение; конечное_условие; инкремент)
{
     фрагмент кода;
}
```

Цикл for имеет следующие параметры.

- ✓ *начальное_значение*. Это выражение определяет переменную-счетчик цикла и инициализирует ее начальным значением. Например, выражение $i=1 указывает, что в качестве счетчика будет использоваться переменная $i, а ее начальное значение равно 1. Обычно счетчик начинается с 0 или 1. При этом в качестве начального значения можно использовать число, числовое выражение (например, 2+2) или значение переменной.
- ✓ *конечное_условие*. Это выражение определяет конечное значение счетчика. Цикл выполняется до тех пор, пока счетчик не достигнет этого значения. Рассмотрим, например, выражение $i<10, которое устанавливает конечное значение, равное 10. При достижении переменной $i этого значения цикл прекращается, поскольку условие $i<10 уже не будет выполняться. Выражение *конечное_условие* может также иметь вид $i<$size;.
- ✓ *инкремент*. Выражение, которое определяет, на какую величину будет изменяться счетчик. Например, выражение $i++; будет увеличивать значение переменной $i на единицу после выполнения каждой итерации (блока цикла). Можно также использовать такие выражения, как $i=+1; или $i--;.

В цикле for в качестве счетчика используется некая переменная, скажем $i, значение которой изменяется в процессе выполнения цикла. При этом переменную-счетчик $i можно также использовать внутри цикла. Рассмотрим пример, в котором строка Здравствуй, мир! выводится три раза:

```
for ($i=1; $i<=3; $i++)
{
     echo "$i. Здравствуй, мир!<br>";
}
```

Операторы тела цикла необязательно вводить с отступом. Для интерпретатора PHP это не имеет значения. Отступ используется для лучшего понимания кода сценария.

В результате получим следующее:

```
1. Здравствуй, мир!
2. Здравствуй, мир!
3. Здравствуй, мир!
```

Вложенные циклы for

Один цикл `for` можно использовать внутри другого. Пусть, например, необходимо вывести таблицу умножения чисел от 1 до 9. Это можно сделать, используя следующий фрагмент кода:

```
for ($i=1; $i<=9; $i++)
{
      echo "\nУмножение на $i \n";
      for($j=1; $j<=9; $j++)
      {
            $result = $i * $j;
            echo "$i x $j = $result\n";
      }
}
```

В результате получим следующий результат:

```
Умножение на 1
1 x 1 = 1
1 x 2 = 2
...
1 x 8 = 8
1 x 9 = 9

Умножение на 2
2 x 1 = 2
2 x 2 = 4
...
2 x 8 = 16
2 x 9 = 18

Умножение на 3
3 x 1 = 3
```

и так далее.

Усложненные циклы for

Структура циклов `for` является достаточно гибкой и позволяет создавать циклы для самых разнообразных целей. Общий синтаксис цикла `for` можно представить в следующем виде:

```
for (начальное выражение; условное выражение; заключительное
выражение)
{
      фрагмент кода;
}
```

Параметры рассмотренного цикла `for` имеют следующее назначение.

- ✓ Начальное выражение выполняется безусловно, один раз, до начала цикла. Оно может включать присваивание начального значения или любое другое выражение, которое требуется выполнять перед началом цикла.
- ✓ Условное выражение проверяется на истинность перед началом каждой итерации цикла.
- ✓ В конце каждой итерации цикла выполняется заключительное выражение. Оно может включать инкремент счетчика цикла или любое другое действие, которое выполняется после каждой итерации цикла.

Рассмотренные выше выражения разделяются точкой с запятой (;). В свою очередь, каждое из них может состоять из нескольких выражений, разделенных запятой (,), или быть пустым.

Рассмотрим следующий пример:

```
$t = 0;
for ($i=0, $j=1; $t<=4; $i++, $j++)
{
      $t = $i + $j;
      echo "$t<br>";
}
```

В качестве начального выражения используется `$i=0, $j=1`, в качестве условного — `$t<=4`, заключительного — `$i++, $j++`.

Результат выполнения приведенного фрагмента будет иметь следующий вид:

```
1
3
5
```

Теперь рассмотрим более подробно процесс выполнения этого цикла.

1. Выполняется начальное выражение. При этом переменной `$i` присваивается значение 0, а переменной `$j` — 1.
2. Проверяется условное выражение. Не превышает ли значение переменной `$t` числа 4? Поскольку это так, то условие истинно и цикл продолжает выполняться.
3. Выполняются действия внутри тела цикла. Значение переменной `$t` вычисляется как сумма `$i` и `$j` (0+1 равно 1) и с помощью функции `echo` выводится на экран.
4. В конце данной итерации выполняется заключительное выражение: `$i++` и `$j++`. Значения переменных `$i` и `$j` увеличиваются на единицу и становятся равными 1 и 2.
5. В начале следующей итерации проверяется условное выражение. Не превышает ли значение переменной `$t` числа 4? Поскольку текущее значение `$t` равно 1, то условие истинно и цикл продолжает выполняться.
6. Выполняются действия внутри цикла. Значение переменной `$t` вычисляется как сумма `$i` и `$j` (1+2 равно 3) и с помощью функции `echo` выводится на экран.
7. В конце данной итерации выполняется заключительное выражение: `$i++` и `$j++`. Значения переменных `$i` и `$j` увеличиваются на единицу и становятся равными 2 и 3.
8. В начале следующей итерации проверяется условное выражение. Не превышает ли значение переменной `$t` числа 4? Поскольку текущее значение `$t` равно 3, то условие верно и цикл продолжает выполняться.
9. Выполняются действия внутри тела цикла. Значение переменной `$t` вычисляется как сумма `$i` и `$j` (2+3 равно 5) и с помощью функции `echo` выводится на экран.
10. В конце данной итерации выполняется заключительное выражение: `$i++` и `$j++`. Значения переменных `$i` и `$j` увеличиваются на единицу и становятся равными 3 и 4.

11. В начале следующей итерации проверяется условное выражение. Не превышает ли значение переменной `$t` числа 4? Поскольку текущее значение `$t` равно 5, то условие ложно и цикл завершается.

Цикл `while`

Цикл `while` выполняет заданные действия до тех пор, пока истинно определенное условие. Этот вид цикла работает следующим образом.

1. Задается условие.
2. Это условие проверяется перед началом каждой итерации.
3. Если условие истинно, цикл продолжает выполняться, в противном случае — завершается.

Общий синтаксис цикла `while` имеет следующий вид:

```
while ( условие )
{
   фрагмент кода;
}
```

В следующем примере цикл `while` используется для нахождения в массиве `$fruit` значения `яблоко`:

```
$fruit = array("апельсин", "яблоко", "виноград");
$testvar = "нет";
$k = 0;
while ( $testvar != "да" )
{
     if ($fruit[$k] == "яблоко" )
     {
          $testvar = "да";
          echo "яблоко\n";
     }
     else
     {
          echo "$fruit[$k] - это не яблоко\n";
     }
     $k++;
}
```

В результате будет выведено следующее:

```
апельсин - это не яблоко
яблоко
```

Рассмотрим процесс выполнения этого цикла более подробно.

1. Перед началом цикла инициализируются необходимые переменные. Создается массив `$fruit` с тремя элементами, тестовая переменная `$testvar` со значением `нет` и счетчик `$k`, принимающий значение `0`.
2. Цикл начинается с проверки на истинность условия `$testvar != "да"`. Поскольку переменной `$testvar` было присвоено значение `нет`, условие верно и цикл не завершается.
3. Проверяется условие в выражении `if`. Верно ли, что `$fruit[$k] == "яблоко"`? Текущее значение `$k` равно `0`, и интерпретатор PHP проверяет на равенство первый элемент массива `$fruit[0]`. Поскольку в нем содержится значение `апельсин`, то рассматриваемое условие ложно, т.е. фрагмент кода, содержащийся после `if`, не выполняется и интерпретатор PHP переходит к обработке инструкции `else`.

4. Выполняется фрагмент кода, содержащийся после `else`. Он выводит первую строку `апельсин – это не яблоко`.
5. Значение переменной `$k` увеличивается на единицу и становится равным 1.
6. Достигается конец цикла и интерпретатор PHP переходит к его началу.
7. Снова проверяется на истинность условие `$testvar != "да"`. Поскольку значение переменной `$testvar` не изменилось и равно `нет`, условие истинно и цикл не прекращается.
8. Опять проверяется условие в выражении `if`. Верно ли, что `$fruit[$k] == "яблоко"`? Теперь текущее значение `$k` равно 1 и интерпретатор PHP проверяет второй элемент массива `$fruit[1]`. Поскольку в нем содержится значение `яблоко`, то рассматриваемое условие истинно, т.е. выполняется фрагмент кода, который следует за `if`.
9. Выполняется фрагмент кода после оператора `if`. В нем переменной `$testvar` присваивается значение `да` и выводится вторая строка `яблоко`.
10. Значение переменной `$k` снова увеличивается на единицу и становится равным 2.
11. Достигается конец цикла, и интерпретатор PHP снова переходит к его началу.
12. Последний раз проверяется истинность условия `$testvar != "да"`. Поскольку значение переменной `$testvar` изменилось и стало равным `да`, условие ложно и цикл завершается.

Вполне возможно создать цикл `while`, который будет бесконечным, т.е. будет выполняться неограниченное количество раз. Это может произойти из-за того, что условие всегда будет оставаться истинным, т.е. если условие никогда не станет ложным, цикл никогда не завершится. Более подробно этот вопрос рассматривается ниже.

Цикл `do..while`

Цикл `do..while` очень схож с циклом `while` и выполняется до тех пор, пока истинны определенные условия. Однако в цикле `do..while` условие проверяется не перед началом каждой итерации, а после ее завершения.

Общий синтаксис цикла `do..while` имеет следующий вид:

```
do
{
     фрагмент кода;
}while ( условие );
```

Перепишем предыдущий пример цикла `while` с использованием цикла `do..while`. При этом результат выполнения будет один и тот же:

```
$fruit = array ("апельсин", "яблоко", "виноград");
$testvar = "нет";
$k = 0;
do
{
     if ($fruit[$k] == "яблоко"))
     {
          $testvar = "да";
          echo "яблоко\n";
     }
     else
     {
          echo "$fruit [$k] – это не яблоко\n";
```

```
    }
    $k++;
}while ($testvar != "да"));
```

В результате на экране отобразятся следующие строки:

```
апельсин – это не яблоко
яблоко
```

т.е. будет получен тот же результат, что и в предыдущем примере с использованием цикла `while`. Различие между циклами `while` и `do..while` заключается во времени проверки условия. В цикле `while` условие проверяется перед началом каждой итерации, т.е. если оно всегда ложно, фрагмент кода никогда не выполнится. В цикле `do..while` условие проверяется после завершения итерации. Поэтому фрагмент кода всегда выполнится хотя бы один раз.

Изменим, например, в предыдущих двух циклах, `while` и `do..while`, начальное значение переменной `$testvar` на да:

```
$testvar = "да";
```

В первом из них сначала проверяется условие, которое всегда ложно. Следовательно, цикл `while` никогда не будет выполнен. Во втором цикле `do..while` итерация выполнится хотя бы один раз, и будет выведена одна строка:

```
апельсин – это не яблоко
```

Таким образом, в результате выполнения цикла `do..while` перед проверкой условия будет выведена одна строка. Поскольку оно всегда ложно, последующие действия выполняться не будут.

Избегайте бесконечных циклов

Нетрудно создать цикл, который будет выполнятся неограниченное количество раз. Такие циклы называются *бесконечными* (infinite loop). Однако вряд ли кто-то создает их намеренно. Чаще всего — это ошибка в написании кода. Например, небольшое изменение фрагмента кода в примере с циклом `while` может привести к бесконечному циклу.

Модифицируем код сценария, который использовался ранее в этой главе:

```
$fruit = array("апельсин", "яблоко", "виноград");
$testvar = "нет";
while ( $testvar != "да" )
{
    $k = 0;
    if ($fruit[$k] == "яблоко" )
    {
        $testvar = "да";
        echo "яблоко\n";
    }
    else
    {
        echo "$fruit[$k] – это не яблоко\n";
    }
    $k++;
}
```

Небольшое изменение касается выражения `$k = 0;`, которое было внесено в цикл `while` извне. Эта модификация делает цикл бесконечным. В результате получим:

```
апельсин – это не яблоко
апельсин – это не яблоко
апельсин – это не яблоко
апельсин – это не яблоко
...
```

и так до бесконечности. При каждой новой итерации переменной $k присваивается значение 0 (несмотря на инкремент в конце цикла), и каждый раз проверяется первый элемент массива $fruit[0], содержащий значение апельсин, т.е. второй элемент массива (со значением яблоко) никогда не будет обработан, и значение переменной $testvar никогда не станет равным да. Таким образом, в результате получим бесконечный цикл.

Не смущайтесь, если напишете бесконечный цикл. Даже самые известные программисты в мире создавали их в большом количестве. Если при тестировании сценария вы обнаружили, что некоторое действие повторяется бесконечно, не делайте из этого трагедию, а выполните одно из следующих действий.

- ✓ **При использовании PHP для создания Web-страниц**. Подождите, пока сценарий сам не остановится через короткое время. По умолчанию оно равно 30 секундам, но может быть изменено администратором. Или щелкните на кнопке Стоп (Stop) в браузере, чтобы остановить выполнение сценария.
- ✓ **При использовании PHP CLI**. Нажмите комбинацию клавиш <Ctrl+C>. Это поможет остановить выполнение сценария. Хотя некоторое время он еще может поработать, но остановится обязательно.

Затем найдите строку кода, генерирующую бесконечный цикл, и исправьте ее.

Наиболее типичной ошибкой при написании циклов является использование при проверке условий одного знака равенства (=) вместо двух (==). Один знак равенства предназначен для присваивания значений переменным, а два знака — для проверки условий. Соответственно следующее выражение всегда истинно:

```
while ($testvar = "да")
```

В нем переменной $testvar присваивается значение да. Это выражение никогда не станет ложным. Здесь, наверное, подразумевалось следующее:

```
while ($testvar == "да")
```

В отличие от предыдущего выражения здесь значение переменной $testvar сравнивается на равенство со значением да. При этом может быть получена как истина, так и ложь.

Еще одной ошибкой является отсутствие выражения, которое изменяет значение переменной-счетчика. Например, если в предыдущем примере кода удалить инкремент счетчика $k++;, то значение переменной $k будет неизменным и равным 0 и в результате получится бесконечный цикл.

Прерывание циклов

Иногда необходимо прервать выполнение цикла. В языке PHP это можно сделать двумя способами.

- ✓ Оператор break полностью прерывает выполнение цикла.
- ✓ Оператор continue прерывает выполнение текущей итерации и переходит к следующей, если соответствующее условие выполнено.

Операторы break и continue обычно применяются в условных операторах. В частности, инструкция break чаще всего используется в конструкции switch. В следующих примерах показано различие в использовании операторов break и continue.

Приведем пример использования оператора break:

```
$counter = 0;
while ( $counter <5 )
```

```
{
      $counter++;
      if ( $counter == 3 )
      {
            echo "break\n";
            break;
      }
      echo "Последняя строка цикла: счетчик=$counter\n";
}
echo "Первая строка после цикла\n\n";
```

Получим следующий результат:

```
Последняя строка цикла: счетчик=1
Последняя строка цикла: счетчик=2
break
Первая строка после цикла
```

Следует заметить, что цикл останавливается сразу по достижении строки, содержащей оператор `break`, и переходит к выполнению команд после цикла `while`. Оператор `continue` действует по-другому.

Ниже приведен пример использования оператора `continue`.

```
$counter = 0;
while ( $counter <5 )
{
      $counter++;
      if ($counter == 3 )
      {
            echo "continue\n";
            continue;
      }
      echo "Последняя строка цикла: счетчик=$counter\n";
}
echo "Первая строка после цикла\n";
```

Получим следующий результат.

```
Последняя строка цикла: счетчик=1
Последняя строка цикла: счетчик=2
continue
Последняя строка цикла: счетчик=4
Последняя строка цикла: счетчик=5
Первая строка после цикла
```

В отличие от примера с использованием оператора `break` при достижении оператора `continue` цикл не завершается. Просто выполнение третьей итерации прерывается и происходит переход к следующей. Цикл завершается тогда, когда условие становится ложным (т.е. когда значение переменной `$counter` станет больше 5).

Оператор `break` также часто используется для предотвращения бесконечных циклов. Следующее выражение внутри цикла сможет предотвратить неограниченное количество повторений:

```
$test4infinity++;
if ($test4infinity >100 )
{
      break;
}
```

Если известно, что цикл не должен повторяться больше 100 раз, приведенный фрагмент кода поможет избежать создания бесконечного цикла.

Глава 8

Повторное использование кода в сценариях PHP

В этой главе...

- Включение файлов в сценарии
- Вопросы безопасности, связанные с включением файлов
- Создание функций
- Использование функций

Зачастую одни и те же действия необходимо выполнить в разных частях сценария. Например, нужно несколько раз получить информацию из базы данных. Или можно представить себе ситуацию, когда один и тот же код используется в разных файлах. Если окажется, что время от времени вы набираете одни и те же десять строк кода (постоянно выделяя и копируя их), то лучше поместить их в отдельный файл и использовать, когда того потребует ситуация. Можно выделить следующие преимущества такого подхода.

- ✓ **Меньше работы** — это всегда стимулирует.
- ✓ **Отладка осуществляется один раз.** Можно написать код один раз, проверить его работоспособность, а затем использовать столько, сколько потребуется. Редко удается сразу переписать код без каких-либо опечаток, не говоря уже о том, чтобы написать новый. Его всегда нужно проверять. Поэтому использование готового проверенного кода позволяет сэкономить много времени
- ✓ **Простота понимания**. Чем меньше кода, тем более простым и читаемым является сценарий. Например, одну строку кода, содержащую функцию `getData()`, проще понять, чем, скажем, десять срок, которые реально используются для получения данных.
- ✓ **Простота поддержки**. Если необходимо внести какие-то изменения в сценарий, то при повторном использовании кода их нужно будет внести всего один раз во внешнем файле. В противном случае пришлось бы искать нужный фрагмент кода везде, где он используется, и изменять его соответствующим образом. Например, если поменяется имя базы данных, то куда практичнее будет внести изменения один раз в одном файле, чем во всех частях, где используется ее имя.

В языке PHP существуют два варианта повторного использования кода: с помощью включения файла в сценарий или путем использования функций. В данной главе будут рассмотрены оба этих подхода.

Включение кода в сценарий

Можно создать отдельный файл, состоящий из нужного количества строк кода, и включить его в исходный сценарий там, где это нужно. Для этого в PHP предназначена функция `include`.

Включение файлов

Предположим, что вы создаете интерактивный каталог товаров и что ваше приложение содержит большое количество страниц, отражающих его содержимое. Размеры изображений (ширина и высота) можно задать как константы, а затем использовать их в дескрипторах HTML для вывода изображений. Таким образом, все картинки будут одинакового размера. Впоследствии можно изменять размеры изображений путем простой замены значений констант. При этом отпадает необходимость вносить изменения в каждый дескриптор `<img>` сценария. Создать константы можно следующим образом:

```
define("HEIGHT", 60);
define("WIDTH", 60);
```

Затем их можно использовать в HTML-дескрипторах:

```
<img src="mypic.jpg" height="<?php echo HEIGHT?>"
width="<?php echo WIDTH?>" />;
```

Если изображения товаров выводятся в разных файлах сценария, нет необходимости в каждом из них размещать определения констант. Константы можно объявить в одном файле, например `size.inc` (это расширение обычно используется для файлов включения), а затем включать его в сценарий там, где это будет нужно:

```
<?php
     define("HEIGHT", 60);
     define("WIDTH", 60);
?>
```

Теперь для включения файла следует воспользоваться выражением

```
include("size.inc");
```

При обработке этого фрагмента кода интерпретатор PHP считывает команды данного файла и вставляет его в исходный файл вместо инструкции `include`, т.е. HTML-дескрипторы, содержащие константы `HEIGHT` и `WIDTH`, будут отображаться следующим образом:

```
<img src="mypic.jpg" height="60" width="60" />
```

Таким образом, на Web-странице будет выведена картинка. Если необходимо автоматически изменить размеры всех изображений, следует один раз поменять значения констант `HEIGHT` и `WIDTH`. Поскольку дескриптор `<img>` является достаточно сложным и содержит большой набор параметров, его можно поместить в отдельный файл, скажем `displayPix.inc`, и включать при выводе изображений. При этом сам `<img>`-дескриптор можно поместить в файл `displayPix.inc`, константы — в файл `size.inc`, а затем оба файла включить в результирующий сценарий либо включить файл `size.inc` в `displayPix.inc`, а затем использовать только `displayPix.inc`.

Зачастую разработчики забывают использовать дескрипторы PHP. Это одна из наиболее частых ошибок, которая связана с вопросом безопасности, поскольку без PHP-дескрипторов содержимое страницы будет без обработки выведено на экран. Безусловно, если пользователь знает размеры изображений, это не создаст проблем. Но, если в файле будет содержаться, например, пароль для доступа к базе данных, это может иметь нежелательные последствия.

Вместо функции `include()` можно использовать выражение

```
include_once("имя_файла");
```

Эта функция позволяет избежать повторного переопределения одинаковых переменных, содержащихся во включаемых файлах. Например, функцию `include_once()` можно использовать для включения определения функций (этот вопрос рассматривается далее в этой главе) с тем, чтобы они не повторялись.

В языке PHP имеются также функции `require()` и `require_once()`, которые подобны `include()`, но отличаются способом обработки ошибок. Это различие проявляется, например, при обращении к файлу, которого не существует. В этом случае функция `require()` выдаст ошибку и сценарий остановится. При использовании `include()` будет выведено только сообщение об ошибке, но сценарий продолжит выполняться.

В качестве имен файлов также можно использовать значения переменных:

```
include("$filename");
```

Например, необходимо вывести различные сообщения в зависимости от дня недели. При этом сообщения находятся в файлах, имена которых соответствуют различным дням. Например, файл с именем `Sun.inc` может содержать следующее:

```
echo "Думайте о будущем. Отсыпайтесь. Сегодня - никакой работы.";
```

И другие файлы в том же духе. Следующий фрагмент кода позволяет корректно вывести нужное сообщение в зависимости от дня недели:

```
$today = date("D");
include("$today"."inc");
```

В результате выполнения функции `date()` (см. главу 5) переменной `$today` будет присвоено сокращенное название дня недели. Вторая строка позволяет подключить необходимый файл. Например, если `$today` равно `Sun` (воскресенье), то будет выведено сообщение, содержащееся в файле `Sun.inc`.

Размещение файлов включения

Размещение файлов включения при разработке Web-страниц является важным вопросом, особенно с точки зрения обеспечения безопасности. Если файл никоим образом не защищен, его сможет загрузить любой пользователь. Теоретически это можно сделать, используя следующий адрес:

```
http://yourdomain.com/secretpasswords.inc
```

Предположим, что в Web-окружении имеется файл `secretpassword.inc`, который содержит следующую информацию:

```
<?php
      $mysecretaccount="account48756";
      $mypassword="secret";
?>
```

В большинстве случаев Web-серверы не настроены на обработку PHP-кода в файлах с расширениями, отличными от `.php`, т.е. интерпретатор PHP не обрабатывает файл `secretpassword.inc`, и его содержимое будет отображено пользователю как HTML-код, что явно не желательно. Для защиты от подобных случаев можно предпринять одно из следующих действий.

- ✓ **Переименовать файл включения с расширением `.php`.** В таком случае Web-сервер будет обрабатывать файл включения так же, как и обычный сценарий PHP. Но тогда следует хорошенько подумать о его содержимом. Дело

в том, что независимая обработка отдельных файлов включения крайне нежелательна. Предположим, например, что файл включения предназначен для удаления записи в базе данных (что, правда, маловероятно). Тогда независимая обработка такого файла (отдельно от файла сценария) может иметь негативные последствия. Еще одним фактором является то, что расширение файла `.inc` явно указывает, что он вспомогательный и предназначен для включения, а не для отдельной обработки.

- ✓ **Настроить Web-сервер на обработку кода PHP не только в файлах с расширением `.php`, но и в других файлах, например с расширением `.inc`.** Этот подход позволит отличить файлы по их функциональному назначению. Однако все равно остается проблема, связанная с независимой обработкой файлов включения.
- ✓ **Разместить файл в месте, недоступном извне.** Это решение является наиболее предпочтительным. Хотя оно не всегда реализуемо, например при использовании услуга Web-хостинговой компании.

Наилучшим вариантом является размещение файлов в каталогах, недоступных извне. Другими словами, необходимо создать каталоги для файлов включения, к которым невозможно будет получить доступ из Web-окружения. Например, Web-сервер Apache по умолчанию сохраняет файлы для Web в каталоге `apache/htdocs` (хотя этот путь можно изменить в конфигурационном файле `httpd.conf`). Поэтому для защиты файлов включения от внешнего доступа их можно разместить, например, в каталоге `d:\include`.

Установка путей для файлов включения

Модуль PHP позволяет задать специальные пути, которые будут использоваться для загрузки файлов включения. Если вы являетесь администратором этого модуля, это можно очень просто сделать в файле конфигурации `php.ini` (этот файл размещается в системном каталоге, как описано в приложении А). Для этого необходимо найти строку, содержащую директиву `include_path`, и изменить указанное в ней значение. Если в начале этой строки находится точка с запятой, ее необходимо удалить. Рассмотрим следующие примеры:

```
include_path=".;d:\include";            # для Windows
include_path=".:/user/local/include";   # для Unix/Linux/Mac
```

Обе эти строки определяют каталоги, в которых интерпретатор PHP будет искать файлы включения. Первое значение (символ точки) соответствует текущему каталогу, за которым следует имя второго каталога. PHP позволяет определить столько каталогов, сколько требуется. При этом порядок проверки каталогов на наличие файлов включения соответствует порядку значений, указанному в директиве `include_path`. Разделителем этих значений для операционных систем Windows является точка с запятой и двоеточие — для Unix/Linux.

Если же вы не имеете доступа к конфигурационному файлу `php.ini`, этот путь можно установить отдельно для каждого сценария следующим образом:

```
ini_set("include_path", "d:\hidden");
```

Это выражение присваивает переменной `include_path` определенное значение только на время выполнения сценария. Оно не устанавливает его для всего Web-окружения.

Для доступа к файлу включения по установленному пути необходимо использовать его имя. При этом полный путь указывать не обязательно:

```
include("secretpasswords.inc");
```

Если необходимый файл не содержится по адресу, указанному в директиве include_path или в том же каталоге, что и сам сценарий (например, в подкаталоге или каталоге, скрытом от Web-окружения), следует указать полный путь:

```
include("d:/hidden/secretpasswords.inc");
```

В остальных случаях будет достаточно одного имени файла.

Создание повторно используемого кода (функции)

Приложения обычно выполняют одни и те же действия в разных частях сценария. И здесь приходят на помощь функции. *Функция* (function) представляет собой набор инструкций, предназначенных для выполнения определенных действий.

Предположим, например, что к каждой Web-странице необходимо добавить следующую информацию:

```
echo '<img src="greenrule.jpg" width="100%" height="7" />
<address>Моя компания
<br />ул. Прекрасная 1234
<br />Сан Диего, CA 92126
</address></font>
<p>или обращайтесь в
     <a href="mailto:sales@company.com">отдел продаж </a>
<img src="greenrule.jpg" width="100%" height="7" />';
```

Обычно в начало или в конец Web-страницы добавляют куда больше информации. Поэтому гораздо эффективнее создать функцию, например с именем add_footer(), чем каждый раз писать один и тот же код в конце страниц. Затем нужно добавить в код сценария имя функции (т.е. *вызвать* ее), которая выведет нужную информацию:

```
add_footer()
```

Обратите внимание на наличие скобок после имени функции. Они являются обязательными и указывают интерпретатору PHP на то, что выполняется вызов функции.

Определение функций

Для создания функции необходимо наполнить тело функции (т.е. определить набор команд, которые она должна выполнять). Общий синтаксис функции имеет вид

```
function имя_функции()
{
   набор команд;
   return;
}
```

Рассмотрим, например, функцию, которая добавляет специальную информацию в конец Web-страницы (она уже упоминалась в предыдущем разделе):

```
function add_footer()
{
   echo '<img src="greenrule.jpg" width="100%" height="7" />
   <address>Моя компания
   <br />ул. Прекрасная 1234
```

```
    <br />Сан Диего, CA 92126
    </address></font>
    <p>или обращайтесь в
    <a href="mailto:sales@company.com">отдел продаж </a>
    <img src="greenrule.jpg" width="100%" height="7" />';

    return;
}
```

Оператор `return` останавливает выполнение тела функции и передает управление основному коду сценария. (Инструкция `return` не является обязательной в конце тела функции, но ее использование позволяет легче объяснить принципы ее работы. Более подробно оператор `return` рассматривается ниже.)

Функции можно определять в любом месте сценария, но обычно их помещают либо в начало, либо в конец сценария. Функции, которые предполагается использовать в нескольких сценариях, следует размещать в отдельном файле и включать его там, где это необходимо.

Наверное, к этому моменту вы задались следующим вопросом: "Почему бы мне просто не поместить информацию, которую необходимо выводить в конце каждой Web-страницы, в отдельный файл, скажем `footer.inc`, и включать его в сценарии?" Хороший вопрос! В принципе так можно поступить. И в данном случае следовало бы. Дело в том, что выражения, используемые для создания необходимой информации, представляют собой набор простых команд, которые выводят HTML-код. Его можно поместить в отдельный файл и включать в конце каждой страницы. Для этого даже не понадобится использовать дескрипторы PHP. Однако, предположим, что компания, для которой создается Web-узел, состоит из трех подразделений. И в информации, которая выводится внизу каждой страницы, необходимо также указать название подразделения и адрес электронной почты для переписки. Для этого можно было бы создать три различных файла включения и каждый раз использовать требуемый. Однако в данном случае решение на основе функций представляется более гибким и быстрым. Функции можно указать (т.е. *передать ей значение* (passing value)), информацию о каком подразделении требуется вывести. Например:

```
function add_footer($division)
{
    echo '<img src="greenrule.jpg" width="100%" height="7" />
    <p>'.$division. '</p>
    <address>Моя компания
    <br />ул. Прекрасная 1234
    <br />Сан Диего, CA 92126
    </address></font>
    <p>или обращайтесь в
    <a href="mailto:'.$division.'@company.com">' .$division.'</a>
    <img src="greenrule.jpg" width="100%" height="7" />';
    return;
}
```

В этом случае функция считывает значение параметра, которое содержится в параметре `$division`. Далее оно используется в теле функции для вывода нужной информации. При использовании функции `add_footer()` в качестве входного аргумента ей необходимо передать некоторое значение:

```
add_footer("Отдел продаж");
```

Для изменения информации о подразделении необходимо передать другой входной параметр:

```
add_footer("Бухгалтерия");
```

Обратите внимание на синтаксис функции echo. Строка заключена в одинарные кавычки. В предыдущем примере, без переменных, одинарные кавычки использовались только в начале и в конце выводимой строки. В данном случае, при использовании переменной, строка заканчивается одинарной кавычкой перед переменной $division и снова открывается после. Запомните, что значение переменной в строке с одинарными кавычками не выводится. Если $division поместить в такую строку, то вместо строки Отдел продаж будет выведен текст $division.

Результатом выполнения функции также может быть некое *возвращаемое значение* (returning value). Для этого используется оператор return. Предположим, что в результате выполнения функции add_footer() информацию необходимо сохранить в переменной и не выводить на экран. Это можно осуществить следующим образом:

```
function add_footer($division)
{
   $str='<img src="greenrule.jpg" width="100%" height="7" />
   <p>'.$division. '</p>
   <address>Моя компания
   <br />ул. Прекрасная 1234
   <br />Сан Диего, CA 92126
   </address></font>
   <p>или обращайтесь в
   <a href="mailto:'.$division.'@company.com">' .$division.'</a>
   <img src="greenrule.jpg" width="100%" height="7" />';
   return $str;
}
```

И воспользоваться таким выражением:

```
$footer = add_footer("Отдел продаж");
echo $footer;
```

При выводе переменной $footer отобразится вся информация, сгенерированная в функции add_footer().

Далее в этой главе подробно рассматриваются вопросы, связанные с созданием и использованием функций. Они позволяют быстро разработать выполняемый сценарий, который легко отлаживать и поддерживать. Именно поэтому высококлассные программисты всегда стараются использовать функции.

Использование переменных в функциях

В функциях можно создавать и использовать переменные, которые называются *локальными* (local). Эти переменные могут быть доступны только в теле функции. (Для того чтобы переменная была доступна извне, ее необходимо объявить *глобальной* (global), используя ключевое слово global.) Рассмотрим пример создания локальной переменной внутри функции:

```
function format_name($first_name, $last_name)
{
   $name = $last_name. ", ".$first_name;
}
```

Функцию format_name() можно вызвать, передав ей соответствующие значения, и попытаться вывести переменную $name:

```
format_name("Джесс", "Джонс");
echo "$name";
```

Однако в результате ничего выведено не будет, поскольку в переменной $name не содержится никакого значения. Эта переменная была создана внутри функции и не доступна из основного кода сценария.

Для того чтобы изменить эту ситуацию, необходимо объявить переменную глобальной. Рассмотрим измененный вариант предыдущего сценария:

```
function format_name($first_name, $last_name)
{
   global $name;
   $name = $last_name. ",".$first_name;
}
```

Теперь переменную $name можно использовать вне тела функции и вывести ее значение:

```
format_name("Джесс", "Джонс");
echo "$name";
```

В результате отобразится следующая информация:

```
Джонс, Джесс
```

Переменную обязательно необходимо объявить глобальной перед тем, как выполнять с ней какие-либо действия. Например, если бы ключевое слово `global` следовало после присваивания значения переменной $name, на экране ничего не отобразилось бы, т.е. функция и сценарий работали бы не корректно.

Аналогичным образом, если переменная объявлена вне тела функции, а затем переменная с тем же именем объявлена внутри функции, то внешняя переменная будет доступна внутри функции только в том случае, если в функции она объявлена глобальной. Для примера рассмотрим следующий фрагмент кода:

```
$first_name = "Джесс";
$last_name = "Джонс";
function format_name()
{
   global $first_name, $last_name;
   $name = $last_name. ", ".$first_name;
   echo "$name";
}
format_name();
```

Если при объявлении переменных $last_name и $first_name в теле функции не использовать ключевое слово `global`, созданные локальные переменные будут отличаться от глобальных. В таком случае они не будут иметь значений, и в результате вывода переменной $name получим запятую (,).

Для корректной работы функции `format_name()` необходимо использовать ключевое слово `global`.

Передача значений в функцию

Для передачи значений в функцию их необходимо заключить в скобки после имени функции:

```
имя_функции(значение1, значение2, ...);
```

Безусловно, переменные просто так ниоткуда не возьмутся, и их нужно предварительно задать. Это делается при определении функции с помощью ключевого слова `function`:

```
function имя_функции(значение1, значение2, ...)
{
     набор команд;
     return;
}
```

Передача значений корректного типа

При передаче значений в качестве аргументов можно использовать переменные любого типа, включая массивы или объекты (объекты рассматриваются в главе 9).

Следующее выражение вызывает функцию `salestax()`, которая вычисляет налог на продажу. Для этого необходимо знать сумму покупки и название штата (поскольку в разных штатах установлены разные ставки), чтобы корректно определить сумму налога, т.е. функции `salestax()` нужно передать два аргумента: числового (сумма покупки) и строкового (название штата) типов. Рассмотрим некоторые примеры.

- ✓ `compute_salestax(2000, "CA");` Этой функции переданы два значения: 2000 и CA (штат Калифорния).
- ✓ `compute_salestax(2*1000, "");` Этой функции переданы два значения: 2000 и "" (пустая строка). Функция должна уметь обрабатывать различные варианты входных данных, в том числе и пустую строку.
- ✓ `compute_salestax(2000, "C"."A");` Этой функции передано два значения: 2000 и "C"."A", что соответствует CA.

В качестве аргументов функции также можно передавать массивы. (Подробно о массивах см. главу 6.) Рассмотрим следующий пример:

```
function add_numbers($numbers)
{
   for($i=0;$i <sizeof($numbers);$i++)
   {
      @$sum = $sum + $numbers[$i];
   }
   return $sum;
}
```

Эта функция позволяет вычислить сумму элементов массива. Если же переданная переменная не является массивом, интерпретатор PHP преобразует ее к нужному типу: числу или строке. Однако при выполнении функции `sizeof($numbers)` произойдет ошибка, поскольку в качестве входного параметра она использует массив. Поэтому хорошо написанная функция должна проверять переданную ей информацию. Например, в предыдущее выражение перед циклом `for` можно добавить следующее:

```
if(!is_array($numbers)
{
   echo "Переданная переменная не является массивом";
   exit();
}
```

Точно так же эта функция должна проверить, что элементы массива являются числами, используя функции, описанные в главе 5.

Рассмотрим пример использования созданной функции `add_number()`:

```
$arrayofnumbers = array(100,200);
$total = add_numbers($arrayofnumbers);
```

В результате переменная `$total` станет равной 300.

Передача значений в корректном порядке

Функция получает значения в том порядке, в котором они ей передаются. Рассмотрим следующий пример:

```
function functionx($x, $y, $z)
{
   выполнение некоторых действий
}
```

Осуществим вызов этой функции:

```
functionx($var1, $var2, $var3);
```

При этом происходит следующее: переменной `$x` присваивается переданное значение `$var1`; переменной `$y` — значение `$var2` и переменной `$z` — `$var3`.

Если при вызове функции порядок переданных аргументов не соответствует тому, который использовался при определении, — возникает ошибка. Рассмотрим, например, определение функции, которая вычисляет налог на продажу:

```
function compute_salestax($orderCost, $custState)
{
   вычисление налога
}
```

Предположим, осуществляется ее вызов:

```
compute_salestax($custState, $orderCost);
```

В этом случае переданное название штата используется в качестве суммы покупки, и интерпретатор PHP присваивает этой переменной нулевое значение (преобразует строку в число). То же происходит и с переменной, соответствующей сумме покупки. В результате будет получено значение 0.

Передача корректного количества переменных

При вызове функции ей необходимо передать определенное количество переменных. Если одна из них пропущена, ей присваивается значение NULL. Если интерпретатор PHP настроен на обработку ошибок, то будет выведено соответствующее сообщение. (Об уровнях ошибок см. в главе 4.) Рассмотрим следующий пример вывода сообщения об ошибке:

```
Warning: Missing argument 2 for format_name() in testing.php on line 9
(Предупреждение: Отсутствует аргумент 2 функции format_name() в файле
testing.php в строке 9)
```

Помните, что *предупреждения* (warning) не останавливают выполнение сценария. Предположим теперь, что осуществляется вызов функции `format_name()`, рассмотренной в разделе "**Использование переменных в функциях**" выше в этой главе:

```
format_name("Джесс");
```

В результате получим:

```
Джесс,
```

Если функции передать слишком много аргументов, лишние из них будут проигнорированы. Однако такая ситуация возникает редко.

Переменным, значения которых передаются в функцию, можно присвоить значения по умолчанию (т.е. если аргумент не передан, используется значение по умолчанию). Они присваиваются при определении функции:

```
function add_2_numbers($num1=1, $num2=1)
{
   $total = $num1 + $num2;
   return $total;
}
```

Однако если при вызове функции указано значение параметра, то именно оно используется вместо значения по умолчанию. Рассмотрим различные примеры вызова функции `add_2_number()`:

```
add_2_numbers(2, 2);
add_2_numbers(2);
add_2_numbers();
```

Будут получены следующие результаты:

```
$total = 4
$total = 3
$total = 2
```

Первый результат (`$total=4`) получен путем сложения двух переданных значений, равных 2. Второй (`$total=3`) — путем сложения числа 2 и значения по умолчанию 1, используемого для переменной `$num2`. Третий (`$total=2`) — путем сложения двух значений, по умолчанию равных 1, используемых для `$num1` и `$num2`.

Передача значений по ссылке

Если при определении функции используется передача значений, то такой подход называется передачей аргументов по значению. Такой подход применяется чаще всего, как, например, в следующем выражении:

```
function add_1($num1)
{
     $num1 = $num1 + 1;
}
```

При передаче аргументов по значению переданное значение копируется в локальную переменную, которая и используется в теле функции. В данном случае таковой является `$num1`, к которой прибавляется 1. Однако значение переменной вне функции (т.е. переменной, которая используется при вызове) не изменяется. Рассмотрим следующий пример:

```
$orig_num = 3;
add_1($orig_num);
echo $orig_num;
```

В результате вывода получим 3. При вызове функции `add_1()` значение переменной `$orig_num` будет скопировано в переменную `$num1`. При этом никакие действия над `$orig_num` осуществляться не будут. Для того чтобы изменить ее значение, необходимо добавить в определение функции следующее выражение:

```
return $num1;
```

Переменной `$orig_num` можно присвоить результат выполнения функции `add_1()`:

```
$orig_num = 3;
$orig_num = add_1($orig_num);
echo $orig_num;
```

Теперь результатом вывода будет 4.

Однако в некоторых случаях необходимо изменить значение переменной напрямую. Предположим, в предыдущем примере к переменной `$orig_name` нужно прибавить 1 без создания локальной копии. В этом простом примере, безусловно, можно воспользоваться глобальной переменной, но это также можно осуществить, используя *передачу по ссылке* (passing by reference). Для этого при определении функции необходимо добавить знак амперсанта (&) перед именем переменной:

```
function add_1(&$num1)
{
     $num1 = $num1 + 1;
}
```

При вызове этой функции в качестве аргумента передается не значение, а адрес, по которому хранится переменная (т.е. указатель контейнера с именем `$orig_num`, где содержится значение 3). Переменная `$num1` становится псевдонимом `$orig_num`, так как если переменной `$num1` присвоить некоторое значение, оно же будет присвоено и `$orig_num`. Эти переменные указывают на один и тот же адрес памяти, где хранится значение. Если изменить значение локальной переменной в теле функции, эти же изменения произойдут и с внешней (по отношению к функции) переменной. Рассмотрим следующий пример вызова функции:

```
$orig_num = 3;
add_1(&$orig_num);
echo $orig_num;
```

В результате вывода получим 4.

Следующее выражение, использующее передачу аргумента по ссылке, не имеет смысла:

```
add_1(&7);
```

Передача по ссылке главным образом используется для передачи больших сущностей, таких как объекты или массивы, поскольку пи таком подходе требуется гораздо меньше памяти.

Возвращаемое значение функции

Для возвращения значения функции в языке PHP используется оператор `return`. Это значение затем может обрабатываться различными способами.

Оператор `return` возвращает некоторое значение и выполняет переход к основному коду сценария. Общий синтаксис этого оператора имеет следующий вид:

```
return значение;
```

Рассмотрим такой пример:

```
function add_2_numbers($num1, $num2)
{
   $total = $num1 + $num2;
   return $total;
}
```

Эта функция возвращает сумму двух значений. Рассмотрим пример ее вызова:

```
$sum = add_2_numbers(5, 6);
```

В этом примере значение локально созданной переменной `$total`, равное 11, будет присвоено переменной `$sum`. Фактически при определении функции можно использовать сокращенную запись, например:

```
return $num1 + $num2;
```

При этом возвращаемое значение функции можно использовать в различных выражениях. Рассмотрим некоторые примеры:

```
$total_height = add_2_numbers($height1, $height2);
$totalSize = $current_size + add_2_numbers($size1, $size2);
if (add_2_numbers($costSocks, $costShoes) > 200.00 )
        $echo "Дорого";
```

При этом следует заметить, что оператор `return` позволяет возвращать только одно значение, которым, однако, может быть и массив. Поэтому, в принципе, функция может возвращать несколько значений.

Выражение `return` также можно использовать в условных выражениях:

```
function find_value($array, $value)
{
  for($i=1;$i<sizeof($array);$i++)
  {
     if($array[$i] = $value)
     {
       echo "$i. $array[$i]<br>";
       return;
     }
  }
}
```

Функция `find_value()` проверяет, содержит ли массив `$array` значение `$value`. Рассмотрим пример вызова этой функции:

```
$names = array("Джо", "Сэм", "Хуан");
find_value($names, "Сэм");
```

Функция ищет среди элементов массива значение Сэм. Если соответствующий элемент найден, выводится его индекс и функция завершает свою работу:

```
1. Сэм
```

Очень часто функции возвращают булевы значения, как, например, в выражении

```
function is_over_100($number)
{
  if($number > 100)
  {
     return TRUE;
  }
  else {
     return FALSE;
  }
}
```

Если значение переменной $number меньше или равно 100, возвращается значение FALSE, в противном случае — TRUE.

Рассмотрим еще один пример создания функции, в котором она возвращает некоторое значение в случае успешного выполнения и значение FALSE в противном случае:

```
function find_value($array, $value)
{
  for($i=1;$i<sizeof($array);$i++)
  {
     if($array[$i] == $value)
     {
       return $i;
     }
  }
  return FALSE;
}
```

Если нужное значение найдено в массиве, будет возвращен индекс элемента, в противном случае — значение FALSE.

Использование встроенных функций

В языке PHP имеется большой набор мощных и полезных встроенных функций, что является одной из причин его чрезвычайной популярности. Они используются наряду со стандартными функциями и ничем не отличаются от пользовательских. В них воплощена работа, которую разработчики PHP сделали за вас.

Специализированные встроенные функции описываются в соответствующих главах, в которых рассказывается, чем они могут быть полезны многим разработчикам. В данном разделе они бы выпадали из контекста. Например, в главе 7 рассматривались функции, которые используются для проверки существования переменной или ее значения. Они имеют следующий вид:

```
isset($varname)
empty($varname)
```

В главе 5 описывались функции, предназначенные для форматирования и обработки числовых и строковых выражений. И так на протяжении всей книги.

В приложении Б приведен обширный список полезных встроенных функций. При написании сценариев лучше иметь этот список при себе, чтобы быстро найти нужную функцию. При этом можно также создавать собственные функции, которые будут работать лучше, чем встроенные. Безусловно, в приложении Б представлены не все функции — их тысячи. Однако там приведены наиболее полезные их них. Весь список встроенных функций можно найти в документации PHP на официальном Web-узле (www.php.net/docs.php).

Обработка ошибок

Как это ни печально, иногда функции работают некорректно. Как бы вы не старались учесть все возможные варианты ошибок, что-то все-таки может не сработать. Например, функция подключения к базе данных может функционировать неправильно, если база данных в данный момент недоступна. Причем это не связано с тем, что в функции содержатся ошибки, — так сложилась ситуация. Хорошо написанная функция должна предвидеть все возможные ситуации и возвращать в таких случаях значение FALSE.

Сценарий должен предвидеть любые ошибки, связанные с работой функций, и соответствующим образом обрабатывать их. Лучше вывести пользователю собственноручно созданное сообщение, чем стандартное предупреждение, сгенерированное PHP. Для этого в PHP предусмотрена функция `die()`. Ее общий синтаксис имеет следующий вид:

```
die("сообщение");
```

Функция `die()` позволяет прервать выполнение сценария и вывести соответствующее сообщение. С функциями ее необходимо использовать следующим образом:

```
имя_функции()    or    die("сообщение");
```

Если в результате выполнения функции будет получено значение FALSE, функция `die()` прервет выполнение сценария и выведет сообщение.

Рассмотрим пример, в котором выполняется подключение к базе данных MySQL:

```
mysql_connect("host", "user", "password")
      or die("База данных не доступна. Попробуйте подключиться позже.);
```

Необходимо помнить следующее. Если функция работает некорректно, интерпретатор PHP сгенерирует сообщение об ошибке. Если вместо него необходимо вывести свое сообщение, следует изменить уровень вывода ошибок или не выводить их вообще (см. главу 4). Иначе будут выведены оба сообщения.

Функцию `die()` можно использовать с любой другой функцией. Однако ее не имеет смысла применять в тех случаях, когда функция возвращает значение FALSE. Помните, что `die()` полностью останавливает выполнение сценария.

Предотвратить некорректную работу функции можно, используя ее вызов в условном выражении. К результату, полученному в предыдущем примере, можно прийти с помощью следующего выражения:

```
if(!mysql_connect("host", "user", "password"))
{
    echo "База данных недоступна. Попробуйте подключиться позже\n";
    exit();
}
```

Обратите внимание на восклицательный знак перед вызовом функции `mysql_connect()`. Он используется для следующего: если не удается подключиться к базе данных (т.е. будет возвращено значение FALSE), условное выражение станет истинным (TRUE) и будет выведено соответствующее сообщение.

При этом функция `exit` не выполняет те же действия, что и `die`. Помните, что любое из этих выражений можно использовать в операторе `if`. Можно даже написать сценарий, который в случае ошибки при подключении к базе данных будет отправлять письмо по электронной почте.

Глава 9

Объектно-ориентированное программирование на PHP

В этой главе...

- Что такое объектно-ориентированное программирование
- Идентификация объектов
- Создание классов
- Использование классов

Самого начала PHP рассматривался как простой язык написания сценариев. Позже, начиная с версии 4, в нем появились объектно-ориентированные свойства. После появления версии 5 возможности объектно-ориентированного программирования на языке PHP были существенно расширены, а кроме того, повысилось быстродействие разрабатываемых программ. Однако большинство из новых свойств остаются невидимыми. Эти изменения в основном касаются ядра Zend 2, которое в PHP 5 стало гораздо эффективнее по сравнению с ядром PHP 4. Кроме повышения скорости обработки сценариев, в PHP 5 были добавлены новые возможности, которые сделали этот язык программирования по-настоящему объектно-ориентированным.

Введение в объектно-ориентированное программирование

Объектно-ориентированный подход к программированию предполагает использование объектов и классов. Более подробно они рассматриваются в этой главе ниже. В настоящее время объектно-ориентированный подход широко распространен, и многие университетские курсы по программированию начинаются именно с его изучения. При объектно-ориентированном программировании чаще всего используются языки программирования Java и C++.

Объектно-ориентированное программирование — это не просто использование другого синтаксиса, а во многом и другой подход к анализу и решению задач. Объектно-ориентированная программа разрабатывается путем моделирования процессов в реальной предметной области. Например, программист, разрабатывающий программу поддержки функционирования отдела продаж некоторой компании, может рассматривать ее с точки зрения взаимодействия покупателей, продавцов и кредитных политик, т.е. в терминах, которыми оперируют специалисты самого отдела продаж.

В объектно-ориентированном программировании основными элементами программы являются *объекты* (object). Они являются представлением реальных объектов, которые задействованы при решении реальной проблемы. Например, если приложение относится к предметной области продажи подержанных автомобилей, то, возможно, в качестве объектов будут выбраны автомобили и покупатели. Если же решаемая задача связана с космосом, то объекты будут представлять звезды и планеты.

Объектно-ориентированное программирование является новой концепцией, которая предоставляет новую терминологию для описания рассматриваемых проблем. Понимание этой терминологии — ключ к пониманию самого объектно-ориентированного подхода.

Объекты и классы

Основными элементами объектно-ориентированных программ являются *объекты* (object). Лучше всего их представлять как физические объекты из реального мира. Например, автомобиль является объектом с набором свойств (или атрибутов), таких как цвет, название модели, двигатель и тип шин. Кроме того, автомобиль может также выполнять некоторые действия, например перемещаться вперед и назад, парковаться, поворачивать и останавливаться.

В общем случае объекты соответствуют именам существительным. Человек является объектом, так же как и животные, дома, офисы, покупатели, мусорные баки, пальто, облака, планеты и кнопки. Однако объекты — это не просто физические объекты. Как и имена существительные, зачастую объекты являются более концептуальными. Например, банковский счет невозможно подержать в руках, но он тем не менее является объектом. То же самое можно сказать и про компьютерную учетную запись, залог, файл, базу данных, заказы, сообщения электронной почты, адреса, песни, телевизионные шоу, встречи, даты и т.д.

В свою очередь, *класс* (class) является шаблоном (каркасом), который можно использовать для создания объектов. Класс определяет свойства и атрибуты объекта, а также действия, которые он может выполнять. Например, рассмотрим класс, определяющий четырехколесный автомобиль с двигателем, который может перемещаться вперед и парковаться. На его основе можно создать новый объект (новый автомобиль). Однако в процессе использования этого объекта можно обнаружить отсутствие некоторых важных деталей, таких как дверь, руль или задняя передача. Это может произойти из-за того, что эти компоненты при создании класса не были учтены.

Разработчик, занимающийся созданием класса, знает все о его внутреннем устройстве. Однако для тех, кто будет использовать этот класс, подобная информация вовсе необязательна. Например, необязательно знать, как функционирует телефон, — позвонить можно и без этой информации. Специалисту, сконструировавшему телефон, известно о его внутренней архитектуре. Поэтому при появлении новых технологий он сможет вскрыть телефонный аппарат и усовершенствовать его. Вместе с тем, до тех пор, пока не изменится внешний интерфейс (клавишная панель и кнопки) телефона, на его использование ничто не повлияет.

Свойства

Объекты имеют *свойства* (property), также называемые *атрибутами* (attribute). Автомобиль может быть красным, зеленым или быть покрашенным в горошек. Такие свойства автомобиля, как цвет, размер или модель, являются внутренними характеристиками объекта. Обычно свойства класса задаются в виде переменных. Например, атрибут цвета может храниться в объекте в виде переменной с отличительным именем, например `$color`. Тогда в объекте "автомобиль" может содержаться переменная `$color` со значением `красный` (`$color = красный`).

Значения переменных, являющихся свойствами объектов, могут присваиваться по умолчанию или явно во время создания объектов. Они могут добавляться и изменяться и позже. Например, первоначально автомобиль может быть красным, а затем стать бледно-зеленым.

Методы

Действия, которые могут выполнять объекты, иногда рассматриваются как его *обязанности* (responsibility). Например, объект "автомобиль" может перемещаться вперед и назад, останавливаться и парковаться. Каждое действие (обязанность), которое объект может выполнять, реализуется в классе в виде *метода* (method).

В языке PHP метод имеет тот же синтаксис, что и функция. Разница заключается в том, что методы находятся внутри класса. Они не могут быть вызваны независимо от класса — интерпретатор PHP не позволит сделать это. Функции такого вида могут использоваться только в контексте объекта.

Как правило, при создании методов им присваивают имена, соответствующие выполняемым действиям, например `parkCar()` или `getColor()`. Методы могут иметь любые допустимые имена, однако в соответствии с общепринятыми соглашениями в их именах используются прописные символы.

Методы представляют собой интерфейс между объектом и остальным миром. Каждому действию (обязанности) объекта должен соответствовать свой метод. С объектом можно взаимодействовать лишь через его интерфейс. Например, если ваш сосед хочет одолжить у вас немного сахара, ему следует сначала постучать в вашу дверь и попросить об этой услуге. Вряд ли вы будете довольны, если сосед залезет в вашу кухню через окно и возьмет его сам. Чтобы этого не случилось, ваш дом (`house`) (как объект) должен иметь входную дверь (`front door`). И только через нее сосед (`neighbor`) будет иметь возможность попасть в дом. Другими словами, в объекте `house` должен содержаться метод `открытьВходнуюДверь` (`openFrontDoor`), который и необходимо использовать соседу. При этом другие варианты проникновения в дом должны отсутствовать. Открытие входной двери `front door` (действие объекта `house`) должно осуществляться с помощью метода `openDoor()`. При создании объектов не оставляйте в них "открытых окон".

Хорошо спроектированный объект должен содержать все необходимое для выполнения своих обязанностей, но при этом не иметь ничего лишнего. Он не должен выполнять обязанности других объектов. Например, объект "автомобиль" должен "уметь" перемещаться и иметь все компоненты, необходимые для выполнения своих обязанностей, например бензин, масло, шины, двигатель и т.д. Он не должен иметь соль и сковородку, а также "уметь" готовить пищу. В свою очередь, объект для приготовления пищи не должен обучать детей игре в футбол.

Наследование

Объекты должны содержать только необходимые свойства и методы. Одним из подходов для разделения свойств и методов между классами является *наследование* (inheritance). Пусть, например, имеются два объекта, один из которых соответствует розам белого цвета, а другой — красного. Можно создать два класса: `redRose` (красная роза) и `whiteRose` (белая роза). Однако большая часть информации будет одинаковой для обоих классов, поскольку оба вида роз относятся к кустарникам, имеют колючки и цветут в июне. Наследование позволяет избежать дублирования информации.

Сначала можно создать один класс `Rose`. Он может содержать общую информацию для роз, такую как `$plant = bush` (вид растения — кустарник), `$stem = thorns` (стебель — с колючками), `$blooms = June` (цветет в июне). Затем можно создать два подкласса для соответствующих видов роз. Тогда класс `Rose` будет являться *главным классом* (master class), или *родительским классом* (parent class). В свою очередь, классы `redRose` и `whiteRose` называются *подклассами* (subclass), или *наследниками* (child class), или *потомками* (child), как их ласково называет мой любимый профессор.

Классы-потомки наследуют все атрибуты и методы родительского класса. Однако они могут иметь и свои собственные свойства, например белый цвет (`$color = белый`) для класса `whiteRose` и красный цвет (`$color = красный`) — для класса `redRose`.

Методы, содержащиеся в классе-наследнике, могут иметь те же имена, что и методы родительского класса. В этом случае метод класса-потомка имеет больший приоритет. При необходимости можно воспользоваться любым из этих методов.

Что отсутствует в объектно-ориентированной парадигме PHP 5

Если вы знакомы с объектно-ориентированным программированием на других языках, то, возможно, обнаружите, что некоторые из его характерных свойств в языке PHP отсутствуют. Однако жизнь не стоит на месте, и многие возможности, которых не было в PHP 4, были добавлены в PHP 5. Однако по-прежнему отсутствуют следующие механизмы.

- ✓ **Полиморфизм** (polymorphism). В сценарии PHP в одном классе не может быть двух методов с одинаковыми именами, даже конструкторов. Другими словами, в PHP нельзя реализовать полиморфизм. В одном классе невозможно иметь два метода с одним и тем же именем, но с разным набором параметров (сигнатурами). Для реализации полиморфизма программисты иногда используют операторы `switch` и другие механизмы.
- ✓ **Множественное наследование** (multiple inheritance). В языке PHP отсутствует механизм множественного наследования. Класс может быть наследником только одного родительского класса.

Разработка объектно-ориентированных программ

Объектно-ориентированные программы требуют тщательного анализа и планирования гораздо больше, чем процедурные программы, в которых операторы обрабатываются от начала до конца и не используются классы. Необходимо тщательно спланировать, какие свойства и методы будут содержать объекты. При этом должны быть учтены все важные аспекты функционирования предметной области. Однако обязанности классов не должны быть раздуты. При реализации сложных проектов может понадобиться создать модель и выполнить тестирование. Лишь после этого можно быть уверенным в том, что в состав программной системы входят все необходимые объекты.

Выбор объектов

При разработке программы сначала нужно сформулировать перечень требуемых объектов. Если приложение не очень большое, то выполнить это достаточно просто. Но если программная система является большой и сложной, то список требуемых объектов может оказаться гораздо менее очевидным. Например, если разрабатывается программное обеспечение для банка, то можно выбрать такие объекты, как счет, кассир, деньги, чековая книжка, корзина для бумаги, служба охраны, сейф, система безопасности, клиент, ссуда, кредитная ставка и т.д. Однако все ли эти объекты действительно необходимы? Что программа должна делать с корзиной для бумаги в холле или со службой охраны? Возможно, понадобится составлять график ее работы.

Один из возможных подходов к идентификации объектов заключается в следующем. Сначала сформируйте перечень всех возможных объектов (т.е. имен существительных) из предметной области. Для этого разработчики иногда могут воспользоваться проектной документацией.

Затем из сформированного списка следует вычеркнуть как можно больше объектов. Необходимо исключить повторяющиеся объекты, объекты с повторяющимися обязанностями или те из них, которые напрямую не относятся к исследуемой предметной области. Например, если проект связан с конструированием автомобилей, в качестве объектов наверняка пригодятся все его детали. С другой стороны, если проект связан с управлением движением автомобилей, то их составные части окажутся гораздо менее важными.

Выбор свойств и методов для каждого объекта

После формирования списка объектов можно приступить к выбору свойств и методов для каждого из них. Спросите себя, что вам нужно знать о каждом объекте. Например, в программе, связанной с ремонтом автомобилей, пригодится информация о том, когда автомобиль последний раз проходил технический осмотр и был в ремонте, каковы характеристики его узлов и агрегатов, и т.д.

Объекты должны быть независимыми, для каждого из них необходимо определить обязанности и методы, которые они будут выполнять. В качестве примера рассмотрим объект "банковский счет". Сначала его нужно создать (т.е. открыть). Для этого подойдет метод `openBankAccount()`. После этого с банковским счетом можно осуществлять различные операции: класть деньги на счет, оплачивать счета, следить за балансом счета и выдавать о нем различную информацию.

Однако после некоторых размышлений и экспериментов может оказаться, что все-таки что-то было упущено. Например, со счетом должны быть связаны данные о владельце его имени и об адресе. Не забыли ли вы добавить метод обновления этой информации, если владелец переедет на новое место жительства?

Создание и использование класса

После выявления объектов и их свойств можно приступать к их созданию и использованию. Для создания и использования объекта нужно выполнить следующие действия.

1. **Использовать ключевое слово** `class`. Ключевое слово `class` является основным при создании класса. С ним связывается блок инструкций со свойствами и методами класса.
2. **Вставить выражение** `class` **в те места, где необходимо использовать соответствующий объект**. Класс можно разместить в самом сценарии. Однако чаще всего конструкцию `class` размещают в отдельном файле, который включается в сценарий с помощью директивы `include`.
3. **Создать объект в сценарии**. На основе описания класса можно создавать объекты, т.е. выполнять *инстанцирование* (instantiation).
4. **Использовать объект**. После создания нового объекта можно приступать к его использованию. При этом можно использовать любые методы класса, определенные в соответствующем блоке при его описании.

Оставшаяся часть этой главы будет посвящена более подробному описанию каждого из этих этапов.

Определение класса

После выявления требуемых объектов, их свойств и методов можно приступать к определению соответствующих классов. Класс является шаблоном (каркасом) для создания объектов.

Создание класса

Выражение `class` используется для определения свойств и методов класса и имеет следующий синтаксис:

```
class имяКласса
{
   Определение свойств класса
   Определение методов класса
}
```

В качестве имени класса можно использовать любой корректный идентификатор PHP, кроме `stdClass` — зарезервированного служебного имени PHP.

Описание свойств и методов класса заключается в фигурные скобки. Если необходимо, чтобы класс наследовал свойства и методы другого класса, следует воспользоваться выражением

```
class whiteRose extends Rose
{
   Определение свойств класса
   Определение методов класса
}
```

Объект, созданный на основе этого класса, будет иметь доступ ко всем свойствам и методам как класса `whiteRose`, так и `Rose`. Однако класс `Rose` не имеет доступа к свойствам и методам класса-наследника.

Следующие несколько разделов посвящены вопросам, связанным с определением свойств и методов класса. Более исчерпывающий пример создания класса приведен ниже, в разделе "Собирая все вместе".

Определение свойств

При создании класса все его свойства следует объявить в начале соответствующего блока.

```
class Car
{
   var $color;
   var $tires;
   var $gas;


   Методы
}
```

В общем случае в PHP объявлять переменные необязательно. Во многих сценариях, приведенных в этой книге, переменные не объявляются, они просто используются. То же самое касается и классов, однако лучше этого не делать. Объявление переменных позволяет легче понять назначение класса. Кроме того, отсутствие явных объявлений является плохим стилем программирования.

При объявлении переменных им можно присвоить значения по умолчанию. Такие значения могут быть лишь самыми простыми и не должны содержать вычисляемых выражений. Приведенные ниже примеры помогут лучше разобраться в этом.

- ✓ Следующие выражения можно использовать в качестве значений по умолчанию:

```
var $color = "черный";
var $gas = 10;
var $tires = 4;
```

- ✓ Эти выражения нельзя использовать в качестве значений по умолчанию:

```
var $color = "синий"." черный";
var $gas = 10-3;
var $tires = 2*2;
```

В качестве переменных можно также использовать массивы, в которых содержатся простые значения.

```
var $doors = array("передняя часть", "задняя часть");
```

Значения соответствующих переменных класса при создании объекта можно задавать или изменять, используя конструктор (который описывается ниже, в разделе "Создание конструктора") или любой другой метод, предназначенный для этих целей.

Использование переменной $this

Внутри класса специальная переменная $this представляет собой ссылку на этот же класс. При этом ее нельзя использовать вне класса. Эта переменная обеспечивает доступ к переменным и методам класса внутри самого класса.

Для использования переменной $this можно использовать следующий синтаксис:

```
$this->имя_переменной
```

Например, для доступа к атрибуту $gas класса Car следует воспользоваться выражением

```
$this->gas
```

Таким образом, с помощью переменной $this можно манипулировать атрибутом $gas внутри класса, как, например, в следующих примерах:

```
$this->gas = 20;
if($this->gas > 10)
$product[$this->size] = $price
```

Как видно в приведенных примерах, выражение $this->*имя_переменной* используется внутри класса точно так же, как и переменная $*имя_переменной* в сценарии.

Обратите внимание на то, что символ $ используется только перед именем this, а не gas. Если воспользоваться выражением $this->$gas, оно будет неправильно проинтерпретировано. При этом может возникнуть (а может, и нет) ошибка, однако в любом случае это не будет иметь никакого отношения к ссылке на переменную $gas текущего класса.

Добавление методов

Методы определяют функциональность объектов и определяются точно так же, как и обычные функции. Например, для класса "автомобиль" может понадобиться метод, который будет обеспечивать заправку бензином его бака. В классе может содержаться переменная $gas, значение которой будет соответствовать количеству бензина, содержащегося в баке в текущий момент. Можно определить метод, который будет добавлять к этой переменной значение $amount (количество заправленного бензина). Это можно осуществить следующим образом:

```
class Car
{
  var $gas = 0;
  function addGas($amount)
  {
     $this->gas = $this->gas + $amount;
     echo "в бак залито $amount галлонов бензина";
  }
}
```

Функция addGas() выглядит как обычная функция, но поскольку она находится внутри класса, то является его методом.

В языке PHP определено несколько специальных методов, имена которых начинаются с двух символов подчеркивания (__). В этой главе рассматриваются три из них: construct, destruct и clone. Поэтому имена своих методов лучше не начинать с двух символов подчеркивания, если только не переопределяется один из специальных методов.

Создание конструктора

Конструктор (constructor) — это специальный метод класса, который выполняется при создании объекта. Конструктор использовать необязательно, если при создании объекта не нужно присваивать значения атрибутам или выполнять какие-либо действия. В классе может содержаться только один конструктор.

Конструктор имеет специальное имя (__construct), поэтому интерпретатор PHP всегда знает, какой метод нужно выполнять при создании объекта. Конструктор имеет примерно следующий вид:

```
function __construct()
{
   $this->gas = 10;    # бак заполнен полностью
   $this->openDoor();
}
```

Этот конструктор создает новый автомобиль (т.е. объект) с полным баком и открытой дверью.

До версии PHP 5 имя конструктора совпадало с именем класса. При этом классы, созданные для более ранних версий интерпретатора, могут использоваться и в PHP 5. При создании объектов интерпретатор PHP 5 сначала пытается вызвать конструктор с именем __construct(). Если такой метод отсутствует, вызывается метод с именем, совпадающим с именем класса.

Собирая все вместе

Класс должен содержать столько атрибутов и методов, сколько необходимо. Эти методы могут быть как простыми, так и сложными. Однако цель объектно-ориентированного программирования заключается в том, чтобы сделать методы как можно более рациональными. Лучше использовать несколько небольших функций и вызывать один метод внутри другого, а не реализовывать все требуемые действия в одной функции. Рассмотрим следующий пример:

```
class MessageHandler
{
  var $message = "Сообщения нет";
  function __construct($message)
  {
     $this->message = $message;
  }
  function displayMessage()
  {
     echo $this->message. "\n";
  }
}
```

Класс MessageHandler имеет один атрибут $message, в котором хранится сообщение, и один метод displayMessage, предназначенный для его вывода.

Предположим, необходимо добавить метод, который будет преобразовывать символы сообщения в нижний регистр, а затем выводить их на экран. Это можно выполнить таким образом:

```
class MessageHandler
{
  var $message = "Сообщения нет";
  function __construct($message)
  {
     $this->message = $message;
  }
  function displayMessage()
  {
     echo $this->message. "\n";
  }
  function lowerCaseMessage()
  {
     $this->message = strtolower($this->message);
     $this->displayMessage();
  }
}
```

Обратите внимание на метод `lowerCaseMessage()`. Поскольку класс уже содержит метод для вывода сообщения, в новом методе `lowerCaseMessage()` используется именно этот метод. Если при создании нового метода оказалось, что часть его кода уже использовалась в другом методе, нужно выполнить их повторное проектирование. В одном и том же классе не должно быть повторяющегося кода.

В листинге 9.1 приведен более сложный пример класса, который можно использовать для создания HTML-форм. Для некоторого упрощения примера в форме содержатся только текстовые поля.

Листинг 9.1. Пример сценария, в котором содержится класс для создания форм

```
<?php
/* Имя класса: form
 * Описание:    класс, создающий простую HTML-форму с
 *              одним текстовым полем. Класс содержит 3 метода.
 */
class Form
{
  var $fields=array();  # содержит имя поля
  var $processor;       # имя программы обработки данных формы
  var $submit = "Отправить данные"; # имя кнопки Submit
  var $Nfields = 0;     # количество полей формы

/* Конструктор: передается имя сценария обработки данных
 * ($processor) и имя кнопки.
 */
  function __construct($processor, $submit)
  {
     $this->processor = $processor;
     $this->submit = $submit;
  }

/* Функция отображения формы.
 */
  function displayForm()
  {
     echo "<form action='{$this->processor}' method='post'>";
     echo "<table width='100%'>";
     for($j=1;$j<=sizeof($this->fields);$j++)
     {
        echo "<tr><td align=\"right\">
              {$this->fields[$j-1]['label']}: </td>\n";
        echo "<td>
              <input type='text'
                name='{$this->fields[$j-1]['name']}'>
              </td></tr>\n";
     }
     echo "<tr><td colspan=2 align='center'>
           <input type='submit'
               value='{$this->submit}'></td></tr>\n";
     echo "</table>";
  }

/* Функция для добавления поля формы. Ее параметрами
 * являются имя и метка поля.
 */
   function addField($name, $label)
   {
     $this->fields[$this->Nfields]['name'] = $name;
```

```
        $this->fields[$this->Nfields]['label'] = $label;
        $this->Nfields = $this->Nfields + 1;
    }
}
?>
```

В созданном классе `Form` содержатся четыре атрибута (свойства) и три метода. Ниже приведен перечень атрибутов.

- ✓ `$fields`. Массив, содержащий поля формы, заполняемые пользователем.
- ✓ `$processor`. Имя сценария, который обрабатывает данные, введенные пользователем в форме. Значение этой переменной используется в атрибуте `action` дескриптора `form`.
- ✓ `$submit`. Текст, который отображается на кнопке `submit`.
- ✓ `$Nfields`. Количество полей, добавленных в форму в текущий момент.

В классе `Form` содержатся следующие методы.

- ✓ `__construct()`. Конструктор, который во время создания объекта присваивает атрибутам `$processor` и `$submit` значения, переданные пользователем.
- ✓ `addField()`. Добавляет имя и метку поля в массив `$fields`. Например, если пользователь добавил в форму поля для ввода имени и фамилии, то этот массив будет иметь следующий вид:

```
$fields[1][name]=first_name
$fields[1][label]=First Name
$fields[2][name]=last_name
$fields[2][label]=Last Name
и т.д.
```

- ✓ `displayForm()`. Используется для вывода формы на экран, т.е. дескрипторов HTML, необходимых для создания формы. При этом используются имена и метки полей, содержащиеся в атрибутах класса.

В следующем разделе описываются принципы использования классов в сценарии, в том числе класса `Form`, приведенного в листинге 9.1.

Использование класса

Класс, который используется в сценарии, должен быть в него каким-то образом добавлен. *Обычно он размещается в отдельном файле и с помощью функции* `include()` *подключается* к основному сценарию.

Для того чтобы использовать объект, его необходимо сначала создать. После этого объект может выполнять любые методы, определенные в классе. Процесс создания объектов называется *инстанцированием* (instantiation). Аналогично тому, как выкройка используется для шитья большого количества похожей, но все же по-своему индивидуальной одежды, точно так же на основе классов создаются объекты. Для создания объекта используется следующий синтаксис:

```
$имя_объекта = new имяКласса(значение, значение, ...);
```

```
$Joe = new Person("мужчина");
$car_Joe = new Car("красный");
$car_Sam = new Car("зеленый");
$customer1 = new Customer("Смит", "Джо", $custID);
```

При создании объекта вызывается конструктор, в результате чего объект связывается с переменной с заданным именем. После этого для вызова метода класса можно воспользоваться следующими выражениями:

```
$Joe->goToWork();
$car_Joe->park("запрещено");
$car_Sam->paintCar("синий");
$name = $customer1->getName();
```

Разные объекты, созданные на основе одного и того же класса, независимы друг от друга. Например, если автомобиль Сэма перекрасить в голубой цвет, то автомобиль Джо никак не изменит свой цвет и будет оставаться красным. Джо покупает талон для парковки, но это никак не повлияет на Сэма (и на его автомобиль).

В листинге 9.2 приведен пример использования класса `Form`, который был создан в предыдущем разделе и представлен в листинге 9.1.

Листинг 9.2. Сценарий создания формы с помощью класса `Form`

```
<?php
/* Имя сценария: buildForm
 * Описание:    использование класса Form для создания
 *              простой HTML-формы.
 */
require_once("form.inc");
echo "<html><head><title>Форма для добавления телефонного
номера</title></head><body>";
$phone_form = new Form("process.php", "Отправить данные");
$phone_form->addField("first_name", "Имя");
$phone_form->addField("last_name", "Фамилия");
$phone_form->addField("phone", "Номер телефона");
echo "<h3>Пожалуйста, заполните поля данной формы:</h3>";
$phone_form->displayForm();
echo "</body></html>";
?>
```

Сначала с помощью функции `require_once()` файл `form.inc` с описанием класса `Form` включается в сценарий. На его основе создается объект с именем `$phone_form`. Затем в форму добавляются три поля, после чего форма выводится на экран. Следует заметить, что в приведенном фрагменте использовалось несколько дополнительных дескрипторов HTML, которые можно было бы разместить и в методе `displayForm()`.

При выполнении приведенного сценария с использованием класса `Form` будет создана форма с тремя полями (рис. 9.1).

Скрытые свойства и методы

Свойства и методы класса могут быть как *открытыми* (public), так и *закрытыми* (private). Открытые свойства и методы доступны извне класса, т.е. из сценария, в котором используется данный класс, или из другого класса. Рассмотрим пример класса, атрибут и метод которого являются открытыми:

```
class Car
{
  var $gas = 0;
  function addGas($amount)
```

```
    {
        $this->gas = $this->gas + $amount;
        echo "в бак залито $amount галлонов бензина";
    }
}
```

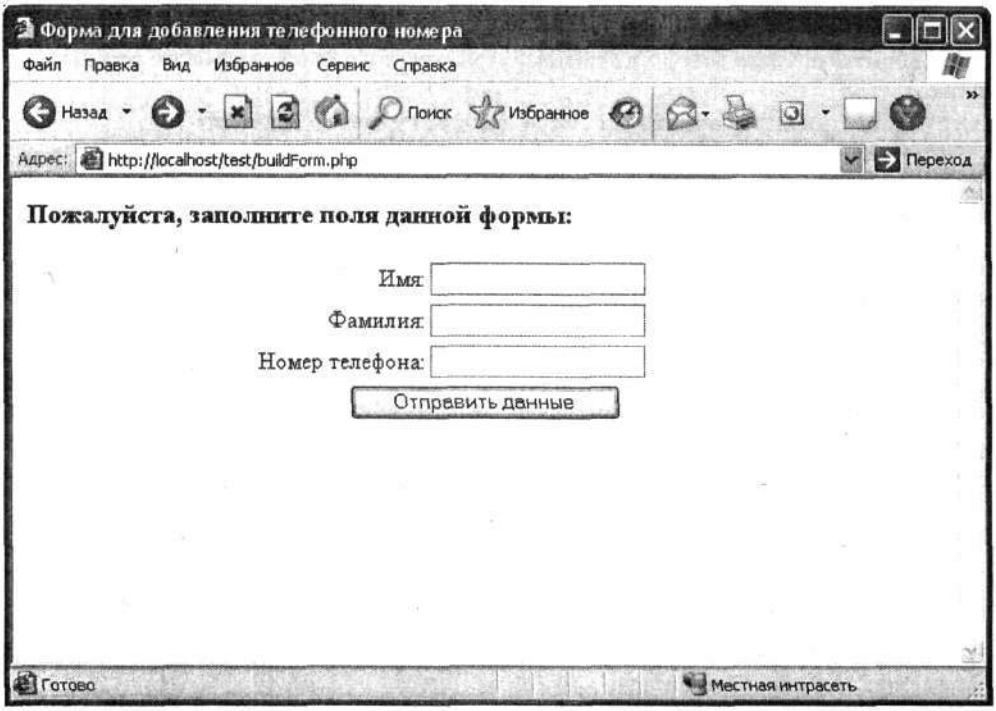


Рис. 9.1. Результат выполнения сценария, представленного в листинге 9.2

Теперь к открытому атрибуту класса `$gas` можно получить доступ из-за его пределов следующим образом:

```
$mycar = new Car;
$gas_amount = $mycar->gas;
```

После выполнения этого фрагмента в переменной `$gas_amount` будет содержаться значение атрибута `$gas` объекта `mycar`. Значение атрибута можно также и изменить извне класса:

```
$mycar->gas = 20;
```

Прямой доступ к атрибутам класса считается плохим стилем программирования. Любое взаимодействие объекта и внешнего сценария или другого класса должно осуществляться лишь посредством методов. Метод `addGas()` из приведенного класса `Car` является открытым и применяется для "заправки" автомобиля (т.е. для увеличения значения атрибута `$gas`):

```
$new_car = new Car;
$new_car->addGas(5);
```

Язык PHP позволяет ограничить доступ к атрибутам и методам класса. Это можно осуществить следующими способами.

- ✓ Спецификатор `private` (закрытый) запрещает доступ к атрибутам и методам класса из-за пределов класса, сценариев или других классов.

✓ Спецификатор protected (защищенный) предоставляет доступ к защищенным свойствам и методам класса только для его классов-наследников. В остальных случаях доступ запрещен.

Для того чтобы атрибут стал скрытым, следует воспользоваться выражением

```
private $gas = 0;
```

При попытке прямого доступа к скрытому атрибуту отобразится следующее сообщение об ошибке:

```
Fatal error: Cannot access private property car::$gas in
c:\testclass.php on line 17
(Фатальная ошибка: Нельзя получить доступ к закрытому атрибуту
car::$gas в файле c:\testclass.php в строке 17)
```

Теперь изменить значение атрибута $gas класса Car можно только с помощью метода addGas(), поскольку он является частью класса Car и, следовательно, имеет доступ к соответствующему скрытому атрибуту.

Подобно атрибутам, методы класса также могут быть скрытыми (private) или защищенными (protected). Однако в классе Car метод addGas() является открытым. В реальных программах нужно быть уверенным в том, что за бензин, которым заправлен автомобиль, уплачено и что он не взялся неизвестно откуда (например, был украден). Поэтому класс Car следует переписать следующим образом:

```
class Car
{
  private $gas = 0;
  private function addGas($amount)
  {
     $this->gas = $this->gas + $amount;
     echo "в бак залито $amount галлонов бензина";
  }
  function buyGas($amount)
  {
     $this->addGas($amount);
  }
}
```

Теперь единственный возможный способ изменения значения атрибута $gas класса Car (т.е. заправки автомобиля) заключается в использовании метода buyGas(). В свою очередь, этот метод использует скрытый метод addGas(). Прямой вызов метода addGas() приведет к возникновению ошибки.

```
$new_car = new Car;
$new_car->addGas(5);
```

Для того чтобы этого не произошло, необходимо воспользоваться методом buyGas():

```
$new_car = new Car;

$new_car->buyGas(5);
```

В результате получим следующий результат:

```
в бак залито 5 галлонов бензина
```

Хороший стиль программирования заключается в сокрытии как можно большего количества деталей реализации класса. Все атрибуты должны быть скрытыми (private), а методы (те из них, которые должны быть открытыми) — public.

В языке PHP атрибуты и методы класса могут быть объявлены открытыми несколькими способами. Это связано с тем, что по умолчанию уровень доступа является открытым. Так, выражение

```
public $gas = 0;
```
равносильно
```
var $gas = 0;
```

Использование исключений

Для обработки ошибок в языке PHP используется класс `Exception` (исключение). Его можно применять для обработки нежелательных событий, которые могут возникнуть в сценарии. Если возникает заранее определенная разработчиком ситуация, то выполняется соответствующая процедура ее обработки. С точки зрения объектно-ориентированного подхода этот процесс называется *генерацией исключения*.

В классе `Car` автомобиль может двигаться до тех пор, пока в его баке есть горючее (т.е. `$gas>0`). Поэтому сценарий должен правильно реагировать на ситуацию, когда значение атрибута `$gas` станет отрицательным. Это и будет исключением. Для его обработки необходимо воспользоваться классом `Exception` с помощью следующего фрагмента кода:

```
$this->gas = $this->gas - 5;
  try
  {
     if ($this->gas < 0)
     {
       throw new Exception("Отрицательное количество бензина. ");
     }
  }
  catch (Exception $e)
  {
     echo $e->getMessage();
     echo "\n<br />\n";
     exit();
  }
```

В приведенном фрагменте содержатся блоки `try` и `catch`.

- ✓ В блоке `try` выполняется проверка условия. Если результат проверки условия — `TRUE`, то генерируется исключение, т.е. создается новый объект класса `Exception`. Этот объект имеет атрибут, в котором хранится сообщение, которое будет использоваться при генерации исключения.
- ✓ В блоке `catch` исключение перехватывается, и ему присваивается имя `$e`. После этого выполняется обработка исключения. В частности, вызывается *метод* `getMesssage()` *класса* `Exception`. *Этот метод возвращает сообщение*, заданное при создании объекта `Exception`, которое выводится с помощью функции `echo`.

Если исключение не сгенерировано, блок `catch` игнорируется, а выполнение сценария продолжается.

Копирование объектов

Для копирования объектов в языке PHP используется специальный метод `__clone()` (с двумя символами подчеркивания). При необходимости этот метод можно переопределить.

По умолчанию (т.е. без переопределения) метод `__clone()` просто скопирует все атрибуты одного объекта в атрибуты другого объекта.

Рассмотрим следующий класс с переопределенным методом `__clone()`:

```
class Car
{
  private $gas = 0;
  private $color = "красный";
  function addGas($amount)
  {
     $this->gas = $this->gas + $amount;
     echo "в бак залито $amount галлонов бензина";
  }
  function __clone()
  {
     $this->gas = 0;
  }
}
```

Используя этот класс, можно создать объект, а затем скопировать его следующим образом:

```
$firstCar = new Car;
$firstCar->addGas(10);
$secondCar=$firstCar->__clone();
```

После выполнения этого фрагмента кода будут созданы два объекта класса `Car`.

- ✓ `$firstCar`. Этот автомобиль имеет красный цвет и 10 галлонов бензина, которые были добавлены с помощью метода `addGas()`.
- ✓ `$secondCar`. Этот автомобиль также имеет красный цвет, но 0 галлонов бензина. Объект `$secondCar` был создан как копия `$firstCar` с помощью вызова метода `__clone()`, который присваивает переменной `$gas` значение 0 и вообще не устанавливает цвет (`$color`).

Если в классе `Car` не переопределить метод `__clone()`, интерпретатор PHP создаст его самостоятельно, по умолчанию скопировав все атрибуты объекта `$firstCar` в `$secondCar`, т.е. автомобиль `$secondCar`, как и `$firstCar`, будет иметь красный цвет и 10 галлонов бензина

Удаление объектов

Удалить ранее созданный объект можно следующим образом:

```
unset($objName);
```

Ниже приведен пример, в котором объект класса `Car` создается, а затем удаляется.

```
$myCar = new Car;
unset($myCar);
```

После вызова функции `unset()` объект больше не существует.

В PHP имеется специальный метод `__destruct()`, который автоматически вызывается при удалении объекта. Ниже приведен класс, содержащий этот метод.

```
class Bridge
{
  function __destruct()
  {
    echo "Мост разрушен";
```

```
    }
}
```

При создании объекта класса `Bridge`, а затем его удалении

```
$bigBridge = new Bridge;
unset($bigBridge);
```

отобразится следующее сообщение:

```
Мост разрушен
```

Оно отображается вследствие вызова метода `__destruct()` при вызове функции `unset()`.

Метод `__destruct()` не является обязательным. По необходимости его можно переопределить, как методы `__construct()` и `__clone()`, и выполнить некоторые специфические действия при удалении объекта. Например, при удалении объекта может потребоваться закрыть некоторые файлы или переписать информацию в базу данных.

Часть IV

Стандартные PHP-приложения

"Ладно, наверное, я забыл сказать, что теперь у нас есть функция управления, которая автоматически подает сигнал тревоги при разрыве связи с Web-узлом Aquarium".

В этой части...

В части IV речь пойдет о том, как использовать язык PHP для решения стандартных задач программирования. Вы узнаете, как создавать сценарии, которые чаще всего используются в Web-приложениях, как язык PHP позволяет взаимодействовать с базами данных, операционными системами и приложениями электронной почты. После изучения материала этой части вы сможете писать сценарии, с которыми пользователи будут взаимодействовать посредством HTML-форм, а также обрабатывать различные данные и решать стандартные задачи, возникающие при разработке Web-приложений.

Глава 10

Основы создания Web-приложений

В этой главе...

- Обеспечение безопасности Web-узлов
- Отображение статических страниц
- Получение информации от пользователей с помощью HTML-форм
- Обработка информации, полученной от пользователей

Первоначально язык PHP был разработан исключительно для Web-программирования. Однако со временем он стал использоваться и для других целей. В настоящее время PHP по-прежнему часто применяется для разработки динамических Web-узлов. *Статические* (static) Web-страницы одинаковы для всех пользователей и не предоставляют возможности интерактивного взаимодействия. В свою очередь, *динамические* (dynamic) страницы предоставляют такую функциональность. В зависимости от введенной пользователем информации в браузере могут отображаться различные страницы. Например, перед тем как получить доступ к ресурсам некоторого Web-узла, пользователю может потребоваться ввести свое имя и пароль. При этом узел сможет "настроиться" на этого пользователя, основываясь на соответствующем профиле. Пользователь может также выбрать необходимый тип товара из интерактивного каталога и получить только нужную информацию.

Динамические Web-страницы получают информацию от пользователей с помощью HTML-форм. Данные, введенные в полях формы, обрабатываются, и в зависимости от результата этой обработки пользователю могут предоставляться различные Web-страницы. При этом для вывода различных страниц данные обработки можно сохранить (этот вопрос более подробно рассматривается в главе 12) или использовать их в условных операторах.

В данной главе не рассматриваются дескрипторы HTML, используемые для создания форм. Предполагается, что вы с ними уже знакомы. (В противном случае обратитесь к книге Эда Титтела, Натаньи Питс и Челси Валентайн *HTML 4 для "чайников"*, 3-е издание, которая вышла в издательстве "Диалектика".) В этой главе также рассматриваются вопросы обеспечения безопасности Web-узлов и использование языка PHP для отображения форм и обработки пользовательской информации.

Обеспечение безопасности Web-узла

Web-приложения очень часто подвергаются атакам взломщиков. Большинство Web-приложений размещается на открытых серверах и предоставляет набор услуг, товаров или информации любому посетителю. Динамические Web-узлы оказываются особенно уязвимыми, поскольку они получают информацию от своих посетителей. Несмотря на то что большинство пользователей являются нормальными людьми, которые пользуются информационными ресурсами в мирных целях, среди них все же есть злоумышленники. Как правило, их можно разделить на следующие категории.

- ✓ **Злоумышленники**, которые пытаются найти файлы с номерами кредитных карточек и другой конфиденциальной информацией, например с паролями.
- ✓ **Взломщики,** пытающиеся разрушить Web-узел. Они считают забавным взломать Web-узел или причинить вред просто для того, чтобы продемонстрировать свое мастерство.
- ✓ **Злоумышленники, пытающиеся нанести ущерб пользователям**, размещают на Web-сервере средства, которые наносят вред его посетителям.

Данную книгу не следует рассматривать как исчерпывающий источник информации, которая позволит решить все вопросы, связанные с обеспечением требуемого уровня защищенности Web-приложения. Это достаточно обширная и сложная область, которой посвящено большое количество специализированных книг. Однако некоторые вопросы все же заслуживают внимания и здесь. Следующие основные замечания позволят существенно повысить уровень защищенности Web-приложения, однако если оно предназначено для обработки действительно важной и секретной информации, лучше обратиться к соответствующим книгам или экспертам.

- ✓ **Обеспечение безопасности компьютера, на котором установлено Web-приложение.** Это относится к сфере ответственности системного администратора.
- ✓ **Ограничение доступа к информации**. Объем предоставляемой информации должен быть не больше, чем требуется для предоставления услуг или решения требуемой задачи. Информацию нужно хранить таким образом, чтобы ее нельзя было легко извлечь через Web.
- ✓ **Осторожность при получении информации от пользователей**. Необходимо тщательно проверять все данные, сгенерированные в клиентской части вашего приложения.
- ✓ **Настройка Web-сервера на функционирование в самом защищенном режиме.** Это потребует дополнительных забот, но, если информация очень важная и секретная, затраченные усилия того стоят.

Более подробно эти вопросы рассматриваются в следующих разделах.

Обеспечение безопасности компьютера, на котором установлен Web-узел

Первой линией защиты Web-приложения является обеспечение защищенности компьютера, на котором оно установлено. Ответственным за решение этой задачи, как правило, является системный администратор. При этом можно установить брандмауэр, активизировать режим шифрования/сокрытия паролей, воспользоваться утилитами сканирования и т.д. В большинстве случаев ни один из разработчиков Web-приложения не является системным администратором. Однако в любом случае необходимо провести тщательный анализ подходов к обеспечению безопасности либо с системным администратором, либо с Web-хостинговой компанией, предоставляющей ресурсы для развертывания Web-приложения.

Ограничение доступа к информации

Доступ к информации, хранящейся на Web-узле, должен быть ограниченным. Вне всякого сомнения, Web-страницы, предназначенные для просмотра пользователями, должны быть открытыми и находиться в соответствующем общедоступном каталоге Web-сервера. При этом пользователю нет необходимости видеть имена хранящихся в них файлов. Вы, наверное, посещали Web-узлы, которые отображали каталог со списком файлов. В общем случае так делать не следует, поскольку посетители смогут просмотреть любой удаленный файл.

Список файлов отображается в том случае, когда в URL-адресе не указано имя целевого файла, а в каталоге отсутствует файл, который отображается по умолчанию. Обычно Web-сервер настроен таким образом, чтобы в подобных ситуациях выводить определенный файл (обычно `index.html`), если явно не указано другое имя. В противном случае пользователь сможет увидеть список файлов, а это — весьма нежелательно. Лучше всего настроить Web-сервер таким образом, чтобы при попытке доступа к каталогам выводилось сообщение о невозможности получения доступа к каталогу.

```
Forbidden
You don't have permission to access /secretdirectory on this server.
(Доступ запрещен
У вас нет прав доступа к каталогу /secretdirectory на этом сервере.)
```

При настройке Web-сервера Apache это можно сделать с помощью директивы `Indexes` конфигурационного файла `httpd.conf`:

```
Options Indexes // позволяет отображать файлы каталога
Options -Indexes // запрещает отображать файлы каталога
```

При настройке других Web-серверов следует обратиться к соответствующей документации.

Кроме того, не следует присваивать файлам легко угадываемые имена. Например, если в Web-приложении используется файл с пользовательскими паролями, то присваивание ему имени `passwords.php` окажется далеко не лучшей идеей. Вместо этого такому файлу следует дать непримечательное или обманчивое имя, например `vegetableRecipes.php`. Можно возразить, что это противоречит правилу присваивания файлам отчетливых имен. (Об этом упоминается во многих других главах этой книги.) Однако данный случай является особенным. Некоторые злоумышленники вводят в браузере URL-адрес `www.yoursite.com/password.html` и ждут, что из этого получится.

Далеко не вся информация Web-узла должна быть общедоступной. Например, база данных не должна размещаться в открытом каталоге. Фактически подобную информацию лучше всего хранить совсем на другом компьютере, доступ к которому нельзя получить через Internet.

Осторожность при получении информации от пользователей

Пользователи могут случайно или злоумышленно ввести в форме опасную (нежелательную) информацию. Поэтому перед ее сохранением или использованием необходимо проверить ее на корректность и отсутствие опасных символов или строк. Введенная информация (даже случайная) может нанести существенный вред сценарию или базе данных. Особенно опасными являются дескрипторы HTML (например, `<script>`), поскольку их можно использовать для размещения на Web-узле опасных сценариев. Если введенную пользователем информацию принять и сохранить в базе данных без проверки, можно столкнуться с целым ворохом проблем, особенно в том случае, если эту информацию впоследствии будут использовать (и запускать) другие посетители Web-узла. Вопросы проверки данных, вводимых в полях форм, более подробно рассматриваются ниже, в разделе "Проверка данных".

Использование безопасного Web-сервера

Взаимодействие Web-приложения с посетителями нельзя считать абсолютно безопасным. Если находящийся на Web-узле файл передается в клиентскую часть приложения (браузер), то вполне вероятно, что во время передачи данных по Internet кто-то посторонний сможет прочитать содержимое файла. Для большинства Web-узлов это не является проблемой. Но если среди передаваемых данных содержатся номера кредитных карточек или пароли, то

вопросы обеспечения их конфиденциальности сразу же выдвигаются на передний план. Уровень защищенности данных можно повысить и при использовании безопасного Web-сервера.

На безопасных Web-серверах для защиты данных часто используется протокол SSL (Secure Sockets Layer). При его использовании информация кодируется и лишь после этого передается через Web. Клиентское программное обеспечение выполняет обратное декодирование. Вне всякого сомнения, развертывание безопасного Web-сервера требует дополнительных усилий, но для некоторых приложений без этого просто не обойтись.

При взаимодействии с использованием протокола SSL URL-адрес начинается с префикса `https`, а не `http`.

Каждый Web-сервер можно защитить по-своему. Информацию о том, как включить режим использования защищенного протокола SSL, следует искать в соответствующей документации. Например, при использовании сервера Apache необходимую информацию можно найти на двух Web-узлах: `www.modssl.org` и `www.apache-ssl.org`. Если вы решили воспользоваться Web-сервером IIS компании Microsoft, то информацию о протоколе SSL можно найти на Web-узле `www.microsoft.com`.

Отображение статических Web-страниц

Наиболее простыми являются статические Web-страницы. Если при разработке Web-узла таких страниц вполне достаточно, то в использовании языка PHP нет никакой необходимости. Однако в то же время статические Web-страницы могут использоваться для взаимодействия с динамическими страницами.

Язык PHP можно использовать для отображения любых Web-страниц, в том числе и статических. При этом для вывода HTML-кода можно воспользоваться функцией `echo`. Если же на странице содержатся только дескрипторы HTML, которые нужно использовать в сценарии PHP, то лучше всего включить определенный файл в требуемое место следующим образом:

```
include("имя_файла");
```

Если по каким-то причинам статическую Web-страницу понадобилось преобразовать в сценарий PHP, выполните следующее. Добавьте в начало и в конец соответствующего файла дескрипторы PHP (`<?php` и `?>`), а затем добавьте функцию `echo` в начале этого файла и заключите весь код HTML в одинарные кавычки.

Работа с HTML-формами

Для того чтобы Web-страница стала интерактивной, она должна обеспечивать получение информации от пользователей. Это можно осуществить с помощью HTML-форм. Получаемые от пользователей данные могут быть простыми, такими как регистрационное имя и пароль. Однако форма может быть достаточно длинной и сложной и состоять из большого количества полей. Так, например, форма, предназначенная для покупки товаров в электронном магазине, должна предоставлять возможность ввода различной информации о покупателе (например, его адреса и номера кредитной карточки). Одним словом, внешний вид формы и набор ее полей — это серьезные вопросы для настоящего исследования.

При использовании HTML-формы для сбора информации сценарий сначала должен отобразить ее на Web-странице. Затем пользователь может ввести необходимые данные, заполняя поля или выбирая требуемые значения из списка. После этого нужно щелкнуть на соответствующей кнопке, чтобы введенная информация была отправлена другому сценарию, который и будет ее обрабатывать.

В этой главе рассматриваются вопросы использования форм при разработке динамических Web-узлов. Зачастую собранная с их помощью информация сохраняется в базе данных. Если вы планируете использовать формы HTML совместно с базой данных MySQL, то для получения более подробной информации можно обратиться к книге *PHP & MySQL For Dummies* издательства Wiley Publishing, Inc., автором которой является ваш покорный слуга.

Получение информации от посетителей Web-узла

Для получения информации от посетителей Web-узлов используются формы HTML. Если вы не имеете опыта работы с языком HTML, обратитесь к книге Эда Титтела, Натаньи Питс и Челси Валентайн *HTML 4 для "чайников"*, 3-е издание, которая вышла в издательстве "Диалектика".

Отображение HTML-форм

Для вывода формы средствами PHP можно воспользоваться любым из следующих способов.

- ✓ **Используйте для вывода HTML-форм функцию `echo`**, как в следующем примере:

```
echo "<form action='processform.php' method='POST'>\n
  <input type='text' name='name'>\n
  <input type='submit' value='Отправить данные'>\n
  </form>\n";
```

- ✓ **Используйте обычный код HTML за пределами разделов с кодом PHP**. При создании статической формы нет необходимости включать ее в код PHP. Например, в следующем примере выводится форма из предыдущего фрагмента:

```
<?php
  код PHP
?>
<form action="processform.php" method="POST">
<input type="text" name="fullname">
<input type="submit" value="Отправить данные">
</form>
<?php
  код PHP
?>
```

Каждый из приведенных примеров позволяют вывести форму, представленную на рис. 10.1.

В этой форме содержится одно пустое текстовое поле и кнопка с именем `Отправить данные`. Пользователь может ввести в это поле свое имя, а затем щелкнуть на кнопке. В результате введенная информация будет передана сценарию, имя которого задано в атрибуте `action` дескриптора `<form>`. В приведенном примере этот атрибут имеет вид `action="processform.php"`. Следовательно, введенные данные будут обработаны сценарием `processform.php`. (Естественно, имя `processform.php` было выбрано исключительно в учебных целях. На самом деле в атрибуте `action` может быть указано любое имя.)

Язык PHP позволяет использовать в формах переменные, что существенно повышает их эффективность. Такой подход позволяет заполнять текстовые поля нужной информацией и динамически создавать раскрывающиеся списки, переключатели и флажки.

Вывод информации в текстовых полях

Иногда может понадобиться заполнить текстовое поле данными, а не выводить его пустым. Например, в поле может содержаться значение по умолчанию. В других случаях нужно обеспечить, чтобы при вводе некорректной информации пользователю пришлось бы повторно вводить данные лишь в те поля, где он допустил ошибку.

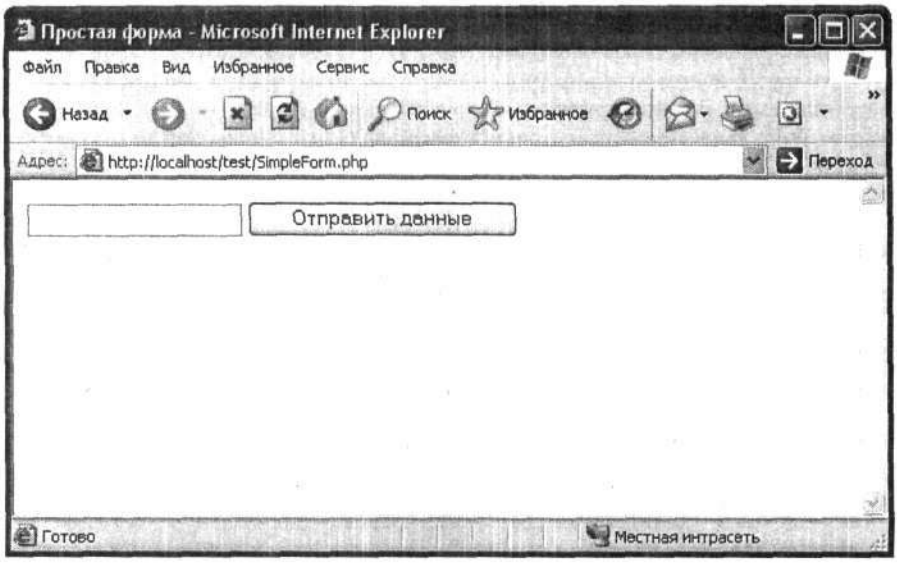


Рис. 10.1. Форма, созданная с использованием дескрипторов HTML

Вывести текстовое поле с данными можно с использованием кода

```
<input type="text" name="fieldname" value="значение">
```

Пусть, например, нужно разработать форму для получения имени покупателя и его адреса. Предположим, известно, что большая часть клиентов проживает в США. Поэтому вполне логично, чтобы в соответствующем поле сразу же выводилось значение США, заданное по умолчанию. При этом если покупатель действительно из Соединенных Штатов, ему ничего не требуется вводить и, следовательно, делать ошибки. Если же покупатель из другой страны, ему придется ввести корректное название своей страны.

Для отображения такого поля необходимо воспользоваться следующим выражением:

```
<input type="text" name="country" value="США">
```

В некоторых случаях в поле формы может потребоваться вывести значение переменной PHP. Предположим, имеется информация о покупателе (например, его телефонный номер), которая хранится в базе данных. Эти данные нужно вывести в форме, чтобы пользователь смог изменить неверную или устаревшую информацию. Для решения этой задачи необходимо выполнить следующее. Во-первых, нужно извлечь информацию из базы данных (как это сделать, описывается в главе 12) и сохранить ее в соответствующих переменных. Затем для вывода формы можно воспользоваться двумя способами. Первый способ заключается в использовании кода PHP, встроенного в дескриптор HTML `<input>`:

```
<input type="text" name="phone" value="<?php echo $phone ?>">
```

Для вывода дескриптора HTML можно также воспользоваться функцией `echo`:

```
echo "<input type='text' name='phone' value='$phone'>";
```

При создании формы с большим количеством полей и немногими переменными, которые с ними связаны, более эффективным является первый способ. Второй подход оказывается гораздо более эффективным при использовании большого количества переменных.

Сценарий из листинга 10.1 выводит форму с информацией о покупателе. Результат работы этого сценария показан на рис. 10.2.

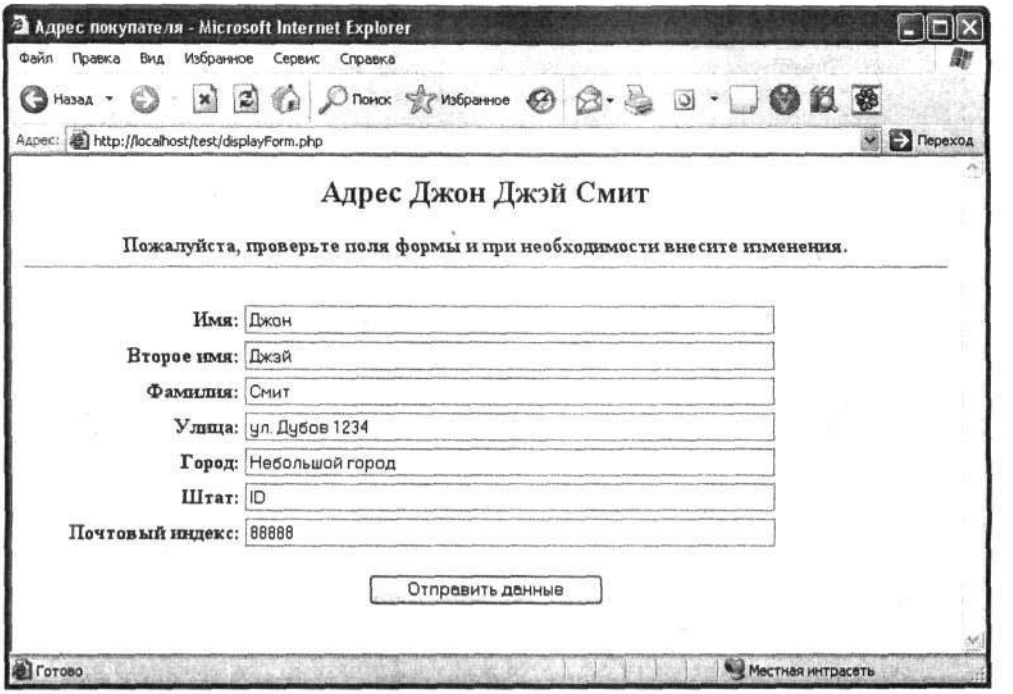


Рис. 10.2. Форма с адресом покупателя

Листинг 10.1. Сценарий для отображения HTML-формы

```
<?php
/*  Имя файла:        displayForm
 *  Описание:         Сценарий отображает форму и заполняет
 *                    ее поля значениями из массива.
 */
  echo "<html>
        <head><title>Адрес покупателя</title></head>
        <body>";
  $customer = array("firstName"=>"Джон",
                    "midName"=>"Джэй",
                    "lastName"=>"Смит",
                    "street"=>"ул. Дубов 1234",
                    "city"=>"Небольшой город",
                    "state"=>"ID",
                    "zip"=>"88888");
  $labels = array("firstName"=>"Имя:",
                  "midName"=>"Второе имя:",
                  "lastName"=>"Фамилия:",
                  "street"=>"Улица:",
                  "city"=>"Город:",
                  "state"=>"Штат:",
                  "zip"=>"Почтовый индекс:");
  echo "<h2 align='center'>Адрес
           {$customer['firstName']}
           {$customer['midName']}
```

```
            {$customer['lastName']}</h2>\n";
  echo "<p align='center'>
        <b> Пожалуйста, проверьте поля формы и при
            необходимости внесите изменения.</b>
        <hr>
        <form action='processform.php' method='POST'>
        <table width='95%' border='0' cellspacing='0'
                                        cellpadding='2'>\n";
  foreach($customer as $field=>$value)
  {
    echo "<tr>
          <td align='right'> <B>{$labels[$field]} </br></td>
          <td><input type='text' name='$field' size='65 '
              maxlength='65' value='{$customer[$field]}'></td>
          </tr>";
  }
  echo "</table>
        <div align='center'><p><input type='submit' value='Отправить
данные'> </p></div>
        </form>";
?>
</body></html>
```

Вот некоторые пояснения к коду сценария из листинга 10.1 (файл `displayForm.php`)

- ✓ **В начале сценария создается массив `$customer`, содержащий информацию для вывода в полях формы.** В реальных приложениях эти данные извлекаются из базы данных, файла или других источников.
- ✓ **Затем создается массив `$labels` с именами полей.**
- ✓ **Для обработки данных формы используется сценарий с именем `processform.php`.**
- ✓ **Форма представляется в виде таблицы HTML.** Таблицы являются важным элементом HTML. Более подробную информацию о таблицах HTML можно найти в книге Эда Титтела, Натаньи Питс и Челси Валентайн *HTML 4 для "чайников"*, 3-е издание, которая вышла в издательстве "Диалектика".
- ✓ **Проход по массиву `$customer` осуществляется в сценарии с использованием оператора `foreach`.** На каждой итерации цикла выводится строка таблицы с соответствующим значением из массива `$field`.

С точки зрения обеспечения безопасности следует всегда использовать атрибут `maxlength`, который определяет максимальное количество символов, которое можно ввести в поле формы. Это позволит ограничить допустимый объем информации и предотвратить опасность ввода опасного кода. Если полученная от пользователей информация будет храниться в базе данных, то соответствующий столбец таблицы также должен иметь ширину `maxlength`.

Добавление в форму раскрывающихся списков, переключателей и флажков

Другие элементы HTML-форм, такие как раскрывающиеся списки, переключатели и флажки, также можно использовать вместе с переменными. Для этого достаточно вставить требуемую переменную в соответствующий дескриптор HTML, а затем вывести его с помо-

щью функции echo. Ниже приведен пример создания раскрывающегося списка с использованием переменных PHP.

```
echo "<select name='dinner' >
        <option>$dinner1</option>
        <option>$dinner2</option>
      </select>";
```

Значения конкретных элементов списка определяются значениями соответствующих переменных. Например, в переменной $dinner1 может содержаться значение цыпленок, а в $dinner2 — рыба. После выбора пользователем нужного значения данные формы передаются для обработки следующему сценарию.

Точно так же можно создавать и переключатели. Например, при использовании приведенного ниже фрагмента пользователь может выбрать значение цыпленок или рыба:

```
echo "<input type='radio' name='dinner'
          value='$dinner1'>$dinner1
      <input type='radio' name='dinner'
          value='$dinner2'>$dinner2";
```

Флажки позволяют одновременно выбрать больше одного значения. Следовательно, атрибут name дескриптора <input> должен быть массивом.

```
echo "<input type='checkbox' name='dinner[]'
          value='$dinner1'>$dinner1
      <input type='checkbox' name='dinner[]'
          value='$dinner2'>$dinner2";
```

В данном примере введенная информация сохраняется в массиве $dinner. Если выбраны оба значения, то этот массив будет иметь следующий вид:

```
$dinner[0]=цыпленок
$dinner[1]=рыба
```

В сценарии из листинга 10.2 выводится Web-страница, содержащая список для выбора даты. По умолчанию выбирается текущая дата.

Листинг 10.2. Сценарий создания формы для выбора даты

```
<?php
/*  Имя файла:        displayDate
 *  Описание:         Сценарий, отображающий три раскрывающихся
 *                    списка для выбора месяца, дня и года.
 *                    По умолчанию выбирается текущая дата.
 */
  echo "<html>
        <head><title>Дата</title></head>
        <body>";
/* Создание массива, индексами элементов которого
 * являются номера, а значениями - названия месяцев.
 */
  $monthName = array(1=> "Январь", "Февраль", "Март",
                         "Апрель", "Май", "Июнь",
                         "Июль", "Август", "Сентябрь",
                         "Октябрь", "Ноябрь", "Декабрь");
  $today = Time();  #текущая дата
  echo "<div align='center'><b>Выберите дату:</b>
        <form action='processform.php' method='POST'>\n";
  /* Создание раскрывающегося списка для месяца */
  $todayMO = date("m", $today);  #получение текущего месяца из $today
  echo "<select name='dateMO'>\n";
```

```
  for ($n=1; $n<=12; $n++)
  {
    echo "<option value=$n";
    if ($todayMO == $n)
    {
      echo " выбран";
    }
    echo "> $monthName[$n]\n";
  }
  echo "</select>";

  /* Создание раскрывающегося списка для дня */
  $todayDay= date("d", $today);    #получение текущего дня из $today
  echo "<select name='dateDay'>\n";
  for ($n=1; $n<=31; $n++)
  {
    echo " <option value=$n";
    if ($todayDay == $n )
    {
      echo " выбран";
    }
    echo "> $n\n";
  }
  echo "</select>\n";

  /* Создание раскрывающегося списка для года */
  $startYr = date("Y", $today);    #получение текущего года из $today
  echo "<select name='dateYr'>\n";
  for ($n=$startYr; $n<=$startYr+3; $n++)
  {
    echo " <option value=$n";
    if ($startYr == $n )
    {
      echo " выбран";
    }
    echo "> $n\n";
  }
  echo "</select>\n";
  echo "</form>\n";
?>
</body>
</html>
```

В приведенном сценарии сначала создается массив `$monthName`, ключами которого являются номера месяцев, а значениями — их названия. В переменной `$today` размещается текущая дата.

В остальной части сценария отображаются три раскрывающихся списка формы для ввода трех элементов даты: месяца, дня и года. Названия месяцев в цикле берутся из массива `$monthName`. При этом номер каждого месяца сравнивается с номером текущего месяца. Если эти номера совпали, то в раскрывающийся список добавляется атрибут с флагом `выбран`. Именно этот месяц и используется в списке в качестве элемента по умолчанию.

Аналогичным образом создаются раскрывающиеся списки для выбора дня и года. При этом в качестве значений по умолчанию также выбираются текущий день и год.

На рис. 10.3 представлена форма, созданная с использованием сценария из листинга 10.2.

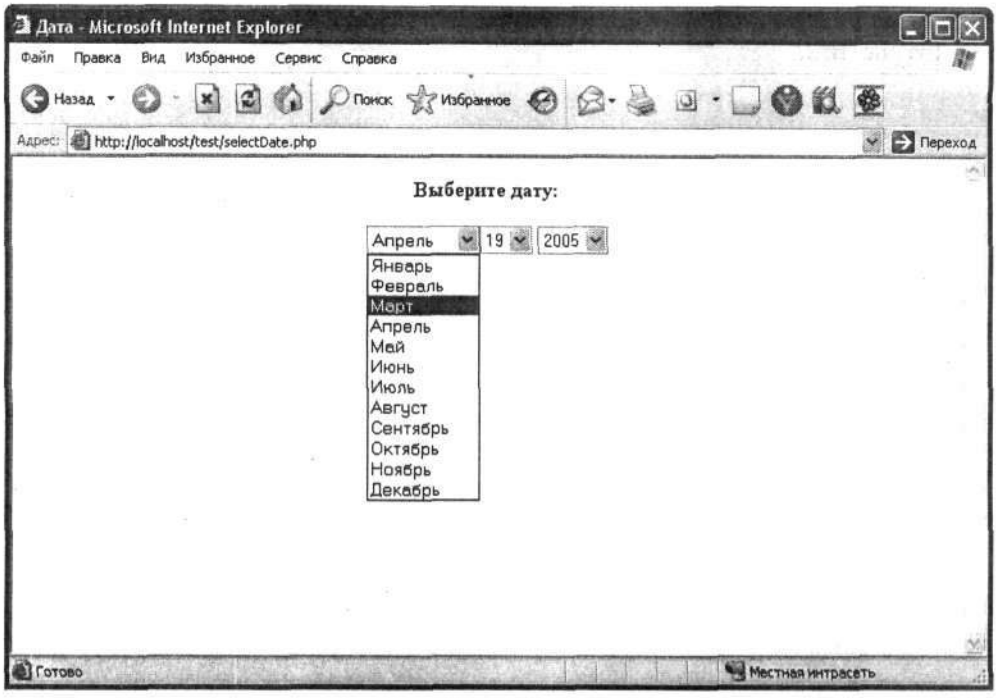


Рис. 10.3. Форма для выбора даты

Получение информации

В атрибуте `action` дескриптора `<form>` указывается имя сценария, который будет обрабатывать данные формы: `action="`*имя_сценария*`"`. Например, в листингах 10.1 и 10.2 использовался сценарий `action="processform.php"`. После щелчка на кнопке передачи данных введенная информация передается сценарию, указанному в атрибуте `action`.

Как говорилось в главе 6, переданные сценарию данные можно извлечь из встроенных массивов PHP. Если при передаче использовался метод `POST`, то данные будут размещены в массиве `$_POST`, а если метод `GET` — в массиве `$_GET`. Независимо от выбранного метода передачи данные можно получить также из массива `$_REQUEST`. При этом в качестве ключей этих массивов используются имена полей формы, а значениями являются пользовательские данные.

Рассмотрим пример, в котором информация передается с использованием метода `POST`, а в форме содержится следующее поле:

```
echo "<input type='text' name='firstName'>";
```

Поскольку именем поля является `firstName`, то для получения введенного значения можно воспользоваться выражением `$_POST['firstName']`. Точно так же можно получить данные, которые были введены в форме с использованием раскрывающегося списка или переключателя. Если в форме содержатся флажки, пользователь может одновременно выбрать несколько значений. Поэтому соответствующий элемент массива `$_POST` тоже будет массивом (т.е. `$_POST` будет многомерным). Например, если перед обедом было выбрано оба возможных варианта, `цыпленок` и `рыба` (см. предыдущий раздел), то в сценарии PHP эти значения можно получить следующим образом:

```
$_POST['dinner'][0] = цыпленок
$_POST['dinner'][1] = рыба
```

Сценарий из листинга 10.3 позволяет отобразить значения всех полей формы, представленной на рис. 10.2.

Листинг 10.3. Сценарий для вывода всех полей формы

```
<?php
/*  Script name:      displayFormFields
 *  Description:      Сценарий, который выводит информацию,
 *                    введенную в полях формы.
 */
  echo "<html>
        <head><title>Адрес покупателя</title></head>
        <body>";
  foreach ($_POST as $field => $value)
  {
     echo "$field = $value<br>";
  }
?>
</body></html>
```

POST против GET

Для передачи данных формы можно использовать два метода: `POST` и `GET`. Каждый из этих методов имеет как преимущества, так и недостатки, а, кроме того, информация при использовании каждого из них передается по-разному.

- ✓ **Метод** `GET`. При использовании этого метода данные формы передаются путем их добавления к URL-адресу вызываемого сценария, предназначенного для обработки информации, например `processform.php?lname=Smith&fname=Goliath`

  Преимуществами этого метода являются его простота и скорость. Недостатки метода `GET` связаны с возможностью передачи лишь ограниченного количества данных, а также с отображением информации в браузере, что с точки зрения безопасности является весьма нежелательным.
- ✓ **Метод** `POST`. При использовании метода `POST` данные формы передаются в области данных передаваемых пакетов. Основные преимущества этого метода заключаются в возможности передачи неограниченного количества данных и их большей защищенности. К недостаткам следует отнести небольшую скорость и необходимость затраты дополнительных усилий разработчиков.

В CGI-программах, отличных от сценариев PHP, полученные из браузера данные необходимо преобразовать и разместить в переменных. В этом случае гораздо проще воспользоваться методом `GET`. Именно поэтому большинство разработчиков его и применяют. Однако в языке PHP все требуемые действия выполняются автоматически. В языке PHP одинаково просто применять оба метода, и `GET` и `POST`. Поэтому с учетом преимуществ метода `POST` (неограниченное количество данных и большая безопасность) в языке PHP лучше использовать именно его.

Этот сценарий можно использовать для обработки данных формы, представленной на рис. 10.2. Для этого в сценарии `displayForm.php` (см. листинг 10.1) необходимо изменить атрибут `action` дескриптора `<form>` на `action="displayFormFields.php"`. Тогда после щелчка на кнопке `Отправить` данные будет вызван сценарий `displayFormFields.php` (см. листинг 10.3), который сгенерирует Web-страницу со следующей информацией:

```
firstName = Джон
lastName = Смит
street = ул. Дубов 1234
```

```
city = Небольшой город
state = ID
zip = 88888
```

После выполнения сценария из листинга 10.3 в окне браузера отобразятся значения, которые были переданы через форму, представленную на рис. 10.2. В большинстве случаев информацию необходимо не только отображать, но и использовать в условных операторах или сохранять в базе данных.

Проверка данных

Перед тем как использовать введенную информацию в сценарии, ее необходимо проверить и удостовериться в том, что получены именно те данные, которые ожидались. Например, пользователь может оставить незаполненным обязательное поле, ошибиться при вводе, так что данные не будут иметь никакого смысла, или ввести информацию, которая может нанести ущерб Web-узлу. Поэтому никогда не стоит доверять информации, введенной пользователями в форме. Ее *всегда* необходимо проверять.

Контроль данных

Обычно *проверка данных* (validating information), введенных в форме, связана с проверкой на наличие незаполненных полей и на соответствие введенных данных заданному формату.

- ✓ **Проверка пустых полей**. Можно потребовать, чтобы некоторую информацию пользователь ввел обязательно, например свое имя и адрес электронной почты. Если соответствующее поле формы оказалось все же незаполненным, то форму можно отобразить повторно и сообщить пользователю о том, что в обязательное поле не было введено никакой информации.
- ✓ **Проверка на соответствие заданному формату**. Обычно пользователь должен ввести данные, которые бы соответствовали определенному формату. Например, почтовый индекс всегда должен содержать пять цифр. Поэтому, если полученные данные не удовлетворяют заданному формату, пользователю необходимо указать на допущенные ошибки и сообщить о том, что их нужно ввести повторно. Например, строку `ab3&*xx` вряд ли можно использовать в качестве почтового индекса.

Для того чтобы проверить, было ли заполнено определенное поле, можно воспользоваться функцией `empty()` со следующим синтаксисом:

```
empty($_POST[имя_поля]);
```

Например, в сценарий можно добавить следующий код:

```
if(empty($_POST['имя_поля']))
{
     echo "Поле не заполнено";
     // повторное отображение формы
}
```

Проверка введенной информации поможет определить ее соответствие нужному формату. Например, если в качестве почтового индекса (zip) пользователь введет строку `8899776`, это будет явной ошибкой. Она слишком длинна для индекса (содержит больше пяти знаков) и слишком коротка для формата zip+4 знака (т.е. девять знаков).

Проверка формата данных формы поможет также предотвратить ввод нежелательной информации, которая может нанести ущерб Web-узлу или базе данных, или кражу информации у других посетителей. Например, очень нежелательно разрешать ввод дескрипторов HTML,

поскольку это может привести к непредсказуемым результатам при их передаче браузеру других пользователей. (Особенно опасным является дескриптор <script>.)

Проверка данных, введенных в *каждом* поле формы, поможет избежать многих проблем. Однако ее следует выполнять с умом. Необходимо стремиться к выявлению как можно большего количества некорректной информации, однако при этом нужно обеспечить и возможность прохождения легитимных данных. Например, при проверке правильности телефонного номера следует контролировать длину введенного числового значения. Тут может возникнуть проблема, связанная с тем, что такие номера, как 555-5555 и (888) 555-5555, также являются допустимыми. Поэтому длину телефонного номера придется увеличить до 14 символов и разрешить ввод дефисов, скобок и пробелов. Как видите, не все так просто, как казалось на первый взгляд. Одним словом, можно дать один следующий совет: хорошенько подумайте о том, какую информацию вы намерены разрешить вводить пользователям.

Использование регулярных выражений для проверки пользовательской информации

Для проверки введенной в полях формы информации можно воспользоваться регулярными выражениями, которые уже рассматривались в главе 7. В этом случае полученные данные проверяются на соответствие заданному шаблону. Если данные шаблону не соответствуют, значит, пользователь ввел некорректную информацию и ему придется ввести ее еще раз.

Например, нужно проверить корректность ввода пользователем в поле формы своей фамилии. При этом вполне логично предположить, что имя может содержать буквы (но не числа), а также символ апострофа (О'Хара), дефис (Смит-Джонс) и пробел (Ван Дайк). Кроме того, достаточно трудно представить себе фамилию, состоящую более чем из 50 символов. Поэтому для проверки введенной строки можно воспользоваться следующим кодом:

```
$last_name = trim($_POST['last_name']);
if ( !ereg("[A-Яа-яA-Za-z' -]{1,50}", $last_name))
{
    сообщение о необходимости повторного ввода фамилии
}
```

В приведенном фрагменте для удаления ненужных пробелов в начале и в конце строки используется функция `trim()`. Затем для проверки соответствия введенной строки заданному шаблону (регулярному выражению) используется оператор `if`. Обратите внимание на операцию `!` в условии этого оператора: если введенная строка не соответствует шаблону, то выполняется последовательность команд в фигурных скобках.

Если в список допустимых символов регулярного выражения, ограниченный квадратными скобками (`[ ]`), необходимо добавить дефис (`-`), его следует разместить либо в начале, либо в конце этого списка. В противном случае (т.е. при размещении между двумя символами) дефис будет интерпретироваться как специальный символ, который используется для задания диапазона.

Использование PHP для создания, отображения формы и проверки ее полей

Сценарий из листинга 10.4 позволяет проверить данные, которые были введены в полях формы. При его первом вызове отображается форма с пустыми полями. После заполнения полей формы и щелчка на кнопке `Отправить` данные введенные данные передаются этому же сценарию, который выполняет проверку на наличие пустых и неправильно заполненных полей. Если в процессе проверки были обнаружены ошибки, то отобразится соответствующее сообщение и форма выведется повторно. Если все данные были введены правильно, отобразится имя пользователя и его адрес.

В сценарии из листинга 10.4 подключаются два дополнительных файла. Один из них предназначен для создания массива, который будет использоваться для построения формы (листинг 10.5). Второй файл обеспечивает ее отображение (листинг 10.6).

Листинг 10.4. Сценарий проверки всех данных, введенных в полях формы

```
<?php
/*  Имя файла:        validateForm
 *  Описание:         Отображает форму и проверяет
 *                    введенную информацию.
 */
  include("info.inc");                                    #6
  ########################
  ## Вывод пустой формы ##
  ########################
  if(!isset($_POST['Submit']))                           #10
  {
    include("addressForm.inc");
  }
#########################################################
## Проверка данных, введенных в форме. Создается       ##
## массив пустых или некорректно заполненных           ##
## полей. Если при заполнении формы были допущены      ##
## ошибки, выводится соответствующее сообщение и       ##
## форма отображается повторно. В противном случае     ##
## отображается введенная информация.                  ##
#########################################################
  else                                                   #21
  {
    foreach($_POST as $field=>$value)                    #23
    {
      if(empty($_POST[$field]))                          #25
      {
        if($field != "midName")
        {
          $blanks[$field] = "blank";                     #29
        }
      }
      else                                               #33
      {
        $value = trim($value);
        if($field != "zipcode")
        {
          if(!ereg("^[A-Яа-яA-Za-z0-9' .-]{1,65}$", $value))
          {
            $formats[$field] = "bad";
          }
        }
        elseif($field == "zipcode")
        {
          if(!ereg("^[0-9]{5}(\-[0-9]{4})?", $value))
          {
            $formats[$field] = "bad";
          }
        }
      }
    }                                                    #51
```

```
  ### если хотя бы в одном поле допущена ошибка,  ###
  ### выводится соответствующее сообщение и форма ###
  ### отображается заново                          ###
 if (@sizeof($blanks) > 0 or @sizeof($formats) > 0)#54
  {
    if (@sizeof($blanks) > 0)
    {
      echo "<b>Вы не заполнили одно или несколько
               обязательных полей. Необходимо
               заполнить следующие поля: </b><br>";
      foreach($blanks as $field => $value)
      {
       echo "   {$labels[$field]}<br>";
      }
    }
    if (@sizeof($formats) > 0)
    {
     echo "<b>В одном или нескольких полях содержатся
              некорректные данные. Исправьте
              информацию в следующих полях:
</b><br>";
      foreach($formats as $field => $value)
      {
       echo "   {$labels[$field]}<br>";
      }
    }
    echo "<hr>";
    include("addressForm.inc");
  }
  else                                                #78
  {
   ### если ошибок не обнаружено, отображаются ###
   ### введенные имя и адрес                   ###
    echo "<html><head><title>Имя и адрес
            </title></head><body>\n";
    foreach($_POST as $field=>$value)
    {
      if($field != "Submit")
      {
        echo "{$labels[$field]} $value<br>\n";
      }
    }
    echo "</body></html>";
  }
 }
?>
```

Некоторые строки сценария из листинга 10.4 были пронумерованы, чтобы можно было сделать акцент на ряде следующих важных моментов.

- ✓ **Строка 6**. Это выражение подключает файл `info.inc`, который создает массив `$labels`, который в сценарии будет использоваться позже. Исходный код включаемого файла приведен в листинге 10.5. (Более подробно о включении файлов в сценарий см. в главе 8.)
- ✓ **Строка 10**. Оператор `if` проверяет, существует ли в массиве `$_POST` элемент с индексом `Submit`. Дело в том, что кнопка, после щелчка на которой данные

полей формы передаются серверному сценарию, имеет имя `Submit`. Поэтому после щелчка на этой кнопке в массив `$_POST` будет помещено соответствующее значение. Именно этот факт и проверяется в условии оператора `if`. Если условное выражение оказалось истинным (т.е. элемента `Submit` не существует), выполняется блок, в котором подключается файл `addressForm.inc` (см. листинг 10.6), предназначенный для вывода формы. Блок оператора `if` выполняется при первом вызове основного сценария, что приводит к отображению формы с пустыми полями.

- ✓ **Строка 21**. С этой строки начинается выполнение `else`-блока (т.е. если в массиве `$_POST` имеется элемент `Submit`). В этом блоке введенные данные проверяются на корректность.
- ✓ **Строка 23**. В этой строке с помощью цикла `foreach` выполняется проход по всем элементам массива `$_POST`.
- ✓ **Строка 25**. В этой строке с использованием оператора `if` каждое поле проверяется на пустоту. Если поле оказалось заполненным, управление передается в строку 33, с которой начинается проверка формата введенных данных.
- ✓ **Строка 29**. В этой строке в массив `$blanks` добавляются элементы, соответствующие незаполненным полям (кроме поля `Второе Имя`, которое является необязательным и, следовательно, может оставаться пустым).
- ✓ **Строка 33**. В этой строке начинается блок `else`, в котором выполняется проверка формата значений, введенных в полях формы. Все поля, кроме почтового индекса, проверяются на наличие запрещенных символов. При этом допустимыми считаются буквы, числа, апостроф, пробел, точка и дефис. Корректность почтового индекса проверяется в отдельном блоке, поскольку он должен в точности соответствовать заданному шаблону. Если оказалось, что в каком-то из полей формы была введена неверная информация, соответствующий элемент добавляется в массив `$formats`.
- ✓ **Строка 51**. В этой строке завершается проверка правильности заполнения полей формы. В этот момент в сценарии существуют два массива, `$blanks` и `$formats`, в которых содержится информация о результатах проведенной проверки. Если никаких ошибок обнаружено не было, то массивы `$blanks` и `$formats` не создаются.
- ✓ **Строка 54**. В этой строке с использованием оператора `if` проверяется, были ли при вводе данных в поля формы допущены какие-либо ошибки. Для этого проверяется, существуют ли массивы `$blanks` и `$formats`. Если хотя бы один из этих массивов был создан, выводится сообщение об ошибке и форма отображается повторно.
- ✓ **Строка 78**. Этот блок `else` выполняется в том случае, если при вводе данных не было допущено ошибок. В результате его выполнения отобразится вся информация, введенная пользователем в форме.

Обратите внимание, что сценарий из листинга 10.4 является достаточно гибким и позволяет обрабатывать информацию, введенную в любой форме (за исключением проверки данных в строках 33–50). Однако эти строки можно без особых проблем модифицировать. Остальная же часть сценария останется без изменений.

В листинге 10.5 приведен исходный код первого файла, включаемого в сценарий из листинга 10.4. В этом сценарии создается массив, используемый для отображения формы и сообщений об ошибках.

Листинг 10.5. Сценарий создания массива

```
<?php
/*  Имя файла:        info.inc
 *  Описание:         создает массив имен для использования
 *                    в форме.
 */
$labels = array("firstName"=>"Имя:",
                "midName"=>"Второе имя:",
                "lastName"=>"Фамилия:",
                "street"=>"Улица:",
                "city"=>"Город:",
                "state"=>"Штат:",
                "zipcode"=>"Почтовый индекс:");
?>
```

В листинге 10.6 приведен исходный код сценария, предназначенного для отображения формы. Он основывается на сценарии из листинга 10.1, который выводит в окне браузера форму, представленную на рис. 10.2.

Листинг 10.6. Сценарий для отображения формы

```
<?php

/*  Имя файла:        addressForm.inc
 *  Описание:         Сценарий для отображения формы.
 */
  echo "<html>
        <head><title>Адрес покупателя</title></head>
        <body>";
  echo "<p align='center'>
        <form action='validateForm.php' method='POST'>
        <table width='95%' border='0' cellspacing='0'
              cellpadding='2'>\n";
  foreach($labels as $field=>$value)
  {
    if(isset($_POST[$field]))                          #13
    {
       $value = $_POST[$field];
    }
    else
    {
       $value = "";
    }
    echo "<tr><td align='right'>{$labels[$field]}</br></td>
          <td><input type='text' name='$field' size='65'
                  maxlength='65'
                  value='$value'> </td> </tr>";
  }
  echo " </table>
         <div align='center'>
           <p><input type='Submit' name='Submit'
```

```
            value='Отправить данные'></p></div>
        </form>";
?>
</body></html>
```

Обратите внимание на блок `if-else`, который начинается в строке 13. В нем устанавливаются значения, отображаемые в полях формы. При первом отображении формы массив `$_POST` еще не существует, поэтому переменной `$value` присваивается пустое значение. В противном случае этой переменной присваивается значение, которое ранее было введено пользователем в соответствующем поле формы. Это значение используется при повторном выводе формы.

Обратите внимание на строку, в которой создается кнопка `Submit`:

```
<p><input type='Submit' name='Submit'
        value='Отправить данные'></p></div>
```

При передаче данных сценарию значение `value` будет добавлено в массив `$_POST`. В форме можно использовать две таких кнопки, с одинаковыми именами, но разными значениями, и выполнять действия в зависимости от того, на какой кнопке щелкнул пользователь. В этом случае пригодится оператор `if` следующего вида:

```
if($_POST['Submit'] == "Отправить данные").
```

На рис. 10.4 показана форма, в которой пользователь случайно ввел свое имя во второе поле и указал несуществующий почтовый индекс. При этом на Web-странице выведены два сообщения об ошибке. Первая ошибка связана с оставшимся пустым полем для ввода имени, а вторая свидетельствует о вводе неправильного почтового индекса.

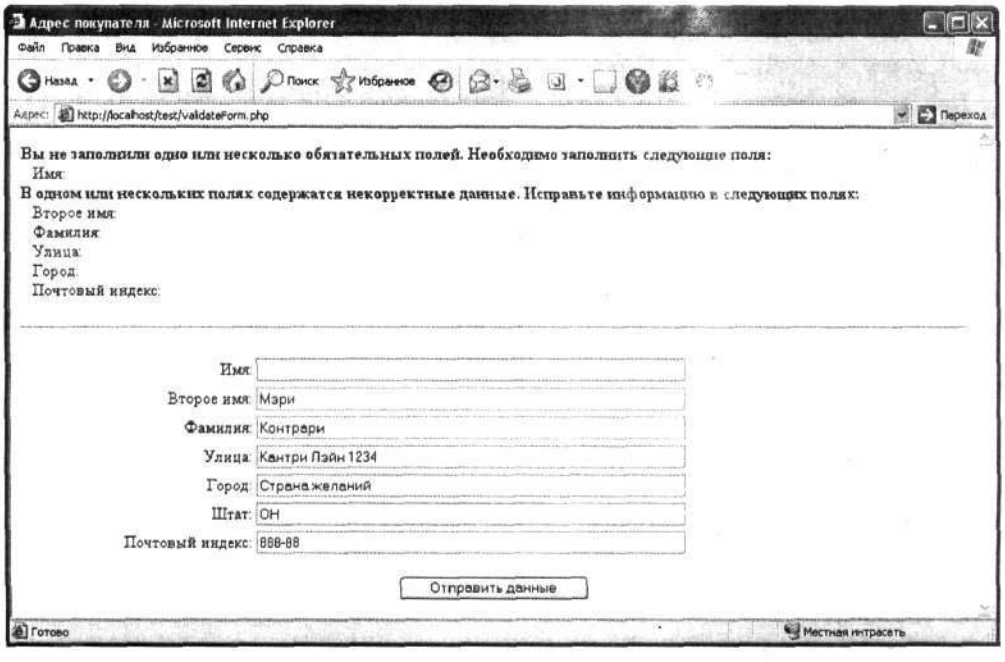


Рис. 10.4. Результат обработки формы, в которой осталось не заполненным одно обязательное поле, а в другое поле были введены некорректные данные

Очистка данных

Тщательной проверки данных зачастую бывает вполне достаточно, чтобы предотвратить ввод нежелательной (опасной) информации. Можно проверить формат введенных данных (например, почтового индекса или телефонного номера) или ограничить список допустимых символов буквами и числами. Однако иногда необходимо предоставить возможность ввода всех символов из числа возможных, включая математические символы или HTML-код. Например, сценарий для доски объявлений должен разрешать пользователям оставлять сообщения любого вида.

Если жестких ограничений на пользовательские данные установить нельзя, злонамеренные взломщики смогут ввести в поля формы опасную информацию. Например, с использованием дескриптора `<script>` можно передать на сервер исполняемый сценарий. От того, как эта информация будет обработана, зависит, сможет ли злоумышленник запустить свой сценарий или загрузить его на компьютеры других посетителей Web-узла.

Для удаления опасной информации в языке PHP предназначены две полезных функции.

- ✓ `strip_tags()`. Эта функция позволяет удалить из текста все дескрипторы, хотя можно указать, какие из них следует оставить.
- ✓ `htmlspecialchars()`. Эта функция позволяет преобразовать некоторые специальные символы HTML в другой формат. При этом они будут отображаться без учета их специального назначения. Например:
  - < преобразовывается в `<`;
  - > преобразовывается в `>`;
  - & преобразовывается в `&`.

Удаление из полученной информации всех дескрипторов позволит существенно повысить безопасность. Например, для удаления всех дескрипторов можно воспользоваться выражением

```
$last_name = strip_tags($last_name);
```

При вызове функции `strip_tags()` интерпретатор PHP выполнит поиск в строке `$last_name` символа начала дескриптора < и удалит все символы, вплоть до символа > или конца строки. При этом можно указать, какие дескрипторы удалять не следует:

```
$last_name = strip_tags($last_name, "<b><i>");
```

В этом случае в строке `$last_name` будут удалены все дескрипторы, кроме <b> и <i>.

Однако в некоторых случаях символы < и > не нужно связывать с дескрипторами, например при их использовании в качестве символов математических операций. Их можно преобразовать в такие символы, которые не будут интерпретироваться как дескрипторы HTML. Для этого необходимо воспользоваться следующей функцией `htmlspecialchars()`:

```
$message = htmlspecialchars($message);
```

Теперь нужно разобраться, чем же отличаются друг от друга функции `strip_tags()` и `htmlspecialchars()`. Пусть имеется переменная `$message`, в которой содержится следующее значение:

```
Используйте дескрипторы <?php ?> для размещения операторов PHP.
```

Для удаления ненужных дескрипторов воспользуемся функцией `strip_tags()`:

```
$message = strip_tags($message);
echo $message;
```

В результате получим:

```
Используйте дескрипторы  для размещения операторов PHP.
```

Однако для достижения этой же цели можно воспользоваться и функцией `htmlspecialchars()`:

```
$message = htmlspecialchars($message);
echo $message;
```

В этом случае получим следующий результат:

```
Используйте дескрипторы &lt;?php ?&gt; для размещения операторов PHP.
```

Причем в браузере эта строка отобразится следующим образом:

```
Используйте дескрипторы <?php ?> для размещения операторов PHP.
```

т.е. исходная строка будет отображаться без ошибок. Это связано с тем, что символы < и > не интерпретируются как относящиеся к дескрипторам, и соответственно код PHP не выполняется.

Для удаления ненужной информации можно воспользоваться также функцией `trim()`. Зачастую в начале или в конце строки пользователи случайно вводят пробелы. Позже такие данные могут быть неправильно обработаны, например при сравнении с заданным шаблоном. Для удаления ненужных пробелов как раз и предназначена функция `trim()`, которую можно вызвать следующим образом:

```
$last_name = trim($last_name);
```

Глава 11

Другие виды Web-приложений

В этой главе...

- Передача информации между страницами
- Использование данных cookie
- Скрытые поля HTML-форм
- Сеансы PHP
- Использование JavaScript в сценариях PHP

Самые простые Web-приложения получают информацию от пользователей через HTML-формы, а затем используют ее для вывода, хранения или проверки условных выражений. (Несколько примеров таких приложений рассматривалось в главе 10.) Однако Web-приложения могут быть и гораздо более сложными. Так, торговая тележка должна "уметь" собирать и отображать различную информацию, следить за заказом клиента, вычислять сумму покупки, налоги и стоимость доставки, проверять состояние кредитной карточки, а также выполнять множество других действий. Такое приложение должно состоять из нескольких сценариев, в которых используются одни и те же данные. Кроме того, разработчики Web-приложения должны обеспечить возможность приема от пользователей целых файлов, а не только данные формы.

В этой главе рассматриваются фундаментальные вопросы, связанные с разработкой сложных Web-приложений.

Независимость Web-страниц

Web-страницы являются *независимыми* друг от друга. После того как пользователь щелкнет на ссылке, Web-сервер передает браузеру новую страницу. При этом браузеру о предыдущей странице уже ничего не известно. Она может оказаться даже первой Web-страницей в мировой истории. При использовании статических Web-страниц, которые пользователь может просто просматривать в браузере, подобная независимость нисколько не мешает. Однако в большинстве динамических Web-приложений информацией нужно обмениваться между страницами. Например, имя пользователя можно ввести на одной странице, а затем вывести его на другой.

В следующих нескольких разделах рассматриваются различные способы передачи данных между страницами.

Перемещение между страницами Web-узла

Большинство Web-узлов состоит из нескольких страниц. Так, на статических Web-узлах переход от одной страницы к другой осуществляется с помощью ссылок (иногда реализованных в виде кнопки), на которых щелкают пользователи. Таким образом, можно путешествовать по узлу и искать требуемую информацию. В динамических Web-приложениях для перехода между страницами можно использовать и другие методы, основные из которых приведены ниже.

- ✓ **Ссылки**. На новую страницу пользователь может перейти, щелкнув на ссылке.
- ✓ **Использование форм**. Формы позволяют перемещаться между страницами щелчком на кнопке `Submit`.
- ✓ **Перемещение пользователей.** В языке PHP имеется функция `header()`, с использованием которой пользователь может перейти на новую страницу без выполнения каких-либо действий.

Все эти методы более подробно рассматриваются в следующих разделах.

Вывод ссылок

В языке PHP можно воспользоваться функцией `echo` и просто вывести ссылку HTML (точно так же, как и другой фрагмент HTML). После этого ее можно использовать для перехода на новую страницу:

```
echo "<a href='newpage.php';>Новая страница</a>";
```

Использование форм

Для перехода на новую страницу можно использовать и формы (см. главу 10). Дескриптор `<form>` позволяет задать сценарий, который будет обрабатывать информацию, введенную пользователем. После щелчка на кнопке `Submit` этому сценарию будут переданы данные формы и отобразится новая Web-страница.

Безусловно, для перехода на новую страницу совсем необязательно использовать форму для сбора информации. Можно воспользоваться пустой формой с кнопкой, после щелчка на которой пользователь переместится на новую страницу. Например, можно создать кнопку `Cancel` (Отменить) или `Next` (Следующая) с использованием дескрипторов `<form>` и `<input>`. Тогда после щелчка на этой кнопке будет выполнен переход на страницу, адрес которой указан в атрибуте `action` дескриптора `<form>`.

Функции, которые следует вызывать первыми

В PHP существуют функции, которыми можно воспользоваться только до начала генерации всего остального содержимого Web-страницы. К таким функциям относятся `header()`, `setcookie()` и `session()`. Если их вызвать после завершения формирования содержимого страницы, будет выведено следующее сообщение:

```
Warning: Cannot modify header information - headers already sent by
(output started at /test.php:2) in /test.php on line 3

(Предупреждение: Нельзя изменить заголовок — заголовки уже
отправлены (/test.php:2) в файле /test.php в строке 3)
```

Как можно заметить, в этом сообщении указано имя файла и строка, в которой была сгенерирована ошибка. Вполне возможно, что сообщение об ошибке выведено не будет, а страница вообще не появится в окне браузера. (Это зависит от используемого режима вывода ошибок в PHP; см. главу 4.) Обработка следующего фрагмента кода приведет к возникновению ошибки, поскольку вывод заголовка расположен после раздела HTML:

```
<html>
<head><title>Тестирование функции header()</title></head>
<body>
<?php
   header("Location: http://janetscompany.com");
?>
</body></html>
```

Как можно заметить, перед вызовом функции `header()` расположено несколько строк кода HTML. Следующий фрагмент кода является работоспособным, однако не имеет особого смысла. Поскольку код

HTML расположен после инструкций PHP, предназначенных для вывода заголовка, то перед его отображением будет выполнен переход на другую страницу.

```
<?php
   header("Location: http://janetscompany.com");
?>
<html>
<head><title>Тестирование функции header()</title></head>
<body>
</body>
</html>
```

Следующий фрагмент приведет к генерации сообщения об ошибке:

```
 <?php
   header("Location: http://company.com");
?>
<html>
<head><title>Тестирование функции header()</title></head>
<body>
</body></html>
```

Причину вывода сообщения об ошибке обнаружить не так то просто. Однако при внимательном анализе перед открывающим дескриптором `<?php` можно заметить символ пробела. Этот символ также отображается на Web-странице, и, следовательно, функция `header()` вызывается после некоторого выражения (хотя оно и является пробелом). Это типичная ошибка начинающих разработчиков, на обнаружение которой можно потратить достаточно много времени.

Перемещение пользователей

Язык PHP предоставляет метод, с использованием которого можно перейти на другую страницу вообще без щелчка на ссылке или кнопке. Это позволяет осуществить функция `header()` со следующим синтаксисом:

```
header("Location: URL");
```

С помощью этой функции Web-серверу можно передать сообщение вида `Location: URL`. В ответ браузеру будет передан файл, расположенный по указанному *URL*-адресу. Ниже приведено несколько примеров корректного использования функции `header()`.

```
header("Location: newpage.php");
header("Location: http://company.com/catalog/catalog.php");
```

Как уже отмечалось выше, функция `header()` имеет одно важное ограничение: она должна вызываться *перед* генерацией какого-либо содержимого страницы. Нельзя сначала поместить код HTML, а после него — где-то в середине сценария — вызов функции `header()`. Эти же ограничения касаются и функций `setcookie()` и `session()` (см. врезку "Функции, которые следует вызывать первыми").

Однако, несмотря на все эти ограничения, функция `header()` оказывается чрезвычайно полезной. Она предоставляет единственный способ перехода на новую страницу без вмешательства пользователя. Другими словами, функцию `header()` можно использовать для вывода различных страниц для разных пользователей. Рассмотрим такой пример:

```
<?php
   if ($typeAcct == "admin")
   {
      header("Location: AdminPage.php");
   }
   else
   {
```

```
        header("Location: SiteHomePage.php");
    }
?>
```

В приведенном фрагменте для администратора выводится специальная страница администрирования, а для других пользователей — главная страница Web-приложения. Перед функцией `header()` можно использовать сколько угодно функций PHP, которые не связаны с формированием HTML-кода Web-страницы. Однако размещение перед вызовом функции `header()` какого-либо HTML-кода приведет к ошибке.

Перемещение данных между страницами

При перемещении пользователя между страницами Web-приложения может возникнуть ситуация, когда информация, введенная на одной странице, может понадобиться на другой странице. Для реализации этого требования на языке PHP можно воспользоваться несколькими способами.

- ✓ **Добавление информации к URL-адресам**. В этом случае требуемую информацию можно добавить к URL-адресу новой страницы. Этот подход оказывается наиболее эффективным только при передаче небольшого количества данных.
- ✓ **Хранение информации с помощью данных cookie**. Для передачи информации между сценариями можно воспользоваться данными *cookie*, которые представляют собой множество пар "переменная–значение" (*имя_переменной=значение*). При этом информация сохраняется в клиентской части приложения, т.е. на локальном компьютере пользователя. После того как данные cookie были сохранены, ими можно воспользоваться с любой Web-страницы. Существенный недостаток такого подхода заключается в возможности отключения режима использования данных cookie в клиентском браузере.
- ✓ **Передача данных с использованием HTML-форм**. Для передачи информации между различными сценариями Web-приложения могут использоваться HTML-формы. После щелчка на кнопке `Submit` данные из полей формы будут переданы серверному сценарию. Такой подход оказывается весьма полезным, если необходимо взаимодействовать с пользователями.
- ✓ **Использование сеансов**. Начиная с версии 4, язык PHP позволяет реализовать поддержку пользовательских сеансов и всю необходимую информацию хранить на сервере. Доступ к этой информации можно получить из любого сценария (Web-страницы). Этот подход оказывается особенно эффективным при просмотре большого количества Web-страниц.

В следующих разделах каждый из упомянутых подходов рассматривается более подробно.

Добавление информации к URL-адресу

Один из наиболее простых способов передачи информации между сценариями заключается в добавлении необходимых данных к URL-адресу. Для этого необходимо воспользоваться следующим синтаксисом:

переменная=значение

В этом случае *переменная* — это имя переменной (без использования символа `$`), а *значение* — ее значение. Пару *переменная=значение* можно добавить к адресу URL после символа `?`. Например:

- ✓ `<a href= "nextpage.php?age=14">перейти на следующую страницу</a>`
- ✓ `header("Location: nextpage.php?age=14");`
- ✓ `<form action="nextpage.php?age=14" method="POST">`

Во всех трех примерах при переходе на новую страницу `nextpage.php` передается переменная `$age` со значением `14`. Строка `age=14` добавляется в конец URL-адреса сразу после знака `?`.

При передаче в адресе URL нескольких пар "переменная–значение" они отделяются друг от друга амперсантом (&). Например:

```
<form action="nextpage.php?state=CA&city=Молл" method="POST">
```

Доступ ко всем переменным, передаваемым в строке URL, можно получить через встроенный массив `$_GET`. В сценарии `nextpage.php` из предыдущего примера для отображения переданных значений можно воспользоваться выражением

```
echo "{$_GET['city']}, {$_GET['state']};
```

и получить следующий результат:

```
Молл, CA
```

Эти же данные можно извлечь и из массива `$_REQUEST`. Например:

```
echo "{$_REQUEST['city']}, {$_REQUEST['state']};
```

С использованием URL удобно передавать лишь небольшое количество информации. Кроме того, такой подход обладает и рядом других недостатков (в том числе связанных и с обеспечением безопасности), среди которых следует выделить следующие.

- ✓ **Общедоступность**. Адрес URL всегда отображается в адресной строке окна браузера. Естественно, что передаваемая информация также будет видна пользователю. Например, если требуется передать пароль, вряд ли стоит использовать для этого URL-адрес.
- ✓ **В URL-адресе можно передавать любую информацию**. Рассмотрим ситуацию, когда пользователь зашел на Web-узел с ограниченным доступом. В этом случае к URL-адресу добавляется строка `auth=yes` (свидетельствующая о том, что аутентификация завершилась успешно). При этом проверка равенства `$_GET['auth']=yes` будет выполняться на каждой странице. Лишь при истинности этого условия будет осуществляться переход на новую страницу. Однако любой пользователь может ввести в окне браузера URL-адрес `http://www.`*имя-узла*`.com/page.php?auth=yes` и без ввода каких-либо данных аутентификации получить доступ к страницам с ограниченным доступом.
- ✓ **Адрес URL можно сохранить в браузере**. Пользователь может сохранить в браузере закладку с URL-адресом, а вместе с ним и добавленную к нему информацию.
- ✓ **Ограниченная длина адреса URL**. Для разных браузеров и их версий максимальная длина URL-адреса всегда конечна (хотя и различается). Если необходимо передать большое количество информации, то объективные ограничения, наложенные на длину адреса URL, могут не позволить этого сделать.

Передача информации с помощью данных cookie

Информацию, которую необходимо передавать от страницы к странице, можно сохранить как данные *cookie*. Эти данные представляют собой набор пар "переменная–значение" (*переменная=значение*) и очень напоминают фрагменты, добавляемые к URL-адресу. Особенность данных cookie заключается в том, что они сохраняются браузером на пользовательском компьютере и позже могут использоваться сценариями.

На первый взгляд, данные cookie позволяют решить все проблемы, связанные с обменом данных между отдельными компонентами (страницами) Web-приложения. Их необходимо хранить в клиентском браузере и использовать по мере необходимости. Сохранив данные cookie один раз, их можно использовать при последующих посещениях данной Web-страницы. Однако, как легко догадаться, не все так просто. Дело в том, что режим использования (или неиспользования) данных cookie в клиентском браузере полностью определяется пользователем. Другими словами, в любой момент он их может удалить или вообще от них отказаться. Большинство посетителей Web-узлов зачастую именно так и поступают. Далеко не все из них рады тому, что кто-то сохраняет у них на компьютере свою информацию. Подобная реакция вполне объяснима. Приведенные выше рассуждения и факты существенно сужают область применения механизма cookie. Если основу функционирования Web-приложения составляют данные cookie и пользователь отключил режим их использования, то оно будет работать некорректно.

Изначально данные cookie предназначались для хранения небольшого количества данных в течение короткого промежутка времени. Если для данных cookie не указать время их жизни, то они будут удалены сразу же после завершения работы с Web-приложением. Несмотря на то что данные cookies достаточно полезны, следует хорошенько подумать, прежде чем принять решение о необходимости их использования. И вот по каким причинам.

- ✓ **Возможность отключения режима использования данных cookie**. До тех пор пока нет полной уверенности в том, что *все* пользователи разрешают использование данных cookie (или согласны установить этот режим работы браузера), возможно возникновение проблем и нарушение корректного функционирования Web-приложения.
- ✓ **Язык PHP предоставляет гораздо более эффективные средства**. Начиная с версии 4 языка PHP, механизм поддержки сеансов обеспечивает возможность хранения данных на протяжении всего *сеанса* (session) работы пользователя — другими словами, до тех пор, пока он остается на Web-узле. При этом информации хранится на сервере, так что пользователь на это никак не может повлиять.
- ✓ **Информацию можно хранить в базе данных**. Если имеется доступ к базе данных, которая позволяет как хранить, так и извлекать информацию, то эта возможность гораздо лучше, чем использование данных cookie. Пользователи не смогут удалить информацию из базы данных.

Манипулирование данными cookie

Для того чтобы сохранить данные cookie, следует воспользоваться функцией `setcookie()`, которая имеет следующий синтаксис:

```
setcookie("переменная", "значение");
```

Параметр *переменная* представляет собой имя переменной (без символа $), а *значение* — ее значение. Это выражение будет сохраняться до тех пор, пока пользователь не покинет Web-узел. Рассмотрим следующий пример, в котором с помощью данных cookie на клиентском узле сохраняется выражение `state=CA`:

```
setcookie("state", "CA");
```

К данным cookie можно обратиться с помощью встроенного массива $_COOKIE. Например, можно вывести выражение

```
echo "Вы живете в штате {$_COOKIE['state']} ";
```

В результате будут получены следующие данные:

```
Вы живете в штате CA
```

Следует отметить, что данные cookie недоступны на той странице, на которой они были установлены. Для получения к ним доступа нужно заново отобразить текущую страницу или перейти на новую.

Установка времени жизни данных cookie

В случае необходимости можно задать время жизни данных cookie. Для этого предназначена функция `setcookie()` с синтаксисом

```
setcookie("переменная", "значение", время-жизни);
```

Параметр *время-жизни* определяет время, по истечении которого данные cookie будут удалены. Обычно это значение вычисляется с помощью функции `time()` или `mktime()`.

- ✓ `time()`. Эта функция возвращает текущее значение даты и времени. Для задания времени жизни данных cookie к результату функции `time()` следует прибавить соответствующее значение в секундах. Например:

```
setcookie("state", "CA", time()+3600); #удаление данных через 1 час
setcookie("Name", $Name, time()+(3*86400)) #удаление данных через 3 дня
```

- ✓ `mktime()`. Эта функция позволяет преобразовать указанное значение даты и времени в форму, которая понятна компьютеру. Для этого требуемую дату необходимо представить в следующем виде: часы, минуты, секунды, месяц, день и год. Если какое-то значение будет пропущено, по умолчанию используется значение, соответствующее текущему моменту времени. Ниже приведено несколько примеров установки времени жизни данных cookie с помощью функции `mktime()`.

```
setcookie("state", "CA", mktime(3,0,0,4,1,2003));
#удаление cookie 1 апреля 2003 года в 03:00
setcookie("state", "CA", mktime(13,0,0,,,)); #удаление данных сегодня в 13:00
```

Для удаления данных cookie соответствующей переменной необходимо присвоить пустое значение. Это можно осуществить двумя способами:

```
setcookie("имя");
setcookie("имя", "");
```

Однако функция `setcookie()` имеет одно существенное ограничение. Она должна вызываться *до* генерации какой-либо части Web-страницы. Невозможно задать данные cookie где-то в середине сценария после того, как некоторая часть страницы уже была сформирована. Более подробную информацию по этому вопросу см. выше, во врезке "Функции, которые следует вызывать первыми".

Передача информации с помощью HTML-форм

Один из стандартных способов передачи информации между страницами заключается в использовании форм. Дело в том, что HTML-формы могут содержать кнопку `Submit`. После щелчка на этой кнопке введенная в поля формы информация передается сценарию, URL-

адрес которого задан в дескрипторе <form>. Для описания формы используется следующий общий синтаксис:

```
<form action="processform.php" method="POST">
    поля формы
<input type="submit" value="строка">
</form>
```

Обычно формы используются для получения информации от пользователей и передачи ее следующей странице (сценарию). (Более подробно об этом см. в главе 10). Однако формы могут оказаться полезными и при передаче данных другого вида.

Поля, которые не отображаются на Web-странице, но используются при передаче данных, называются *скрытыми* (hidden fields). Их можно использовать как вместе с обычными полями, так и отдельно от них. После щелчка на соответствующей кнопке значения скрытых полей передаются следующей странице. В следующем примере после щелчка на кнопке Следующая страница регистрационные данные пользователя будут переданы странице nextpage.php:

```
<?php
  $acct = "admin";
  echo "<form action='nextpage.php' method='POST'>
        <input type='hidden' name='acct' value='$acct'>
        <input type='submit' value='Следующая страница'>
        </form>\n";
?>
```

На Web-странице отображается только кнопка Следующая страница, причем пользователю не нужно вводить никакой информации. После щелчка на этой кнопке вызывается сценарий nextpage.php, а значение скрытого поля сохраняется в элементе массива $_POST['acct']. Такой подход позволяет передавать требуемую информацию между страницами Web-узла. Например, приведенный фрагмент можно включить в сценарий, предназначенный для отображения различных видов товаров. Тогда после щелчка на кнопке Следующая страница регистрационные данные пользователя будут переданы новой странице и, таким образом, окажутся доступными для соответствующего сценария.

Использование сеансов PHP

Сеанс (session) можно определить как промежуток времени, в течение которого пользователь взаимодействует с Web-приложением. В течение одного сеанса посетители Web-узла обычно просматривают большое количество страниц, и очень часто требуется иметь доступ к одной и той же информации на каждой из них. Начиная с версии 4.0, язык PHP позволяет реализовать такой подход, основанный на механизме поддержки сеансов.

Как работают сеансы PHP

Язык PHP предоставляет средства для создания сеансов и хранения связанных с ними переменных. После создания сеанса значения установленных переменных становятся доступными на любой странице (или в сценарии) Web-приложения. При открытии сеанса интерпретатором PHP выполняются следующие действия.

1. Сеансу присваивается *уникальный идентификатор* (session ID number). Обычно идентификатор представляет собой очень длинную строку, которая является уникальной для конкретного пользователя (например, 523afa15f4a8e05e95241481c0cbc71e). Такую строку подобрать практически невозможно. В окружении PHP это значение хранится в переменной $PHPSESSID.
2. Все необходимые переменные сеанса сохраняются в файле, который размещается на сервере. Причем имя файла совпадает с идентификатором сеанса. Каталог, в котором

хранятся файлы сеансов, определяется директивой `session.save_path` файла `php.ini`. Причем этот каталог должен быть создан предварительно.

3. Интерпретатор PHP передает идентификатор сеанса каждой странице. Если у пользователя активизирован режим использования данных cookie, то передача требуемых данных между сервером и клиентским браузером осуществляется с использованием этого механизма. В противном случае работа модуля PHP определяется значением директивы `trans-sid` конфигурационного файла `php.ini`. (Более подробно этот механизм описывается ниже, в разделе "Поддержка сеансов при отключенном режиме использования данных cookie".)
4. Файл с данными сеанса доступен для всех сценариев в рамках этого сеанса. При этом переменные хранятся также в массиве `$_SESSION`.

В версиях PHP 4.1.2 и более ранних директива `trans-sid` доступна только в том случае, если модуль PHP скомпилирован с параметром `--enable-trans-sid`.

Открытие и закрытие сеансов

Сеанс нужно открывать в начале каждой страницы, используя функцию `session_start()`. Эта функция сначала проверяет, был ли ранее создан сеанс. Если да, то выполняется установка переменных сеанса. В противном случае создается новый сеанс с уникальным идентификатором.

Поскольку механизм поддержки сеансов во многом зависит от режима поддержки данных cookie, то на использование функции `session_start()` накладываются те же ограничения. Другими словами, ее нужно вызывать *до* вывода какой-либо информации на Web-странице. Более подробно этот вопрос рассматривался во врезке "Функции, которые следует вызывать первыми".

Интерпретатор PHP может автоматически добавлять функцию `session_start()` в начале каждой серверной страницы. Для этого необходимо внести соответствующие изменения в конфигурационный файл `php.ini`, а именно: задать для директивы `session.auto_start` значение `1`. Для того чтобы внесенные изменения вступили в силу, нужно перезапустить Web-сервер. После этого добавлять функцию `session_start()` в начало каждой страницы уже не понадобится.

Зачастую необходимо, чтобы Web-узлы посещали только пользователи с корректным идентификатором и паролем. Для каждого из них создается свой собственный сеанс. Однако после завершения работы необходимо обеспечить закрытие текущего сеанса работы. Для этого предназначена функция `session_destroy()`.

Эта функция удаляет всю информацию, хранящуюся в переменных сеанса. С этого момента идентификатор сеанса больше не будет передаваться между страницами. Однако следует заметить, что на текущей странице эти переменные по-прежнему будут доступны. Для их полного удаления предназначена функция

```
unset($переменная1, $переменная2, ...);
```

Использование сеансовых переменных

Для того чтобы сохранить переменную в рамках сеанса и обеспечить ее доступность на других Web-страницах, нужно сохранить ее в массиве `$_SESSION` следующим образом:

```
$_SESSION['имя_переменной'] = "Джон Смит";
```

При открытии сеанса на другой странице к предварительно сохраненным значениям можно обратиться через массив `$_SESSION`.

Для удаления переменной в любой момент можно воспользоваться функцией `unset()`:

```
unset($_SESSION['имя_переменной']);
```

В следующих двух сценариях показано, как с помощью механизма сеансов передавать информацию между страницами. Сценарий из листинга 11.1 отображает первую страницу, а сценарий из листинга 10.2 — вторую.

Листинг 11.1. Создание сеанса

```
<?php
 /* Имя сценария:          sessionTest1.php
  * Описание:              создает сеанс и сохраняет связанные
  *                        с ним переменные.
  */
  session_start();
  $_SESSION['session_var'] = "тестирование";
?>
<html>
<head><title>Тестовая страница 1 сеанса</title></head>
<body>
  <p>Тестирование свойств сеансов.
  <form action="sessionTest2.php" method="POST">
  <input type="text" name="form_var" value="тестирование">
  <input type="submit" value="Перейти на следующую страницу">
  </form>
</body>
</html>
```

В приведенном сценарии с помощью функции `session_start()` создается сеанс с одной переменной `session_var`. На странице содержится также форма с одним текстовым полем и кнопкой `Перейти на следующую страницу`, после щелчка на которой вызывается сценарий `sessionTest2.php`.

Листинг 11.2. Вторая страница сеанса

```
<?php
 /* Имя сценария:          sessionTest2.php
  * Описание:              получение данных текущего сеанса.
  */
  session_start();
?>
<html>
<head><title>Тестовая страница 2 сеанса</title></head>
<body>
<?php
  $session_var = $_SESSION['session_var'];
  $form_var = $_POST['form_var'];
  echo "session_var = $session_var<br>\n";
  echo "form_var = $form_var<br>\n";
?>
</body>
</html>
```

Этот сценарий отображает значения переменных, переданных из предыдущего сценария (`sessionTest1.php`).

Если пользователь загрузил в браузер страницу `sessionTest1.php` и щелкнул на кнопке `Перейти на следующую страницу`, то будет получен следующий результат:

```
session_var = тестирование
form_var = тестирование
```

Как можно увидеть, значения обеих переменных, `session_var` (связанная с сеансом) и `form_var` (из поля формы), можно получить из встроенных массивов `$_SESSION` и `$_POST` соответственно.

Поддержка сеансов при отключенном режиме использования данных cookie

Зачастую пользователи отключают в браузере режим использования данных cookie. При использовании механизма поддержки сеансов модуль PHP проверяет, допускает ли клиентский браузер использование данных cookie. Если соответствующий режим включен, выполняются следующие действия.

- ✓ Переменной `$PHPSESSID` присваивается значение идентификатора сеанса.
- ✓ Данные cookie используются для передачи `$PHPSESSID` между сценариями или страницами.

Если клиентский браузер запрещает использовать данные cookie, то модуль PHP функционирует по-другому.

- ✓ Создается константа с именем `SID`, в которой содержится пара "переменная–значение" вида `PHPSESSID=`*`идентификатор-сеанса`*.
- ✓ После этого использование идентификатора сеанса определяется значением директивы `trans-sid` в конфигурационном файле `php.ini`. Если эта директива включена, то идентификатор сеанса передается между страницами или сценариями, в противном случае — нет.

По умолчанию директива `trans-sid` отключена. Для ее включения следует внести соответствующие изменения в строку `session.use_trans_sid=` файла `php.ini`. Если в этой строке содержится значение `0`, то директива `trans-sid` отключена, если `1` — включена. После внесения изменений нужно перезапустить Web-сервер.

Включение директивы `trans-sid` имеет как преимущества, так и недостатки.

- ✓ **Преимущества**. Механизм поддержки сеансов функционирует корректно даже в том случае, если пользователь отключил режим использования данных cookie, т.е. при использовании директивы `trans-sid` можно создавать сценарии, не зависящие от настроек клиентского браузера.
- ✓ **Недостатки**. Идентификатор сеанса передается в URL-адресе и, как следствие, отображается в адресной строке браузера. С точки зрения обеспечения безопасности этого следует избегать.

Использование сеансов с включенной директивой `trans-sid`

Если директива `trans-sid` включена и в клиентском браузере отключен режим использования данных cookie, то идентификатор сеанса автоматически пересылается с помощью URL или скрытых полей формы. При переходе на новую страницу с помощью ссылки, функции `header()` или формы с методом `GET` идентификатор сеанса размещается в URL-адресе. Если же новая страница отображается в результате заполнения полей формы, в которой для передачи данных на сервер указан метод `POST`, то идентификатор сеанса размещается в скрытых полях. При этом в обоих случаях идентификатор сеанса хранится в переменной `$PHPSESSID`.

Идентификатор сеанса добавляется только к URL-адресам, которые находятся в рамках одного Web-узла. Если же в URL содержится имя сервера, идентификатор сеанса не добавляется. Например, к ссылке

```
<a href="newpage.php">
```

интерпретатор PHP добавит идентификатор сеанса. Однако при обработке ссылки

```
<a href="HTTP://www.janetscompany.com/newpage.php">
```

идентификатор сеанса добавляться уже *не* будет.

Использование сеансов с отключенной директивой `trans-sid`

Если в директиве `trans-sid` содержится значение `0` и в клиентском браузере отключен режим использования данных cookie, то модуль PHP не будет пересылать идентификатор сеанса на следующую страницу или в сценарий. Разработчик должен сделать это самостоятельно.

К счастью, в окружении PHP имеется соответствующая константа `SID`, значение которой можно пересылать на следующую страницу вручную. В этой константе содержится пара "*имя-переменной=значение*", которую к адресу URL можно добавить следующим образом:

```
<a href="nextpage.php?<?php echo SID?>" > следующая страница </a>;
```

В приведенной ссылке после имени файла стоит вопросительный знак и константа `SID` с идентификатором сеанса. Выражение `echo SID` позволяет получить примерно следующий результат:

```
PHPSESSID=f8544042bf883ca93b5e2c5dc6794755
```

Таким образом, адрес URL будет выглядеть примерно так:

```
nextpage.php?PHPSESSID=f8544042bf883ca93b5e2c5dc6794755
```

По определенным причинам (которые уже рассматривались выше) появление идентификатора сеанса в окне браузера может привести к самым нежелательным последствиям. Поэтому можно воспользоваться еще одним способом передачи идентификатора сеансов: с помощью скрытых полей формы и метода `POST`. Сначала нужно получить идентификатор сеанса, а затем переслать его в скрытом поле. Рассмотрим пример, в котором иллюстрируется данный подход:

```
<?php
  $PHPSESSID = session_id();
  echo "<form action='nextpage.php' method='POST'>
        <input type='hidden' name='PHPSESSID'
              value='$PHPSESSID'>
        <input type='submit' value='Следующая страница'>
        </form>";
?>
```

В приведенном фрагменте выполняется следующее.

1. Переменной `$PHPSESSID` присваивается значение идентификатора сеанса, возвращаемое функцией `session_id()`.
2. Затем с использованием скрытого поля формы значение `$PHPSESSID` передается на следующую страницу.

Таким образом, переменная `$PHPSESSID` автоматически будет доступна на следующей странице `nextpage.php`.

Создание сеансов для групп пользователей

Сеансы прекрасно подходят для обеспечения ограниченного доступа к ресурсам Web-узла и ввода регистрационных данных. На таких узлах обычно содержится большое количество страниц. Однако заставлять пользователя постоянно вводить регистрационные данные нежелательно. К счастью, сеансы PHP позволяют отслеживать действия зарегистрированного пользователя и по необходимости предотвратить возможность доступа к той или иной странице тем пользователям, которые ранее не были зарегистрированы. С помощью механизма поддержки сеансов можно выполнить следующие действия.

1. Отобразить страницу регистрации для ввода имени пользователя и пароля.
2. После успешной регистрации создать и сохранить соответствующие переменные сеанса.
3. При переходе пользователя на следующую страницу на основе переменных сеанса проверить факт успешно пройденной процедуры регистрации.
4. Если пользователь зарегистрировался, отобразить Web-страницу.
5. В противном случае предложить пользователю ввести регистрационное имя и пароль.

Для проверки правильности введенных имени и пароля пользователя в начало каждой страницы необходимо добавить следующий фрагмент кода:

```
<?php
session_start()
if (@$_SESSION['login'] != "go")
{
      header("Location: loginPage.php");
      exit();
}
?>
```

В этом фрагменте кода проверяется переменная сеанса `login` (которая была инициализирована при вводе) на равенство значению `go`. Если данное условие не выполняется, это означает, что пользователь не был зарегистрирован, и в браузере отображается страница для ввода регистрационных данных. В противном случае сценарий продолжает выполняться.

Загрузка файлов

Зачастую необходимо обеспечить, чтобы пользователи могли удаленно загружать файлы на Web-узел. Например, на Web-узел вакансий необходимо загружать резюме, а в электронный фотоальбом — фотографии.

Использование форм для загрузки файлов

Для обеспечения загрузки файлов на Web-узел можно разработать специальную HTML-форму, которая может иметь следующий вид:

```
<form enctype="multipart/form-data"
          action="processfile.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="30000">
  <input type="file" name="user_file">
  <input type="submit" value="Загрузить файл">
</form>
```

Рассмотрим элементы формы более подробно.

- ✓ **Атрибут** `enctype` **дескриптора** `<form>`. Для обеспечения корректной загрузки файла этот атрибут должен иметь значение `multipart/form-data`.
- ✓ **Скрытое поле, в котором указан максимальный размер загружаемого файла** `MAX_FILE_SIZE` **(в байтах)**. Пользователь не сможет загрузить файл, размер которого превышает `MAX_FILE_SIZE`. Однако перед использованием значения `MAX_FILE_SIZE` в конфигурационный файл `php.ini` нужно внести следующие изменения.
  - `upload_max_filesize`. Значение `MAX_FILE_SIZE` не может превышать значение `upload_max_filesize`. Если необходимо загрузить файлы большего размера, следует изменить значение `upload_max_filesize` в файле `php.ini`. По умолчанию оно составляет 2 Мбайт.
  - `post_max_size`. Эта переменная определяет максимальный объем информации, который можно передать с использованием метода `POST`. По умолчанию значение `post_max_size` равно 8 Мбайт, но его можно изменить.
- ✓ **Поле** `file`**, которое используется для удаленной загрузки файла.**

Значение `MAX_FILE_SIZE` должно быть установлено перед удаленной загрузкой файла. Лишь в этом случае можно ограничить его размер.

При загрузке файла он сохраняется во временном каталоге. Поскольку после завершения выполнения сценария этот файл из временного каталога удаляется, следует скопировать его в другое место. С помощью функции `phpinfo()` можно определить, где расположен временный каталог. Его можно изменить с помощью директивы `upload_tmp_dir` файла `php.ini`. Если же в файле `php.ini` путь не указан, то используется временный каталог, установленный по умолчанию.

Получение информации о загружаемом файле

Кроме самого файла, вместе со значениями полей формы передается и информация о самом файле. Эта информация сохраняется во встроенном массиве `$_FILES`, который можно использовать для получения данных о каждом загружаемом файле. При этом доступ к информации осуществляется посредством имен полей. Общий формат извлечения информации о файле имеет вид

```
$_FILES['имя-поля']['name']
$_FILES['имя-поля']['type']
$_FILES['имя-поля']['tmp_name']
$_FILES['имя-поля']['size']
```

Пусть, например, файл загружается с помощью следующего поля формы:

```
<input type="file" name="user_file">
```

Тогда при загрузке файла `test.txt` информацию о нем можно получить следующим образом:

```
$_FILES[user_file][name] = test.txt
$_FILES[user_file][type] = text/plain
$_FILES[user_file][tmp_name] = D:\WINNT\php92C.tmp
$_FILES[user_file][size] = 435
```

В массиве _FILES значение name определяет имя загружаемого файла; type — его тип; tmp_name — путь или имя временного файла; size — размер файла. Обратите внимание, что в поле name содержится только имя файла, а в поле tmp_name — полный путь и имя.

Если же размер файла оказался слишком большим, то в поле tmp_name будет содержаться значение none, а в поле size — 0.

По умолчанию загружаемые файлы временно сохраняются в системном каталоге Windows (Windows — в системе Win98/XP и Winnt — в Win2000) и в каталоге /tmp систем Unix/Linux. Чтобы изменить временный каталог, нужно модифицировать файл php.ini. Для этого найдите в этом файле строку

```
;upload_tmp_dir =
```

Затем удалите в начале строки точку запятой и задайте требуемый путь. Например:

```
upload_tmp_dir = d:\tempfiles
```

При этом каталог tempfiles должен быть создан предварительно. В противном случае загружаемые файлы будут сохраняться в каталоге, заданном по умолчанию.

Перемещение файлов в требуемый каталог

Для перемещения загруженных файлов из временного каталога в требуемое местоположение предназначена функция move_uploaded_file() со следующим общим синтаксисом:

```
move_uploaded_file(путь/имя_временного_файла,
путь/имя_постоянного_файла);
```

В элементе tmp_name массива $_FILES хранится временное имя файла и его местоположение. Так что для перемещения файла, например в каталог c:\data\new_file.txt, можно воспользоваться функцией move_uploaded_file() следующим образом:

```
move_uploaded_file($_FILES['user_file']['tmp_name'],'c:\data\new_file.txt');
```

Следует заметить, что каталог, в который перемещается файл (в приведенном примере c:\data\), должен быть предварительно создан, поскольку функция move_upload_file() этого не делает.

С точки зрения обеспечения безопасности удаленная загрузка файлов может оказаться достаточно опасным делом. Эти файлы могут содержать опасное содержимое. Поэтому перед сохранением файла необходимо тщательно проверить его формат и размер. Иногда для обеспечения большей защищенности стоит даже изменить имя и расширение загруженного файла.

Совмещая все вместе

Пример полноценного сценария приведен в листинге 11.3. Этот сценарий отображает форму для загрузки файлов изображений и позволяет сохранять эти файлы. Кроме того, после проверки файла и его успешной загрузки выводится соответствующее сообщение. Форма для загрузки изображений показана на рис. 11.1.

Листинг 11.3. Сценарий для загрузки файлов с помощью формы с методом POST

```
<?php
 /* Имя сценария: uploadFile.php
  * Описание:     загружает файл с помощью протокола HTTP
  *               и формы с методом POST.
  */
```

```
  if(!isset($_POST['Upload']))                                    #5
  {
    include("form_upload.inc");
  } # endif
  else                                                            #9
  {
    if($_FILES['pix']['tmp_name'] == "none")                      #11
    {
      echo "<b>Файл не был загружен. Проверьте его
              размер. Размер файла должен быть менее 500K.<br>";
      include("form_upload.inc");
      exit();
    }
    if(!ereg("image", $_FILES['pix']['type']))                    #16
    {
      echo "<b>Файл не является изображением.
              Попробуйте загрузить другой файл.</b><br>";
      include("form_upload.inc");
      exit();
    }
    else                                                          #23
    {
      $destination = 'c:\data'. "\\".$_FILES['pix']['name'];
      $temp_file = $_FILES['pix']['tmp_name'];
      move_uploaded_file($temp_file, $destination);
      echo "<p><b>Файл успешно загружен:</b>
            {$_FILES['pix']['name']}
            ({$_FILES['pix']['size']})</p>";
    }
  }
?>
```

В листинге 11.3 заслуживают внимания следующие строки.

- ✓ В **строке 5** с использованием условного оператора `if` проверяется, были ли значения из формы переданы пользователем. Если нет, исходная форма отображается путем включения файла, приведенного в листинге 11.4.
- ✓ **Строка 9**. В блоке `else` обрабатывается файл и вся полученная информация.
- ✓ В **строке 11** с использованием оператора `if` проверяется успешность загрузки файла. Если файл не был загружен, выводится соответствующее сообщение об ошибке и исходная форма отображается повторно.
- ✓ В **строке 16** проверяется корректность формата файла. Если загружаемый файл не является изображением, выводится соответствующее сообщение об ошибке и исходная форма отображается заново.
- ✓ **Строка 23**. В блоке `else` (т.е. при успешной загрузке) загруженный файл перемещается в требуемое местоположение и выводится соответствующее сообщение.

В листинге 11.4 приведен код включаемого файла, который обеспечивает отображение исходной формы для загрузки изображений.

Листинг 11.4. Включаемый файл, обеспечивающий отображение формы для загрузки файлов

```
<!-- Имя сценария: form_upload.inc
     Описание:     отображает форму для загрузки файла -->
<html>
<head><title>Загрузка файла</title></head>
<body>
<ol><li>Введите имя изображения, которое вы
        хотите поместить в архив, или воспользуйтесь
        кнопкой Обзор, чтобы указать его
        местоположение.</li>
    <li>После появления пути к файлу в
        текстовом поле щелкните на кнопке Загрузить
        изображение</li>
</ol>
<div align="center"><hr>
<form enctype="multipart/form-data"
        action="uploadFile.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="500000">
  <input type="file" name="pix" size="60">
  <p><input type="submit" name="Upload"
        value="Загрузить изображение">
</form>
</body></html>
```

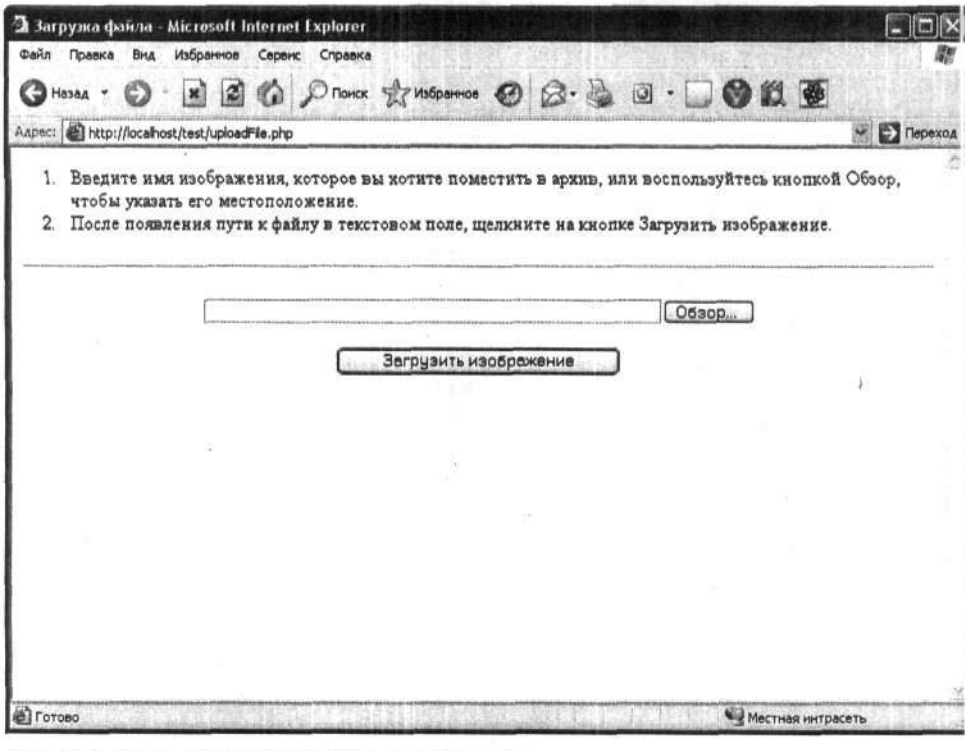


Рис. 11.1. Форма для загрузки файла с изображением

Обратите внимание, что в этом файле содержится только HTML-код.

Исходная форма для загрузки файлов изображена на рис. 11.1. В ней содержится текстовое поле для ввода имени загружаемого файла и кнопка для его поиска.

Использование JavaScript и PHP

При разработке Web-узлов может оказаться, что нужно использовать и язык сценариев JavaScript. Например, может понадобиться изменить содержимое Web-страницы в зависимости от местоположения курсора мыши или щелчка на ней, т.е. изменить страницу без ее повторного отображения. Это нельзя сделать с помощью PHP, поскольку он является серверным языком разработки сценариев. Интерпретатору PHP ничего неизвестно о том, что происходит в клиентском браузере, поскольку он выполняет сценарии только на сервере. Поэтому для изменения Web-страницы следует воспользоваться языком создания клиентских сценариев, к которым и относится язык JavaScript.

Пользователь может отключить режим использования сценариев JavaScript, и браузер не сможет их выполнить. Поэтому, прежде чем принять решение об использовании языка JavaScript, необходимо удостовериться в том, что соответствующий режим поддерживается клиентским программным обеспечением.

В данной главе язык клиентских сценариев JavaScript подробно рассматриваться не будет. Здесь речь пойдет лишь о том, как совместно использовать языки сценариев JavaScript и PHP. (Более подробную информацию о языке JavaScript можно найти в книге *JavaScript для "чайников"* Эмили Вандер Вер, вышедшей в издательстве "Диалектика".)

Добавление кода JavaScript в сценарий PHP

Код JavaScript, так же как и код HTML, интерпретируется и выполняется клиентским браузером. Поэтому его можно добавить в сценарий PHP точно так же, как и код HTML. По существу, сценарии JavaScript являются частью HTML-кода Web-страницы, которые можно добавить следующим образом:

```
<script language="JavaScript">
    код JavaScript
</script>
```

В сценариях PHP код JavaScript используется точно так же, как и код HTML: выводится с помощью функции `echo`. Например:

```
<?php
  echo "<script language=\"JavaScript\">
          <!--
            document.write('Данная страница изменялась: '+ docu-
ment.lastModified + '<br>')
          // -->
        </script>";
?>
```

При выполнении этого фрагмента кода в браузере будет выведена строка с датой и временем, когда данная страница была в последний раз модифицирована:

```
Данная страница изменялась: 03/24/2003 12:01:47
```

Код JavaScript, так же как и дескрипторы HTML, можно размещать вне дескрипторов PHP и не использовать вместе с функцией `echo`. В этом случае он будет обрабатываться только клиентским браузером.

Использование переменных PHP в сценариях JavaScript

Переменные PHP допустимо использовать в коде JavaScript точно так же, как и дескрипторы HTML. Например, в приведенный выше код можно добавить переменную `$string` следующим образом:

```
<?php
  $string = "Данная страница изменялась:";
  echo "<script language=\"JavaScript\">
          <!--
            document.write('$string'
              + document.lastModified + '<br>')
          // -->
        </script>";
?>
```

В свою очередь, в сценариях JavaScript используются собственные переменные, которым можно присваивать значения переменных PHP. Например:

```
<?php
  $string = "Данная страница изменялась:";
  echo "<script language=\"JavaScript\">
          <!--
            var message = \"$string\";
            document.write(message
              + document.lastModified + '<br>')
          // -->
        </script>";
?>
```

Поскольку код JavaScript выполняется в клиентском браузере, то "обменяться" значениями переменных JavaScript с переменными PHP на текущей странице нельзя. Значения JavaScript должны быть переданы следующему сценарию PHP. Для этого их можно добавить к URL-адресу, разместить в данных cookie или передать в качестве элемента формы.

Глава 12

Хранение данных с использованием PHP

В этой главе...

- Запись и чтение файлов
- Обмен данными между PHP и другими приложениями
- Поддержка баз данных в PHP
- Использование PHP для взаимодействия с базами данных
- Обработка ошибок при подключении к базе данных

Во многих приложениях требуется обеспечить длительное хранение информации. Непродолжительное хранение данных в сценариях PHP можно осуществить с помощью механизма *сеансов* (session). Однако зачастую информация должна быть доступна в любое другое время, например через день или на следующей неделе. Как уже говорилось в главе 11, после завершения сеанса это можно обеспечить с помощью данных cookie. Однако при их использовании разработчик не управляет режимом поддержки данных cookie. Пользователь может удалить или изменить информацию в любой момент или вообще отключить соответствующий режим в своем браузере. Поэтому информацию следует хранить в более безопасном месте с ограниченным доступом, например на сервере.

На сервере информация может храниться в текстовых файлах или в базе данных. Текстовые файлы хранятся в локальной файловой системе. Их можно прочитать с помощью команд операционной системы, например команды `at` систем Linux/Unix. Для редактирования таких файлов можно воспользоваться любым текстовым редактором, таким как Notepad или vi. Информация в файлах обычно хранится в строковой форме. Для ее извлечения сценарию PHP необходимо "знать" соответствующий формат. Например, для извлечения из файла имени покупателя сценарию PHP необходимо "знать" о том, что имена покупателей занимают первые 20 символов каждой строки.

В свою очередь, при использовании баз данных требуется установить специализированное программное обеспечение, такое как MySQL или Oracle. Базы данных тоже представляют собой файлы, которые, однако, создаются соответствующими приложениями и могут быть доступны только через них. В базе данных можно хранить информацию любой сложности. При этом нет необходимости знать принципы ее хранения — важно иметь навыки работы с соответствующим программным обеспечением. Например, для получения имени покупателя в сценарии PHP достаточно правильно сформулировать запрос на языке SQL, чтобы система управления базами данных обработала этот запрос и предоставила результат. При этом абсолютно не важно, где и в каком виде хранится требуемая информация.

Ниже приводится несколько преимуществ использования текстовых файлов по сравнению с базами данных.

- ✓ **Гибкость**. Создавать и хранить данные можно в файловой системе любой операционной системы. При этом не требуется устанавливать какое-либо дополнительное программное обеспечение. Кроме того, текстовые файлы можно обрабатывать во многих текстовых редакторах или электронных таблицах.
- ✓ **Простота использования**. В установке дополнительной системы управления базами данных, создании и проектировании самой базы данных нет никакой

необходимости. Достаточно создать файл средствами PHP и хранить в нем полезную информацию.

- ✓ **Экономия дискового пространства**. Текстовые файлы занимают гораздо меньше дискового пространства, чем базы данных.

Подводя итог, можно сказать, что текстовые файлы являются простым, эффективным и экономным (с точки зрения использования ресурсов) способом хранения небольших объемов информации. Они просты в использовании и обеспечивают быстрый доступ к данным. Кроме того, текстовый формат поддерживается различными редакторами и электронными таблицами. В то же время текстовые файлы доступны для всех, кто обладает правами на доступ к требуемому каталогу, поэтому их можно использовать в качестве средства обмена информацией.

Базы данных также имеют свои преимущества.

- ✓ **Безопасность**. Кроме обеспечения безопасности на уровне операционной системы, базы данных предоставляют дополнительный уровень защиты. Информация в базах данных защищена от внешнего вторжения гораздо лучше, чем в простых файлах.
- ✓ **Поддержка сложной структуры данных**. В базах данных можно хранить информацию с очень сложной структурой, различными типами данных и взаимосвязями. Это позволяет существенно упростить поиск и извлечение требуемой информации.
- ✓ **Поддержка многопользовательского режима**. Если к одному и тому же файлу (например, с именами и адресами клиентов) обращаются много пользователей, то с помощью средств системы управления базами данных можно обеспечить согласованность данных и предотвратить опасность их затирания или потери.

Одним словом, при использовании баз данных требуется гораздо больше усилий и дискового пространства. Однако вместе с тем они позволяют обрабатывать данные со сложной структурой. Базы данных существенно упрощают извлечение требуемой информации, обеспечивают больший уровень защиты и предоставляют возможность одновременного подключения многих пользователей.

При установке PHP 5 по умолчанию устанавливается расширение SQLite, которое предназначено для использования файлов и баз данных. При использовании расширения SQLite информация хранится в текстовых файлах, однако для доступа к ним применяется стандартный язык запросов SQL. Поэтому устанавливать дополнительное программное обеспечение не требуется. Расширение SQLite удобно использовать для хранения и обработки небольших объемов данных, однако для организации большого хранилища данных все же лучше воспользоваться специализированными пакетами.

Использование текстовых файлов

Текстовые файлы использовать гораздо проще, чем базы данных. Кроме того, при их использовании не нужно устанавливать дополнительное программное обеспечение. Для чтения или записи в файл достаточно воспользоваться соответствующими операторами языка PHP

При работе с текстовыми файлами нужно выполнить следующие этапы.

1. **Открыть файл.**
2. **Записать или извлечь данные из файла.**
3. **Закрыть файл.**

Каждый из этих этапов более подробно рассматривается в следующих разделах.

Доступ к файлам

Перед чтением или записью информации из файла необходимо его открыть. Для этого предназначена функция

```
$fh = fopen("имя_файла", "режим");
```

Переменная `$fh` является *дескриптором файла* (file handle), который используется при чтении или записи данных в файл. Дескриптор файла содержит информацию о местоположении открытого файла.

При открытии файла необходимо указать соответствующий *режим*, который определяет те действия, которые планируется выполнять с файлом после его открытия. Возможные режимы открытия файлов приведены в табл. 12.1.

Таблица 12.1. Режимы открытия файлов

Режим	Описание	Что происходит, если файл не существует
`r`	Только чтение	Выводится предупреждающее сообщение
`r+`	Чтение и запись	Выводится предупреждающее сообщение
`w`	Только запись	Создается новый файл. Если файл существует, новый файл перезаписывается поверх старого
`w+`	Чтение и запись	Создается новый файл. Если файл существует, новый файл перезаписывается поверх старого
`a`	Добавление данных в конец файла	Если файл не существует, создается новый файл
`a+`	Чтение и добавление данных в конец файла	Если файл не существует, создается новый файл

Параметр *имя_файла* может содержать имя файла (`filename.txt`), полный путь к нему (`c:/data/filename.txt`) или адрес URL (`http://yoursite.com/filename.txt`).

Открытие файла в режиме чтения

Для открытия файла `file1.txt` и считывания из него информации воспользуйтесь функцией `fopen()` со следующими параметрами:

```
$fh = fopen("file1.txt", "r");
```

В результате интерпретатор PHP выполнит поиск файла `file1.txt` в текущем каталоге. Текущим считается каталог, в котором расположен сценарий PHP. Если файл не существует, может быть выведено сообщение об ошибке (в зависимости от выбранного режима вывода сообщений об ошибках, см. главу 4):

```
Warning: fopen(file1.txt): failed to open stream: No such file or
directory in d:\test2.php on line 15
(Предупреждение: fopen(file1.txt): ошибка открытия потока: Файла или
каталога не существует в файле d:\test2.php на строке 15)
```

Однако не забывайте о том, что вывод предупреждения не приведет к прерыванию выполнения сценария. Поэтому любые функции, оперирующие с несуществующим файлом, будут работать некорректно.

Для прерывания выполнения сценария в случае отсутствия файла воспользуйтесь фрагментом

```
$fh = fopen("file1.txt", "r") or die("Не удалось открыть файл");
```

Как упоминалось в главе 8, функция `die()` прерывает выполнение сценария и выводит заданное сообщение.

Открытие файла в режиме записи

Для открытия файла в заданном каталоге в режиме записи воспользуйтесь функцией `fopen()` со следующими параметрами:

```
$fh = fopen("c:/testdir/file1.txt", "w");
```

Если заданный файл не существует, будет создан новый файл. Однако если отсутствует требуемый каталог, будет выведено сообщение об ошибке. (Другими словами, каталог должен быть предварительно создан.)

Проверить существование каталога можно следующим образом:

```
if(is_dir("c:/tester"))
{
    $fh = fopen("c:/testdir/file1.txt", "w");
}
```

Теперь открытие файла будет выполняться только в том случае, если существует требуемый каталог.

Открытие файла на другом Web-узле

Файл, расположенный на другом Web-узле, можно открыть следующим образом:

```
$fh = fopen("http://janet.valade.com/index.html", "r");
```

URL-адрес можно использовать для открытия файлов только для чтения.

Закрытие файла

После использования файл нужно закрыть:

```
fclose($fh);
```

В качестве входного параметра функция использует дескриптор файла `$fh`, который ранее был создан при его открытии.

Запись в файл

После открытия файла можно приступать к записи в него различной информации. Для этого предназначена функция `fwrite()` со следующим синтаксисом:

```
fwrite($fh, данные);
```

Параметрами этой функции являются дескриптор файла `$fh`, созданный при его открытии, и *данные*, которые необходимо сохранить. В качестве *данных* можно использовать текстовую строку или переменную. Например:

```
$today = date("Y-m-d");
$fh = fopen("file2.txt", "a");
fwrite($fh, $today);
fclose($fh);
```

В этом фрагменте текущая дата записывается в файл `file2.txt`. Причем файл открывается для добавления информации. Если при открытии файла не существовало, будет создан новый файл. При этом данные будут записаны в его первую строку. При открытии существующего файл текущая дата добавляется в конец файла. Таким образом, можно создать журнал регистрации, содержащий, например, время запуска сценария. Следует заметить, что функция `fwrite()` записывает в файл только ту информацию, которая была указана при ее вызове. Так, после повторного запуска сценария содержимое файла будет выглядеть следующим образом:

```
2003-04-222003-04-22
```

Гораздо удобнее размещать различные даты в отдельных строках. Для этого функцию `fwrite()` нужно вызвать таким образом:

```
fwrite($fh, $today"\n");
```

После добавления символа `\n` файл `file2.txt` будет содержать следующие данные:

```
2003-04-22
2003-04-22
```

Если в файл нужно *добавить* информацию, его необходимо открыть в режиме `a`. При открытии в режиме записи файл каждый раз будет создаваться заново, а информация — *перезаписываться*.

Чтение файла

Для чтения файла предназначена функция `fgets()` со следующим синтаксисом:

```
$line = fgets($fh);
```

где `$fh` — дескриптор файла, который был создан при его открытии. Функция `fgets()` считывает файл до тех пор, пока не будет достигнут конец строки или файла. При этом извлеченная информация сохраняется в переменной `$line`. Построчно можно прочитать весь файл. Для определения конца файла в языке PHP предназначена функция `feof()`. В следующем примере считываются и выводятся все строки файла:

```
while(!feof($fh))
{
    $line = fgets($fh);
    echo "$line";
}
```

Функция `feof()` возвращает значение `TRUE` при достижении конца файла с дескриптором `$fh`. Восклицательный знак в условном выражении цикла `while` обеспечивает чтение всех строк файла.

Если приведенный фрагмент использовать для чтения журнала регистрации, созданного в предыдущем разделе, получим следующий результат:

```
2003-04-22
2003-04-22
```

Как можно увидеть, символ перевода строки используется функцией `fgets()` по умолчанию. В некоторых случаях этого не требуется, так что эти символы нужно удалить:

```
while(!feof($fh))
{
    $line = rtrim(fgets($fh));
    echo "$line";
}
```

Функция `rtrim()` удаляет пробелы в конце строки и символы перевода строки. Так что при ее использовании будет получен следующий результат:

```
2003-04-222003-04-22
```

Чтение файлов по частям

Иногда из файла необходимо считать строки определенного размера. Для этого следует воспользоваться функцией `fgets()` со следующим синтаксисом:

```
$line = fgets($fh, n);
```

В данном случае строка длиной $n-1$ будет считываться до тех пор, пока не будет достигнут конец строки или файла. Например:

```
while(!feof($fh))
{
   $char4 = fgets($fh, 5);
   echo "$char4\n";
}
```

В приведенном фрагменте считываются каждые четыре символа, пока не будет достигнут конец файла. При этом будет получен следующий результат:

```
2003
-04-
22

2003
-04-
22
```

Обратите внимание, что после каждой строки по умолчанию используется символ перевода строки.

Размещение файла в массиве

Зачастую содержимое всего файла удобно разместить в массиве. Это можно выполнить следующим образом:

```
$fh = fopen("file2.txt", "r");
while(!feof($fh))
{
   $content[] = fgets($fh);
}
fclose($fh);
```

В результате будет создан массив `$content`, ключами которого будут целые числа, а значениями — строки, считанные из файла.

То же самое можно осуществить с помощью одной функции `file()`. При этом получится аналогичный результат:

```
$content = file("file2.txt");
```

Эта функция открывает файл, размещает строки в массиве `$content` и закрывает файл.

Если открываемый файл очень большой, то использование функций `file()` может существенно замедлить работу сценария. Это зависит от размера доступной памяти на компьютере. Если сценарий действительно работает медленно, вместо функции `file()` следует воспользоваться функцией `fgets()`, а затем проверить, повлияло ли это на быстродействие.

Функцию `file()` можно использовать для открытия файлов во включаемом каталоге (см. главу 8):

```
$content = file("file2.txt", 1);
```

Параметр `1` указывает, что поиск файла `file2.txt` следует осуществлять в каталоге включений, а не в текущем каталоге.

Размещение файла в строке

Иногда содержимое файла необходимо разместить в одной длинной строке. Например, это может оказаться полезным, если данные из файла нужно отправить по электронной почте. Для этого предназначена

```
$content = file_get_contents("file2.txt", 1);
```

Функция `file_get_contents()` полностью аналогична `file()`, за исключением того, что содержимое файла размещается не в массиве, а в строке. Затем можно вывести значение переменной `$content`:

```
echo $content;
```

При этом получим следующий результат:

```
2003-04-22
2003-04-22
```

Обратите внимание, что символы перевода строки входят в состав строки `$content` и, естественно, учитываются при выводе.

Функция `file_get_contents()` появилась в PHP 4.3.0. В более ранних версиях она отсутствует.

Обмен данными с другими программами

Файлы оказываются очень полезными, если нужно обеспечить взаимодействие сценариев PHP с другими приложениями. Практически все без исключения программы предоставляют возможность считывания или записи данных в файл. Например, текстовые редакторы по умолчанию сохраняют информацию в своем собственном формате. Однако они позволяют работать и с текстовыми файлами, т.е. созданный с использованием сценария PHP текстовый файл можно прочитать и в других приложениях.

Однако при взаимодействии со многими программными продуктами все же приходится считаться с используемым в них специальным форматом. Например, в адресной книге каждому фрагменту данных соответствует свое предопределенное место. Так, первые 20 символов строки могут использоваться для хранения имени, вторые 20 символов — для адреса и т.д. Поэтому для работы с файлами подобного типа нужно точно знать их формат. В сценарии PHP такой формат можно сгенерировать с помощью функции `fwrite()`, которая уже рассматривалась выше.

Файлы CSV (comma-separated values — значения, разделяемые запятой), иногда называемые также файлами с символами-разделителями , (comma-delimited file), являются стандартным средством обмена данными между различными приложениями. Файлы CSV могут использоваться для обмена информацией, содержащейся в таблице (со столбцами и строками). Такой формат поддерживается большинством табличных приложений (например, Microsoft Excel). Файлы CSV удобно использовать и при передаче данных между различными базами данных, такими как MySQL и Microsoft Access. Формат CSV достаточно распространен и поддерживается различным программными обеспечением.

CSV-файл имеет следующую структуру: каждая его строка соответствует строке таблицы, а значения, разделенные запятой, — столбцам. Рассмотрим следующий пример адресной книги, организованной с использованием файла CSV:

```
Джон Смит, ул. Дубов 1234, Большой город, OR, 99999
Мэри Джонс, Сосновая ул. 5678, Больший город, ME, 11111
Луис Рохас, ул. Вязов 1234, Самый большой город, TX, 88888
```

Excel позволяет прочитать этот файл и преобразовать его в таблицу, состоящую из пяти столбцов. При этом запятая используется в качестве символа-разделителя. Outlook Express также сможет импортировать этот файл в свою адресную книгу.

Следующий фрагмент кода на языке PHP позволяет создать файл в формате CSV.

```
$address[] = "Джон Смит, ул. Дубов 1234, Большой город, OR, 99999";
$address[] = "Мэри Джонс, Сосновая ул. 5678, Больший город, ME, 11111";
$address[] = "Луис Рохас, ул. Вязов 1234, Наибольший город, TX, 88888";
$fh = fopen("addressbook.txt", "a");
```

```
for ($i=0;$i<3;$i++)
{
   fwrite($fh,$address[$i]. "\n");
}
fclose($fh);
```

Для чтения CSV-файла можно воспользоваться функцией `file()` или `fgets()`, как описывалось выше, в разделе "Размещение файла в массиве". Однако специально для этих целей в языке PHP имеется функция `fgetcsv()`. При чтении файла с использованием этой функции каждая строка записывается в массив, элементами которого являются значения столбцов. Например:

```
$address = fgetcsv($fh, 1000);
```

В приведенной строке из файла с идентификатором `$fh` считывается строка длиной 1000 символов. Результатом выполнения функции `fgetcsv()` будет следующий массив:

```
$address[0] = Джон Смит
$address[1] = ул. Дубов 1234
$address[2] = Большой город
$address[3] = OR
$address[4] = 99999
```

Формат CSV очень удобно использовать при передаче данных между различными приложениями. Однако не забывайте о том, что если символ , является частью данных, то его нельзя использовать в качестве символа-разделителя. Рассмотрим следующую строку:

```
Компания Смита, мл., ул. Дубов 1234, Большой город, OR, 99999
```

Запятая используется в названии компании и соответственно при считывании приведет к его разделению его на две части (столбца): `Компания Смита` и `мл.`, т.е. вместо ожидавшихся пяти столбцов будет шесть. Для решения этой проблемы в качестве символа-разделителя следует использовать другой символ, например символ табуляции (`\t`). (Именно так и поступают чаще всего.) В этом случае файл уже будет называться TSV (tab-separated values — значения, разделенные символами табуляции).

Для чтения такого файла можно воспользоваться функцией `fgetcsv()` такого вида:

```
$address = fgetcsv($fh, 1000, "\t");
```

В общем случае в качестве третьего параметра, т.е. разделителя, можно указать любой символ.

В листинге 12.1 приведен сценарий, который преобразует файл CSV в файл TSV.

Листинг 12.1. Сценарий, преобразующий файл CSV в TSV-файл

```
<?php
 /* Имя сценария: Convert
  * Описание:     считывает CSV-файл и записывает данные в
  *               TSV-файл. CSV-файл должен иметь расширение .CSV.
  */
  $myfile = "testing";                                    #7
  function convert($filename)                             #8
  {
    if(@$fh_in = fopen("{$filename}.csv", "r"))           #10
    {
      $fh_out = fopen("{$filename}.tsv", "a");            #12
      while(!feof($fh_in))                                #13
      {
        $line = fgetcsv($fh_in, 1024);                    #15
        if($line[0] == "")                                #16
        {
```

```
        fwrite($fh_out, "\n");
      }
      else {                                          #20
        fwrite($fh_out,implode($line, "\t")."\n");#21
      }
    }
    fclose($fh_in);
    fclose($fh_out);
  }
  else {                                              #27
    echo "Файл не существует\n";
    return FALSE;
  }
  echo "Преобразование завершено!\n";
  return TRUE;                                        #32
}
convert($myfile);                                     #34
?>
```

В листинге 12.1 некоторые строки пронумерованы. Ниже представлены соответствующие пояснения.

- ✓ В **строке 7** задается имя обрабатываемого файла.
- ✓ В **строке 8** определяется функция `convert()` с параметром `$filename`.
- ✓ **Строка 10**. В этой строке для чтения открывается файл с расширением `.csv`. Если файл был успешно открыт, выполняется его преобразование в блоке `if`. В противном случае выполняется блок `else` строки 27.
- ✓ **Строка 12**. В этой строке в режиме добавления открывается файл с расширением `.tsv`. Предполагается, что файл находится в текущем каталоге, т.е. там же, где расположен и сам сценарий.
- ✓ **Строка 13**. В этой строке начинается цикл `while`, который завершается при достижении конца файла.
- ✓ **Строка 15**. Строка файла считывается в массив `$line`. Каждый элемент массива содержит значение столбца.
- ✓ **Строка 16**. В этой строке проверяется, содержит ли строка исходного файла какую-либо информацию. Если нет, выводится символ перевода строки (`\n`). Тем самым обеспечивается, что пустая строка исходного файла будет содержаться и в выходном файле.
- ✓ **Строка 20**. Если строка исходного файла не пуста, выполняется ее преобразование в формат TSV и запись в файл.
- ✓ **Строка 21**. В этой строке выполняется преобразование строки. При этом результат записывается в выходной файл. Функция `implode()` используется для преобразования массива `$line` в текстовую строку, в которой в качестве символов-разделителей применяются символы табуляции (`\t`).
- ✓ **Строка 27**. Блок `else` выполняется в том случае, если исходный файл отсутствует. При этом выводится соответствующее сообщение об ошибке, а функция `convert()` возвращает значение `FALSE`.
- ✓ **Строка 32**. Если функция выполнена успешно, возвращается значение `TRUE`.
- ✓ **Строка 34**. В этой строке вызывается функция `convert()`, которой в качестве параметра передается значение переменной `$myfile`.

Работа с базами данных

Если необходимо хранить данные со сложной структурой, обеспечивать безопасность и обрабатывать множество одновременных запросов, базы данных оказываются гораздо более эффективными, чем обычные файлы. Если же вы имеете опыт работы со специальным программным обеспечением, то это позволит существенно упростить решение поставленной задачи.

Системы управления базами данных

Базы данных (database) напоминают электронную картотеку, в которой хранится упорядоченная информация, извлекаемая по мере необходимости. База данных может быть небольшой, с простой структурой данных, например база данных имен, адресов и телефонных номеров всех ваших друзей. В свою очередь, в базе данных Amazon содержится огромное количество информации, которая имеет чрезвычайно сложную структуру.

Формально *базу данных* можно определить как файл или группу файлов, в которых хранятся данные. Доступ к этим данным можно получить с использованием системы управления базами данных (СУБД) (DBMS — Database Management System). В настоящее время практически все СУБД являются *реляционными* (relational), поддерживающими хранение данных в наборе взаимосвязанных таблиц.

Одно из значительных преимуществ языка PHP заключается в поддержке более двадцати возможных СУБД. К наиболее известным относятся следующие.

- ✓ IBM DB2
- ✓ Informix
- ✓ Ingres
- ✓ Microsoft SQL Server (MS SQL)
- ✓ mSQL
- ✓ MySQL
- ✓ Oracle
- ✓ PostgreSQL
- ✓ Sybase

Кроме того, язык PHP поддерживает также и стандарт ODBC (Open Database Connectivity — открытый интерфейс доступа к базам данных) компании Microsoft. Его поддерживают практически все реляционные СУБД, особенно те из них, которые предназначены для использования в операционной системе Windows. Используя поддержку ODBC в языке PHP, можно обращаться к таким базам данных, как DB2 и Access. При использовании стандарта ODBC для взаимодействия с базой данных необходимо установить соответствующий драйвер. Более подробную информацию по этому вопросу можно получить в соответствующей документации.

После установки СУБД к ней можно обращаться из сценариев PHP, в которых информацию нужно сохранить или извлечь из базы данных. Выбор и установка системы управления базами данных выполняется независимо от PHP. После установки СУБД следует проверить ее работоспособность и научиться ее использовать. После этого можно приступать к ее использованию в сценариях PHP.

Выбор реляционной СУБД зависит от реальных потребностей. Одна из них может прекрасно подходить для решения одной задачи, тогда как при решении другой задачи, возможно, потребуется воспользоваться совершенно другим программным обеспечением. Следует взвесить все "за" и "против" и лишь после этого приступать к выбору системы управления базами данных, которая будет использоваться впоследствии. При выборе СУБД следует учитывать следующее.

- ✓ **Стоимость**. Стоимость СУБД может быть достаточно разной. Она может быть как бесплатной, так и достаточно дорогостоящей. Так, системы управления базами данных MySQL, mSQL и PostgreSQL были разработаны как открытое программное обеспечение и, следовательно, являются бесплатными. Другие СУБД, например Sybase, MS SQL Server и Oracle, являются коммерческим программным обеспечением, приобретение которого связано с существенными денежными затратами.
- ✓ **Предоставляемые возможности**. Различные системы управления базами данных предоставляют разные возможности. Например, система mSQL обладает ограниченным набором функций, однако в определенных условиях их может оказаться вполне достаточно. С другой стороны, Oracle позволяет выполнить практически любые действия, кроме, возможно, управления автомобилем. Кроме того, не следует забывать и о том, что чем больше возможностей предоставляет СУБД, тем больше она потребляет ресурсов и является более дорогостоящей. Поэтому не стоит устанавливать компоненты, которые в дальнейшем наверняка не понадобятся.
- ✓ **Потребляемые ресурсы**. Некоторые СУБД потребляют гораздо больше ресурсов (например, дискового пространства и памяти), чем другие. Так, системы mSQL и MySQL не требуют больших ресурсов, а для функционирования Oracle в зависимости от установок может потребоваться достаточно много вычислительных ресурсов.
- ✓ **Поддержка**. Вне всякого сомнения, при использовании коммерческого и открытого программного обеспечения предоставляется различная техническая поддержка.
- ✓ **Коммерческое программное обеспечение**. Компании-разработчики предоставляют различную техническую поддержку потребителям своего коммерческого программного обеспечения. Иногда это связано с дополнительными денежными затратами, в других случаях — с потерей времени на ожидание связи со специалистами. Но в любом случае компания-разработчик всегда предоставляет помощь и помогает справиться с возникающими проблемами.
- ✓ **Программное обеспечение с открытым кодом**. При использовании открытого программного обеспечения нельзя воспользоваться прямой телефонной связью с его разработчиком. Его развитие обеспечивается сообществом пользователей. При возникновении проблем можно обратиться к соответствующему списку рассылки или форуму. Иногда это позволяет получить ответ гораздо быстрее, чем при телефонном звонке в службу технической поддержки компании-разработчика.

После выбора системы управления базами данных необходимо установить соответствующее программное обеспечение, а затем научиться его использовать. Кроме того, нужно уметь проектировать и создавать базы данных, к которым впоследствии будут обращаться сценарии PHP.

Как правило, база данных состоит из двух частей: структуры данных и самих данных. В свою очередь, структура определяет саму базу данных и таблицы, в которых будет храниться информация. Перед тем как приступить к наполнению базы данных информацией, необходимо спроектировать саму структуру базы данных. Реляционные таблицы аналогичны любым другим таблицам и состоят из строк и столбцов.

Пусть, например, необходимо разработать интерактивный каталог товаров, который позволял бы клиентам делать заказы. Для этого можно создать базу данных `Catalog` (Каталог), в состав

которой входила бы таблица `Product` (Товар). Каждая строка таблицы `Product` соответствует одному товару. В данном случае товаром являются рубашки. Тогда каждый столбец будет соответствовать их различным характеристикам: название (тенниска, белая рубашка к вечернему костюму, спортивная рубашка с короткими рукавами и т.д.), описание, размер, цвет и т.п.

При создании таблицы каждому столбцу нужно присвоить название, т.е. *имя поля* (field name). Так, таблица `Product` будет содержать столбцы со следующими именами:

- ✓ `Type` (тип)
- ✓ `Description` (описание)
- ✓ `Size` (размер)
- ✓ `Color` (цвет)
- ✓ `Price` (цена)

Конечно, в базе данных `Catalog` могут содержаться и другие таблицы с информацией о стоимости доставки и налогах.

Кроме проектирования и создания базы данных, необходимо решить также и вопросы, связанные с обеспечением безопасности. Обеспечение требуемого уровня защиты данных является одной из отличительных возможностей системы управления базами данных. В то же время это может существенно усложнить сохранение и извлечение информации.

Так, доступ к данным может получить лишь зарегистрированный пользователь. При этом, если база данных размещается на сервере Web-хостинговой компании, ее администраторы должны создать и предоставить соответствующие учетные записи. Если же вы сами обеспечиваете работоспособность системы управления базы данных, все эти действия придется научиться делать самостоятельно.

После выполнения всех действий по проектированию и созданию базы данных можно приступать к ее использованию. Соответствующие средства языка PHP позволяют существенно упростить решение этой задачи.

Поддержка баз данных в PHP

Для взаимодействия с базами данных в языке PHP предназначены специальные функции. Причем каждой поддерживаемой СУБД соответствует свой собственный набор средств. Например, для взаимодействия с базой данных MySQL версии 4.0 или ниже следует воспользоваться функциями `mysql_connect()` и `mysql_query()`, а для подключения к базе данных MySQL версии 4.1 или выше — `mysqli_connect()` и `mysqli_query()`. Для работы с базой данных Sybase предназначены функции `sybase_connect()` и `sybase_query()`.

В PHP по умолчанию поддерживается стандарт ODBC, поддержку других баз данных нужно активизировать самостоятельно. При работе с Web-хостинговой компанией такую поддержку должны обеспечить ее администраторы. Если же база данных размещается на вашем компьютере, все требуемые настройки придется выполнить самостоятельно (подробности — в следующем разделе).

Поддержка баз данных в системах Unix/Linux/Mac

Поддержка баз данных обеспечивается при использовании в процессе компиляции модуля PHP соответствующего параметра. В приложении А параметры компиляции рассматриваются более подробно. Например, для включения поддержки mSQL в процессе компиляции нужно воспользоваться командой

```
./configure --with-msql=/usr/msql
```

В табл. 12.2 приведены параметры, включающие поддержку других систем управления базами данных. Помните о том, что если база данных устанавливается в каталоге по умолчанию, параметр DIR указывать не нужно. Например:

```
./configure --with-msql
```

Таблица 12.2. Параметры включения поддержки различных баз данных

СУБД	Параметр компиляции	Значение DIR по умолчанию
IBM DB2	`with-ibm-db2=`*DIR*	`/home/db2inst1/sqllib`
Informix	`with_informix=`*DIR*	Значение по умолчанию отсутствует
Ingres II	`with-ingres=`*DIR*	`/II/ingres`
mSQL	`with-msql=`*DIR*	`/usr/local/Hughes`
MySQL 4.0 или ниже	`with-mysql=`*DIR*	`/usr/local/mysql`
MySQL 4.1 или выше	`with-mysqli=`*DIR*	Значение по умолчанию отсутствует. Значение параметра DIR должно совпадать с путем к файлу `mysql_config`, который устанавливается вместе с MySQL 4.1 или выше
Oracle 7 или более новые версии	`with-oci8`	Значение по умолчанию содержится в переменной окружения ORACLE_HOME
Ранние версии Oracle	`with-oracle=`*DIR*	Значение по умолчанию содержится в переменной окружения ORACLE_HOME
PostgreSQL	`with-pgsql=`*DIR*	`/usr/local/pgsql`
Sybase	`with-sybase=`*DIR*	`/home/sybase`
Sybase-CT	`with-sybase-ct=`*DIR*	`/home/sybase`

После компиляции модуля PHP с соответствующим параметром нужно удостовериться в успешном завершении процесса компиляции. Это можно осуществить с помощью функции `phpinfo()`, которая предоставит всю необходимую для этого информацию.

Поддержка баз данных в системе Windows

Включение режима поддержки баз данных в модуле PHP для системы Windows предполагает выполнение двух следующих этапов.

1. **Копирование одной из динамически подключаемых библиотек (Dynamic Link Library — DLL) в основной каталог.**
2. **Активизация режима поддержки базы данных.**

После выполнения этих этапов, которые подробно описываются ниже, с помощью функции `phpinfo()` можно проверить конфигурационные параметры, связанные с поддержкой баз данных.

Копирование dll-библиотеки

Все необходимые dll-библиотеки можно найти в архивном zip-файле, который ранее был загружен с Web-узла PHP. Процесс загрузки архива и ручной установки модуля PHP более подробно рассматривается в приложении А. После установки в каталоге, в котором был уста-

новлен пакет PHP, можно найти также и каталог ext (например, c:\php\ext). Именно в этом каталоге и будут содержаться динамические библиотеки поддержки баз данных.

Скопируйте требуемый dll-файл в основной каталог PHP, такой как c:\php. Далее, для активизации поддержки баз данных, например PostgreSQL в системе Windows 2000, скопируйте файл c:\php\ext\php_pgsql.dll в основной каталог PHP. (Для этого вполне подойдут команды cd и copy.)

Если для установки модуля PHP использовалась автоматическая процедура инсталляции, то dll-файлы поддержки баз данных по умолчанию не устанавливаются. Поэтому после завершения автоматической процедуры установки все требуемые действия понадобится выполнить вручную, как описывалось выше.

Активизация режима поддержки баз данных

Для включения режима поддержки баз данных нужно внести изменения в конфигурационный файл php.ini. Найдите в этом файле строки следующего вида:

```
;extension=php_pgsql.dll
;extension=php_msql.dll
```

Каждый элемент этого перечня соответствует одной из баз данных, которая поддерживается языком PHP. Легко заметить, что в начале каждой строки содержится точка с запятой. Это означает, что все строки закомментированы и, следовательно, неактивны. Для включения режима поддержки требуемой базы данных найдите соответствующую строку и удалите в ее начале точку с запятой. Например:

```
extension=php_pgsql.dll
```

В этой строке активизируется поддержка системы управления базами данных PostgreSQL. Для того чтобы внесенные изменения вступили в силу, нужно перезапустить Web-сервер.

Если в файле php.ini был включен режим поддержки базы данных, а соответствующая динамическая библиотека не была скопирована, то при попытке запуска сценария будет выведено сообщение об ошибке примерно такого вида:

```
Unknown(): Unable to load dynamic library 'php_pgsql.dll'.
           The specified module could not be found.
(Unknown(): Не удается подключить динамическую библиотеку 'php_pgsql.dll .
           Указанный модуль не найден.)
```

Если оба этапа были выполнены корректно (копирование dll-библиотеки и активизация соответствующего расширения в файле php.ini), но при этом не было установлено программное обеспечение баз данных, будет выведено следующее сообщение об ошибке:

```
The dynamic link library msql.dll could not be found in the specified path
(Динамически подключаемая библиотека msql.dll отсутствует по указанному пути)
```

Для пользователей MS SQL. Кроме сервера баз данных необходимо также установить набор средств MS SQL Server Client Tools. Его можно найти на установочном компакт-диске MS SQL.

Для пользователей MySQL. Убедитесь, что вы используете файл php_mysql.dll для MySQL версии 4.0 или ниже или файл php_mysqli.dll для MySQL версии 4.1 или выше.

Взаимодействие с базой данных

Большинством систем управления базами данных поддерживается язык SQL (Structured Query Language — язык структурированных запросов). Оператор SQL, называемый также *запросом* (query), определяет те действия, которые должна выполнить система управления базой

данных. Например, SQL-запрос может быть предназначен для создания базы данных или таблицы, сохранения информации в базе данных, извлечения или удаления данных и т.д.

Несмотря на то что практически все системы управления базами данных предоставляют возможность использования языка SQL, при его использовании все же могут возникнуть определенные нюансы. Так, сервер mSQL поддерживает только ограниченный набор операторов SQL. В то же время Oracle и Sybase поддерживают свои собственные расширения SQL, которые даже не определены в соответствующем стандарте.

Полное описание языка SQL не входит в задачи этой книги. Если же вы нуждаетесь в получении дополнительной информации по этому вопросу, обратитесь к какой-нибудь другой книге. Например, в книге *mySQL*, 2-е издание, вышедшей в Издательском доме "Вильямс", рассматриваются вопросы использования сервера баз данных MySQL. Подробную информацию о языке запросов SQL можно найти в книге *SQL для "чайников"*, 5-е издание, которая вышла в издательстве "Диалектика".

Рассмотрим несколько простых SQL-запросов, которые ниже будут использоваться в реальных примерах сценариев PHP. Получить информацию из базы данных можно с помощью запроса

```
SELECT * FROM имя_таблицы
```

Этот запрос позволяет извлечь все данные, содержащиеся в таблице *имя_таблицы*. Символ * является специальным *символом-заполнителем* (wild card), который позволяет выбрать все поля таблицы. Предположим, необходимо извлечь информацию из базы данных `Catalog` (см. выше раздел "Системы управления базами данных"). В этой базе данных содержится одна таблица `Product` с информацией обо всех товарах из каталога. Для получения всех данных из таблицы `Product` можно воспользоваться запросом

```
SELECT * FROM Product
```

Для того чтобы добавить запись в таблицу, можно воспользоваться следующим SQL-запросом:

```
INSERT INTO имя_таблицы (имя_поля1, имя_поля2, ...)
       VALUES (значение1, значение2, ...)
```

При выполнении этого запроса в таблице *имя_таблицы* будет создана новая запись, в которой в соответствующих столбцах будут размещены заданные значения. Например, добавить новый товар в каталог можно с помощью следующего запроса:

```
INSERT INTO Product (Type, Description, Size, Color, Price)
       VALUES ("Тенниска", "100% хлопок", "L", "Черный", 20)
```

Обратите внимание на то, что все строки помещаются в двойные кавычки, а числа — нет. Кроме того, следует заметить, что для каждого указанного поля задано соответствующее значение. В противном случае будет сгенерировано сообщение об ошибке.

В следующем разделе вы узнаете, как сценарии PHP взаимодействуют с базами данных.

Использование PHP для взаимодействия с базами данных

Независимо от типа используемой базы данных для ее использования нужно выполнить следующие шаги.

1. **Подключиться к базе данных.**
2. **Передать системе управления базой данных SQL-запрос с инструкциями.**
3. **Получить и обработать данные.**
4. **Закрыть соединение с базой данных.**

Все эти шаги более подробно рассматриваются в следующих разделах.

Подключение к базе данных

Перед использованием базы данных к ней нужно сначала подключиться. Для этого можно воспользоваться специальными функциями PHP, которым в качестве входных параметров нужно передать следующую информацию.

- ✓ **Местоположение**. База данных может содержаться не только на том же компьютере, где установлен интерпретатор PHP. Поэтому для подключения к базе данных нужно указать имя компьютера, на котором она содержится. В качестве такого имени можно использовать либо доменное имя (например, `mycompany.com`), либо IP-адрес (`172.17.204.2`). Если же база данных расположена на том же компьютере, что и модуль PHP, то в качестве имени узла достаточно указать имя `localhost`.
- ✓ **Регистрационное имя**. Для того чтобы получить доступ к базе данных, необходимо указать корректное регистрационное имя, которое должен предоставить администратор.
- ✓ **Пароль**. Для доступа к базе данных необходимо также указать пароль, который тоже должен быть предоставлен администратором.
- ✓ **Имя базы данных**. Поскольку система управления базами данных может одновременно поддерживать несколько хранилищ, при подключении следует указать имя базы данных, которую вы хотите использовать.

Для повышения уровня защиты параметры соединения с базой данных следует хранить в отдельном файле, который подключался бы в сценариях PHP с помощью функции `include()`. Поскольку включаемые файлы обычно хранятся в безопасном месте (см. главу 8), то и содержащаяся в них информация также будет в безопасности. Например, создадим файл `info.inc` со следующими параметрами подключения к базе данных:

```
$host = "localhost";
$account = "admin";
$password = "secret";
$dbname = "Catalog";
```

После этого во все сценарии, в которых используется база данных, нужно добавить следующую строку:

```
include("info.inc");
```

Для подключения к разным базам данных в языке PHP предназначены различные функции. В противном случае все было бы слишком просто. Однако в то же время следует заметить, что все функции достаточно похожи друг на друга. Так, при разработке Web-приложений наиболее популярной системой управления базами данных по праву считается MySQL. Для подключения к серверу MySQL 4.0 и более ранних версий предназначены функции

```
$connect = mysql_connect($host, $account, $password);
$db = mysql_select_db("Catalog", $connect);
```

Первая функция устанавливает соединение с системой управления базой данных, а вторая позволяет выбрать конкретную базу данных. При взаимодействии с сервером MySQL 4.1 и более поздних версий используются другие функции:

```
$connect = mysqli_connect($host, $account, $password);
$db = mysqli_select_db($connect, "Catalog");
```

При подключении к базе данных некоторые СУБД требуют использования двух отдельных функций (как в случае с MySQL). Например, для установки соединения с серверами mSQL и Sybase используются похожие пары функций:

```
$connect = msql_connect($host, $account, $password);
$db = msql_select_db("Catalog", $connect);

$connect = sybase_connect($host, $account, $password);
$db = sybaseselect_db("Catalog", $connect);
```

При подключении к базе данных другого типа может оказаться достаточно одной функции. Например, для соединения с базой данных PostgreSQL можно воспользоваться функцией

```
$connect = pg_connect("host=$host user=$user
          password=$password dbname=Catalog");
```

При подключении к другим базам данных используется аналогичный синтаксис, хотя и с некоторыми отличиями. Так, для соединения с базой данных Oracle с помощью интерфейса OCI8 предназначена функция

```
$connect = OCILogon($account, $password);
```

При подключении к базе данных Oracle требуется, чтобы была установлена также переменная окружения `ORACLE_SID`.

Функции, которые необходимо использовать для подключения к конкретной базе данных, можно найти в справочном руководстве PHP. Это руководство можно загрузить с Web-узла `www.php.net`.

Передача запросов к базе данных

После установки соединения с базой данных можно выполнять любые действия, связанные, например, с извлечением, изменением или добавлением информации. Эти действия задаются с использованием запросов SQL, передаваемых системе управления базой данных с помощью специальных функций PHP. Как и при подключении к базе данных, эти функции имеют похожий синтаксис, хотя каждая из них имеет и свои особенности. Например, передать SQL-запрос серверу MySQL можно следующим образом:

```
$sql = "SELECT * FROM Product";
$result = mysql_query($sql, $connect);
```

В первой строке SQL-запрос присваивается переменной `$sql`. Этот запрос предназначен для извлечения всех данных из таблицы `Product`. Функция `mysql_query()` передает SQL-запрос `$sql` базе данных, определяемой идентификатором `$connect`. Затем полученные данные сохраняются во временной таблице `1`, указатель на которую содержится в переменной `$result`.

При использовании базы данных PostgreSQL запрос SQL может выглядеть так:

```
$sql = "SELECT * FROM Product";
$result = pg_query($connect, $sql);
```

В первой строке SQL-запрос сохраняется в переменной `$sql`, а во второй строке этот запрос выполняется. В свою очередь, при взаимодействии с сервером Oracle для выполнения аналогичных действий придется вызвать три функции:

```
$sql = "SELECT * FROM Product";
$query = OCIParse($connect, $sql);
$result = OCIExecute($query);
```

В первой строке создается SQL-запрос. (Обратите внимание, что первая строка одинакова для всех трех баз данных.) Далее, для выполнения запроса и получения результата используются две другие функции.

Любой SQL-запрос передается базе данных с использованием одной и той же функции. В трех предыдущих примерах информация извлекалась из базы данных. Ниже приведен пример, в котором осуществляется ввод данных.

```
$first_name = "Джон";
$last_name = "Смит";
```

```
$sql = "INSERT INTO Customer (firstName, lastName) VALUES
                ('$first_name', '$last_name') ";
$result = mysql_query($sql);
```

Если при обработке запроса SQL не было получено никаких данных, переменной `$result` присваивается значение `TRUE`.

Перечень функций, предназначенных для передачи SQL-запросов базе данных, содержится в справочном руководстве PHP. Его можно найти на Web-узле `www.php.net`.

Обработка данных

Если запрос к базе данных предназначен для извлечения информации, то вполне естественно, что после получения данные должны быть каким-то образом обработаны. Например, на их основе на Web-странице можно создавать раскрывающиеся списки, формы со значениями полей по умолчанию и т.д. Для обработки полученной информации ее необходимо сначала извлечь из временной таблицы, в которой она была размещена после выполнения SQL-запроса. Для извлечения данных из временных таблиц в языке PHP предназначены специальные функции.

Временная таблица состоит из строк и столбцов. Функции PHP позволяют извлечь информацию из одной строки этой таблицы и сохранить ее в массиве. При этом имена полей базы данных будут использоваться в качестве ключей. Для MySQL соответствующая функция имеет следующий вид:

```
$row = mysql_fetch_array($result);
```

В примерах из предыдущего раздела результаты выполнения SQL-запроса сохранялись в переменной `$result`. Функция `mysql_fetch_array()` возвращает из временной таблицы (которая определяется переменной `$result`) одну строку данных.

После вызова функции `mysql_fetch_array()` массив `$row` будет содержать такую информацию:

```
$row['firstName'] = Джон
$row['lastName'] = Смит
```

Для обработки всех данных, хранящихся во временной таблице, можно реализовать цикл, в котором на каждой итерации будет извлекаться одна строка. Например, для базы данных PostgreSQL можно воспользоваться циклом

```
while($row=pg_fetch_asoc($result))
{
      foreach($row as $value)
      {
            echo "$value<br>";
      }
}
```

Одна строка временной таблицы извлекается на каждой итерации цикла `while`. При этом на каждой такой итерации создается массив `$row`. В свою очередь, для прохода по массиву `$row` и вывода соответствующих значений используется цикл `foreach`.

В следующем фрагменте для извлечения информации из временной таблицы PostgreSQL используется цикл `for`. При этом достигается аналогичный результат:

```
$Nrows = pg_num_rows($result);
for($i=0; $i<$Nrows; $i++)
{
      $row = pg_fetch_row($result,$i);
      foreach($row as $value)
      {
            echo "$value<br>";
      }
}
```

В приведенном примере функция pg_num_rows() возвращает количество строк временной таблицы. (При работе с MySQL можно воспользоваться аналогичной функцией mysql_num_rows($result).) На каждой итерации цикла for из таблицы выбирается, а затем обрабатывается одна строка. Проход по каждой строке (т.е. по массиву $row) осуществляется с помощью цикла foreach. Обратите внимание на то, что функции pg_fetch_row() передается параметр $i, определяющий номер строки, которую нужно извлечь из таблицы.

При работе с базой данных Oracle можно также выполнить аналогичные действия:

```
$Nfields = OCINumCols($result);
while (OCIFetch($result))
{
      for($i=1; $i<=$Nfields; $i++)
      {
            $value = OCIResult($result, $i);
            echo "$value<br>";
      }
}
```

В данном примере в переменной $Nfields содержится количество полей таблицы, а функция OCIFetch() используется для извлечения из таблицы одной строки. Цикл while выполняется до тех пор, пока не будут обработаны все строки временной таблицы. Функция OCIResult() возвращает значение из одного поля строки, полученной с помощью функции OCIFetch(). Цикл for предназначен для прохода и отображения значений всех полей текущей строки временной таблицы.

В предыдущих примерах приведен минимальный набор функций, которого достаточно для взаимодействия с реляционной базой данных. Однако следует заметить, что в языке PHP для работы с каждой системой управления базами данных имеется свой собственный богатый набор функций. Например, функция mysql_affect_rows() возвращает количество строк, которые были изменены в процессе выполнения запроса, а функция pg_field_name() возвращает имена полей.

Для более детального знакомства с функциями, предназначенными для работы с различными базами данных, обратитесь к справочному руководству PHP, которое можно найти на Web-узле www.php.net. Там же вы найдете и некоторые примеры. Много полезной информации о базах данных MySQL содержится в книге *mySQL*, 2-е издание, которая вышла в Издательском доме "Вильямс".

Закрытие соединения с базой данных

По завершении работы сценария любое открытое соединение с базой данных автоматически закрывается. Однако хорошим стилем программирования является закрытие соединения в самом сценарии (с помощью специальной функции PHP). Это позволит избежать многих потенциальных проблем. Например, закрыть соединение с базой данных MySQL можно следующим образом:

```
mysql_close($connect);
```

Ниже приведены аналогичные функции для других баз данных.

```
ocilogoff($connect);    # Oracle
pg_close($connect);     # PostgreSQL
mssql_close($connect); # MS SQL
```

Обработка ошибок

Установка соединения с базой данных с использованием функций _connect() далеко не всегда завершается успешно. Например, сервер баз данных может оказаться незапущенным, или при подключении могут быть указаны неверные регистрационные данные. Так, при попытке подключения к базе данных MySQL с неверным паролем будет выведено следующее сообщение:

```
Warning: mysql_connect(): Access denied for user:
         'root@localhost' (Using password: YES) in
         c:\test12.php on line 10
(Предупреждение: mysql_connect(): Доступ
         для пользователя запрещен:
         'root@localhost' (Использование пароля: ДА)
         в файле c:\test12.php в строке 10)
```

В приведенном сообщении содержится регистрационное имя и указывается, использовался ли пароль при попытке соединения. При этом сам пароль не отображается. Получение такого сообщения свидетельствует о том, что доступ к базе данных MySQL запрещен, поскольку при установке соединения использовались неверные регистрационные данные.

Это лишь предупреждение, а не сообщение об ошибке. Поэтому после его вывода выполнение сценария продолжится. Однако это далеко не всегда будет корректным. Дело в том, что, если невозможно получить доступ к базе данных, оставшаяся часть сценария может выполняться с ошибками. Поэтому лучше воспользоваться функцией `die()`, которая позволит прервать выполнение сценария в случае неудачной попытки соединения с базой данных. Ниже приведен пример использования функции `die()` при подключении к базе данных PostgreSQL.

```
$connect = pg_connect("host=$host user=$user
           password=$password dbname=Catalog")
       or die("Не удалось подключиться к базе данных");
```

Если попытка подключения к базе данных по каким-то причинам завершилась неудачей, выполнение сценария будет прервано и отобразится сообщение, указанное в качестве параметра функции `die()`.

При извлечении данных из временной таблицы также возможно возникновение различных проблем и, как следствие, вывод предупреждающих сообщений. Например, рассмотрим случай, когда в сценарии содержится следующий фрагмент:

```
$sql = "SELECT * FROM Productt";
$result = mysql_query($sql);
$row = mysql_fetch_array($result);
```

В результате использования неверного имени таблицы (лишний символ `t` в имени `Productt`) отобразится следующее предупреждающее сообщение:

```
Warning: mysql_fetch_array(): supplied argument is not a
         valid MySQL result resource in test.php on line 9
(Предупреждение: mysql_fetch_array():
         используемая переменная является некорректным
         результатом выполнения запроса к базе данных
         MySQL в файле test.php в строке 9)
```

В этом сообщении указано, что в переменной `$result` не содержится корректных данных, т.е. ее нельзя использовать в качестве указателя на временную таблицу с результатами обработки запроса. Обычно подобные сообщения свидетельствуют о том, что SQL-запрос нельзя выполнить корректно или были получены не те данные, которые ожидались. В приведенном примере использование неверного имени таблицы привело к тому, что переменной `$result` было присвоено значение `FALSE`. В свою очередь, передача этого значения в качестве параметра функции `mysql_fetch_array()` может привести (и привела) к получению только неверного результата.

Обратите внимание, что при вызове функции `mysql_query()` не было выведено никакого предупреждающего сообщения, хотя ошибка была допущена именно в самом запросе. Для того чтобы вывести ошибку и прервать выполнение сценария при неверно заданном запросе SQL, можно воспользоваться специальной функцией `mysql_error()`. При ее использовании можно более точно выявить причину и место возникновения ошибки. Для этого предыдущий фрагмент можно переписать так:

```
$sql = "SELECT * FROM Productt";
$result = mysql_query($sql)
          or die("Запрос не выполнен: ".mysql_error());
$row = mysql_fetch_array($result);
```

Это приведет к получению следующего результата:

```
Запрос не выполнен: Table 'Catalog.productt' doesn't exist
(Запрос не выполнен: Таблица 'Catalog.productt' не существует)
```

Если в приведенном фрагменте попытка выполнения запроса завершилась неудачно, функция `die()` выведет сообщение с ошибкой SQL (которая генерируется функцией `mysql_error()`) и прервет выполнение сценария.

Общие процедуры обработок ошибок в сценариях PHP применимы и при работе с базами данных. Например, сообщения об ошибках можно не выводить на экран, а записывать в специальном журнале. (Об обработке ошибок см. в главе 4.)

Собирая все вместе...

В предыдущих разделах рассматривались функции, которые нужно использовать для реализации взаимодействия PHP-приложений с базами данных. В этом разделе все эти функции будут совместно использоваться в полнофункциональном сценарии.

В первом примере используется база данных PostgreSQL с именем `Sales` (Продажи), в которой содержится таблица `Customer` (Покупатель). В этой таблице содержатся имена, фамилии и номера телефонов покупателей. В листинге 12.2 приведен код сценария, отображающего все содержимое таблицы `Customer` на Web-странице.

Листинг 12.2. Сценарий для отображения информации о покупателях

```
<?php
/* Имя сценария: DisplayCustomers
 * Описание:     этот сценарий позволяет извлечь из базы
 *               данных PostgreSQL всю информацию о
 *               покупателях и вывести ее на Web-странице.
 */
include("info.inc"); # содержит параметры соединения с базой данных
$connect = pg_connect("host=$host user=$user
           password=$password dbname=Sales")
       or die("Не удалось подключиться к базе данных");
$sql = "SELECT * from Customer";
$result = pg_query($sql)
          or die("Запрос не выполнен:".mysql_error());
$Nrows = pg_num_rows($result);
echo "<html>
     <head><title>Список покупателей</title></head>
     <body>
     <table width=\"100%\" border=\"0\">\n";
for($i=0;$i<$Nrows;$i++)
{
     echo "<tr>";
     $row = pg_fetch_row($result,$i);
     echo "<td>{$row[1]}, {$row[0]}</td>
          <td>{$row[2]}</td>";
     echo "</tr>\n";
}
echo "</table></body></html>";
?>
```

Этот сценарий извлекает из базы данных `Customer` всю информацию о покупателях, а затем с помощью цикла `for` выводит данные на Web-странице в виде таблицы HTML.

Во втором примере (листинг 12.3) создается группа флажков, с использованием которых пользователь может выбрать вид товара, который он хочет просмотреть. При этом виды товара выбираются из базы данных MySQL. Таблица этой базы данных состоит из двух полей, `ProductType` (Тип товара) и `Description` (Описание).

Листинг 12.3. Сценарий для создания группы флажков

```
<html>
<head><title>Тестирование файлов</title></head>
<body>
<?php
/* Имя сценария: DisplayCheckboxes
 * Описание:     получает информацию из базы данных MySQL и
 *               создает группу флажков на Web-странице.
 */
include("info.inc"); # содержит параметры соединения с базой данных
$connection = mysql_connect($host,$user,$password)
      or die ("Не удалось подключиться к серверу");
$db = mysql_select_db("Catalog",$connection)
      or die ("Не удалось выбрать базу данных");
$query = "SELECT * FROM ProductType";
$result = mysql_query($query)
      or die("Запрос не выполнен: ".mysql_error());
echo "<html><head><title>Товары</title></head>
     <body><div style='margin-left: .2in'>
     <h3>Какой вид товаров Вас интересует?</h3>\n";
## создание формы с флажками ##
echo "<form action='processform.php' method='post'>\n";
echo "<hr><table width='100%'>";
while ($row = mysql_fetch_array($result))
{
     echo "<tr>";
     echo "<td width='20%'><b><input type='checkbox'
       name=\"interest['ProductType']\ "
       value=\"{$row['ProductType']}\">{$row['ProductType']}
       </b></td>\n";
     echo "<td>{$row['Description']}</td>";
     echo "</tr>\n";
}
echo "</table>";
echo "<p><input type='submit' value='Выбрать товар'>
          </form>\n";
?>
</div></body></html>
```

В приведенном сценарии создается форма с HTML-таблицей. В цикле `while` из временной таблицы `$result` выбираются строки, данные из которых затем используются при создании строк HTML-таблицы. Этот цикл завершается после обработки всех строк временной таблицы, полученной в результате выполнения SQL-запроса.

Обратите внимание на вывод элементов массива в ячейках таблицы с использованием фигурных скобок. В противном случае при синтаксическом анализе будет сгенерирована ошибка.

При выполнении сценария из листинга 12.3 будет сгенерирована страница, показанная на рис. 12.1. В группе флажков представлены все виды товаров, информация о которых содержится в базе данных.

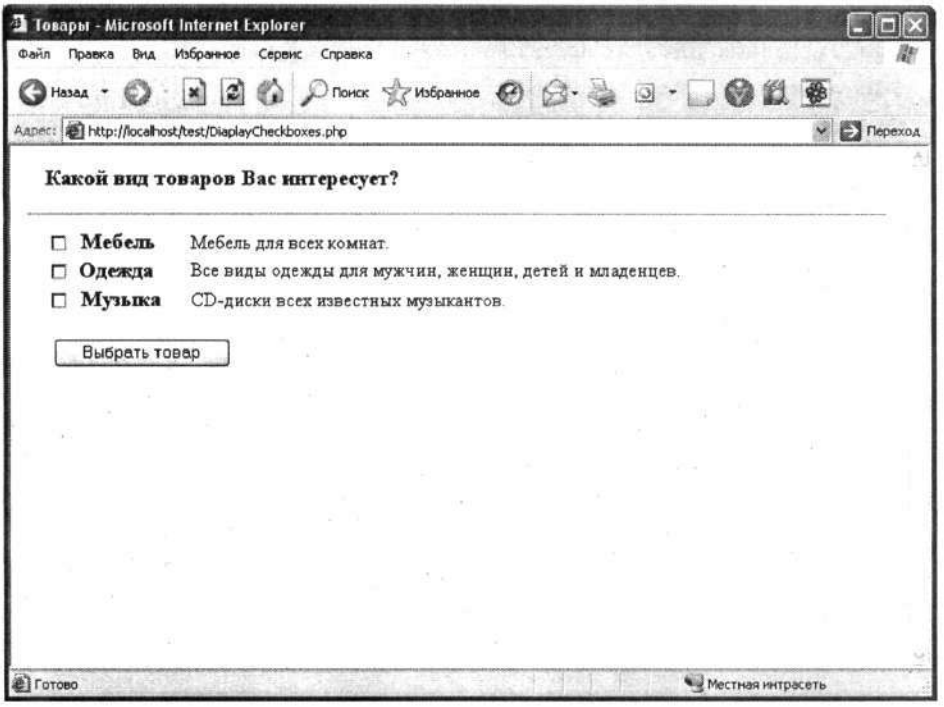


Рис. 12.1. Web-страница с флажками, сгенерированная в результате выполнения сценария из листинга 12.3

Использование расширения SQLite

Расширение SQLite позволяет взаимодействовать с данными, как будто бы они хранятся в базе данных, а не в обычных файлах. При этом не требуется устанавливать никакого специального программного обеспечения, а для извлечения информации можно использовать язык SQL. Расширение SQLite позволяет использовать основные преимущества текстовых файлов, которые связаны с незначительным потреблением вычислительных ресурсов. Кроме того, расширением SQLite предоставляются и преимущества SQL, поскольку нет необходимости изучать команды операционных систем, предназначенные для открытия и манипулирования файлами. В некоторых случаях расширение SQLite обеспечивает и более высокую скорость доступа к данным. Однако расширение SQLite не лишено и некоторых существенных недостатков. К ним следует отнести низкий уровень защищенности информации и невозможность обработки сложных структур данных. Одним словом, можно сказать, что расширение SQLite обеспечивает быстрый и простой доступ к данным, хранящимся в текстовых файлах, однако его нельзя рассматривать как альтернативу баз данных в тех случаях, когда нужно работать с большими объемами информации, сложными структурами данных или обеспечить высокий уровень безопасности.

Методы хранения и извлечения данных с использованием средств SQLite аналогичны методам, которые применяются при работе с базами данных. Для взаимодействия с файлом

данных используется язык SQL, а для передачи SQL-запросов и получения данных — функции PHP. При этом необходимо выполнить аналогичные действия.

1. **Установить соединение с файлом данных.**

   Так же как и при работе с базами данных, сначала нужно установить соединение с файлом данных. Для этого предназначена следующая функция PHP:

   ```
   $db = sqlite_open("testdb");
   ```

   Функция `sqlite_open()` позволяет открыть файл `testdb`. Если указанный файл не существует, будет создан новый файл.

2. **Передать SQL-запрос.**

   Для передачи SQL-запроса предназначена функция `sqlite_query()`.

   ```
   $sql = "SELECT * FROM Product";
   $result = sqlite_query($db, $sql);
   ```

3. **Обработать полученные данные.**

   Как и при взаимодействии с базами данных, полученная в результате обработки SQL-запроса информация сохраняется во временной таблице. В языке PHP имеются функции, массиве. При этом в качестве ключей массива будут использоваться имена полей.

   ```
   $row = sqlite_fetch_array($result);
   ```

   После вызова этой функции в массиве `$row` будет содержаться следующая информация:

   ```
   $row['firstName'] = Джон
   $row['lastName'] = Смит
   ```

   Для обработки всех строк временной таблицы можно воспользоваться циклом `while`.

   ```
   while($row=sqlite_fetch_array($result))
   {
         foreach($row as $value)
         {
               echo "$value<br>";
         }
   }
   ```

4. **Закрыть соединение с файлом данных.**

После завершения работы с файлом его нужно закрыть с помощью функции

```
sqlite_close($db);
```

При использовании расширения SQLite можно применять те же основные принципы обработки ошибок, которые рассматривались в разделе "Обработка ошибок". Например, очень полезной является функция `die()`. В расширении SQLite имеется также функция `sqlite_error()`, которая позволяет генерировать сообщения об ошибках SQLite в случае неудачной обработки SQL-запроса. Например:

```
$sql = "SELECT * FROM Product";
$result = sqlite_query($sql)
          or die("Запрос не выполнен:".sqlite_error());
$row = sqlite_fetch_array($result);
```

Весь материал разделов, посвященных работе с базами данных, можно в полной мере отнести и к расширению SQLite. Единственное отличие заключается в том, что при использовании SQLite информация хранится в простых текстовых файлах, а не в специальных базах данных, которые создаются с помощью специализированного программного обеспечения.

Глава 13

PHP и операционная система

В этой главе...

- Работа с файлами
- Запуск внешних программ
- Передача файлов с одного компьютера на другой
- Отправка электронных сообщений и вложений

Как и любой другой универсальный язык программирования, PHP предоставляет полный набор средств для работы с файлами. С помощью языка PHP с информацией, хранящейся на компьютере, можно выполнять любые действия. Так, файлы можно создавать, копировать, удалять или выполнять их поиск. Язык PHP позволяет также запускать любые программы, пересылать файлы с одного компьютера на другой и передавать информацию по электронной почте. В этой главе рассматриваются все те возможности языка PHP, которые позволяют ощутить реальную мощь современной вычислительной техники.

Управление файлами

Информация, которая хранится на жестком диске, упорядочена в *файлах* (file). В свою очередь, для обеспечения их более эффективной обработки (например, для поиска) файлы распределены по *каталогам* (directory) или *папкам* (folder). Набор файлов и каталогов называется *файловой системой* (file system). Она имеет иерархическую структуру и содержит единственный *корневой* (root) каталог, такой как `C:\` в операционной системе Windows или `/` — в Linux. В корневом каталоге содержатся другие каталоги, каждый из которых также содержит другие каталоги, и т.д. При этом количество вложенных каталогов может быть произвольным.

В общем случае каталог следует рассматривать как специальный файл, используемый для объединения других файлов. В каталоге содержится список файлов и информация, необходимая операционной системе для их поиска. Очевидно, что каталог может содержать как файлы, так и другие каталоги.

В языке PHP имеются функции, которые позволяют открывать файлы и считывать или записывать в них информацию (см. главу 12). Файлы можно копировать, удалять, переименовывать, а также выполнять над ними много других действий. Именно такие функции и рассматриваются в следующих разделах. Кроме того, в этой главе вы узнаете, как управлять каталогами и исследовать их содержимое.

В этой главе рассматривается только часть функций для работы с файлами. Если над файлами или каталогами необходимо выполнить какие-либо действия, обратитесь к справочному руководству и посмотрите, не определена ли в языке PHP соответствующая функция. Если такая функция отсутствует, можно воспользоваться командами операционной системы или программой, написанной на другом языке. Все эти возможности более подробно рассматриваются ниже, в разделе "Использование команд операционной системы".

Получение информации о файле

Зачастую требуется получить какие-нибудь данные о файле. В языке PHP определены функции, которые в сценарии позволяют получить всю необходимую информацию.

Для того чтобы узнать, существует ли файл, предназначена функция `file_exists()`.

```
$result = file_exists("stuff.txt");
```

После вызова этой функции переменной `$result` присваивается либо значение `TRUE`, либо `FALSE`. Обычно функция `file_exists()` используется в условных операторах. Например:

```
if(!file_exists("stuff.txt"))
{
     echo "Файл не найден!\n";
}
```

После того как вы убедитесь в существовании файла, можно приступать к сбору информации о нем.

В табл. 13.1 приведены функции, позволяющие получить полезную информацию о файлах

Таблица 13.1. Функции, предоставляющие информацию о файлах

Функция	Описание	Возвращаемое значение
`is_file("stuff.txt")`	Проверяет, является ли файл обычным файлом, или представляет собой каталог, или любой другой файл специального типа	`TRUE` или `FALSE`
`is_dir("stuff.txt")`	Проверяет, является ли файл каталогом	`TRUE` или `FALSE`
`is_executable("do.txt")`	Проверяет, является ли файл исполняемым	`TRUE` или `FALSE`
`is_writable("stuff.txt")`	Проверяет, можно ли записывать данные в файл	`TRUE` или `FALSE`
`is_readable("stuff.txt")`	Проверяет, можно ли из файла считывать данные	`TRUE` или `FALSE`
`fileatime("stuff.txt")`	Возвращает время последнего доступа к файлу	Время в формате Unix (например, `1057196122`) или `FALSE`
`filectime("stuff.txt")`	Возвращает время создания файла	Время в формате Unix или `FALSE`
`filemtime("stuff.txt")`	Возвращает время последней модификации файла	Время в формате Unix или `FALSE`
`filegroup("stuff.txt")`	Возвращает присвоенный файлу идентификатор группы	Целое число (идентификатор группы) или `FALSE`
`fileowner("stuff.txt")`	Возвращает идентификатор владельца файла	Целое число (идентификатор пользователя) или `FALSE`
`filesize("stuff.txt")`	Возвращает размер файла в байтах	Целое число или `FALSE`
`filetype("stuff.txt")`	Возвращает тип файла	Тип файла (например, `file`, `dir`, `link`, `char`) или `FALSE` при возникновении ошибки или невозможности определения типа
`basename("/t1/do.txt")`	Возвращает имя файла на основе указанного пути	`do.txt`
`dirname("/t1/do.txt")`	Возвращает имя каталога, где размещается файл	`/t1`

Часть информации можно получить лишь в операционных системах Linux/Unix/Mac, другую ее часть — и в системе Windows.

В языке PHP имеется полезная функция `pathinfo()`, которая позволяет получить путь и имя файла. Например:

```
$pinfo = pathinfo("/topdir/nextdir/stuff.txt");
```

После вызова этой функции в массиве `$pinfo` будет содержаться следующая информация:

```
$pinfo[dirname] = /topdir/nextdir
$pinfo[basename] = stuff.txt
$pinfo[extension] = txt
```

При проверке файла с помощью функции вида `is_нечто()` любая ошибка при использовании аргумента, например опечатка в имени файла, приведет к возврату значения `FALSE`. Например, при вызове функции `is_dir("tyme")` будет получено значение `FALSE`, если `tyme` является файлом, а не каталогом. Это же значение будет возвращено, если файла `tyme` не существует.

Некоторые функции, приведенные в таблице, возвращают результат в формате времени Unix. Для преобразования его к стандартному виду даты следует воспользоваться функцией `date()` (см. главу 5).

Копирование, переименование и удаление файлов

В главе 12 рассматривались вопросы создания файлов и записи информации в них. В данном разделе описываются другие возможности работы с файлами, в том числе копирование и удаление.

PHP позволяет скопировать существующий файл в новый и получить два идентичных файла с разными именами. Копирование файлов особенно полезно при создании резервных архивов важной информации. Для этого в PHP используется функция `copy()`.

```
copy("fileold.txt", "filenew.txt");
```

После выполнения этой функции файл `fileold.txt` скопируется в `filenew.txt`. Если же файл с именем `filenew.txt` уже существует, он будет перезаписан. Чтобы избежать этого, следует воспользоваться таким фрагментом кода:

```
if(!file_exists("filenew.txt"))
{
     copy("fileold.txt", "filenew.txt");

}
else
{
     echo "Файл уже существует!\n";
}
```

Для переименования файла предназначена функция `rename()`:

```
rename("oldname.txt", "newname.txt");
```

Если попытаться присвоить файлу уже существующее имя, будет выведено предупреждающее сообщение и файл не будет переименован:

```
Warning: rename(fileold.txt,filenew.txt): File exists in c:\test.php
on line 17
```

Для удаления ненужного файла можно воспользоваться функцией `unlink()`:

```
unlink("badfile.txt");
```

После вызова этой функции файл `badfile.txt` будет удален. Однако следует заметить, что, если файл не существует, никакое сообщение об ошибке не отобразится. Поэтому следует быть внимательным при использовании этой функции.

Организация файлов

Как уже отмечалось, файлы размещаются в *каталогах* (directory), которые также называют *папками* (folder). В этом разделе рассматриваются функции, позволяющие создавать и удалять каталоги, а также получать списки находящихся в них файлов.

Создание каталога

Для создания каталога в языке PHP используется функция `mkdir()`:

```
mkdir("testdir");
```

После вызова этой функции будет создан новый каталог с именем `testdir` в той же папке, где находится и основной сценарий PHP. Например, если сценарий расположен по адресу `/test/test.php`, то новый каталог будет помещен в папку `/test/testdir`. Если попытаться создать каталог с уже существующим именем, будет выведено предупреждающее сообщение следующего вида:

```
Warning: mkdir(): File exists in d:/test/test.php on line 5
```

Для проверки существования каталога с определенным именем можно воспользоваться функцией `is_dir()`. Рассмотрим следующий фрагмент кода:

```
if(!is_dir("mynewdir"))
{
     mkdir("mynewdir");
}
else
{
     echo "Каталог уже существует! ";
}
```

После создания каталога с его содержимым можно выполнять любые действия. Например, можно скопировать или удалить файлы (см. раздел "Копирование, переименование и удаление файлов").

Для создания каталога в другом месте следует задать полный путь к нему. Например:

```
mkdir("/topdir/nextdir/mynewdir");
```

В функции `mkdir()` можно также использовать относительный путь к каталогу, как в следующем примере.

```
mkdir("../mynewdir");
```

Если сценарий находится в каталоге `/topdir/test/makedir.php`, то в результате выполнения этой команды новый каталог будет размещен по адресу `/topdir/mynewdir`.

Для изменения имени каталога в PHP предусмотрена функция следующего вида:

```
chdir("../anotherdir");
```

Получение списка файлов каталога

Очень часто необходимо знать содержимое каталогов. Например, нужно вывести пользователю список предлагаемых для загрузки файлов или извлечь изображения из каталога, чтобы отобразить их в окне браузера.

PHP предоставляет набор разнообразных функций для открытия и чтения каталогов. Рассмотрим следующую строку кода:

```
$dh = opendir("/topdir/testdir");
```

Функция `opendir()` позволяет открыть каталог и, таким образом, получить к нему доступ. Если же попытаться обратиться к несуществующему каталогу, будет выведено следующее предупреждающее сообщение:

```
Warning: opendir(testdir): failed to open dir: Invalid argument in
test13.php on line 5
```

В строке, содержащей вызов функции `opendir()`, переменная `$dh` используется в качестве *дескриптора каталога* (directory handle). Он является своеобразным указателем, который впоследствии можно использовать для обращения к файлам этого каталога. Для считывания имени файла из данного каталога в PHP предназначена функция `readdir()`:

```
$filename = readdir($dh);
```

После ее вызова в переменной `$filename` будет содержаться имя файла, но не полный путь к нему. Чтобы считать все файлы каталога, можно воспользоваться циклом `while`. Например:

```
while($filename = readdir($dh))
{
     echo $filename. "\n";
}
```

Следует также отметить, что при использовании функции `readdir()` порядок считывания файлов может быть произвольным.

Предположим теперь, что на Web-странице необходимо создать "картинную галерею", в которой будут содержаться изображения из определенного каталога. Для этого можно воспользоваться функциями `opendir()` и `readdir()`. В листинге 13.1 приведен сценарий, который позволяет это осуществить.

Листинг 13.1. Сценарий, создающий "картинную галерею"

```
<?php
/*Имя сценария: displayGallery
 *Описание:     отображает все картинки, хранящиеся в
 *              определенном каталоге.
 */
echo "<html><head><title>Галерея изображений</title></head>
      <body>";
$dir = "../test1/testdir/";                                    #8
$dh = opendir($dir);                                           #9
while($filename = readdir($dh))                                #10
{
  $filepath = $dir.$filename;                                  #12
  if(is_file($filepath)and ereg("\.jpg$",$filename))           #13
  {
     $gallery[] == $filepath;
  }
}
sort($gallery);                                                #16
foreach($gallery as $image)                                    #17
{
   echo "<hr>";
   echo "<img src='$image'><br>";
}
?>
</body></html>
```

Обратите внимание на наличие номеров в конце некоторых строк. Рассмотрим код этих строк более подробно.

- ✓ **Строка 8.** Переменной `$dir` присваивается значение пути, которое будет впоследствии использоваться в сценарии. Обратите внимание на наличие косой черты (/) в конце строки, а также на использование этого символа вместо обратной косой (\) (даже в операционной системе Windows).
- ✓ **Строка 9.** В этой строке происходит открытие каталога.
- ✓ **Строка 10.** Начинается цикл `while`, считывающий все файлы из каталога.
- ✓ **Строка 12.** Создается переменная `$filepath`, значение которой соответствует полному пути к файлу.

Если в строке 8 не включить символ /, переменная `$filepath` будет содержать ошибочное значение.

- ✓ **Строка 13.** В этой строке проверяется, является ли файл графическим, т.е. имеет ли он расширение `.jpg`. Если это так, полный путь к изображению добавляется в массив с именем `$gallery`.
- ✓ **Строка 16.** В этой строке осуществляется сортировка значений массива `$gallery` в алфавитном порядке.
- ✓ **Строка 17.** С помощью цикла `foreach` изображения выводятся на Web-страницу.

Использование команд операционной системы

Операционная система содержит большое количество команд, которые пользователь может выполнить из командной строки. Например, если необходимо вывести список файлов каталога или скопировать файл, в Windows можно воспользоваться командами `dir` и `copy`, а в Unix/Linux — `ls` и `cp`. (Для доступа к консольному окну, позволяющему работать с командной строкой в операционной системе Windows 2000, нужно выбрать следующую команду меню: Start⇨Programs⇨Accessories⇨Command Prompt (Пуск⇨Программы⇨Стандартные⇨Командная строка.)

В этом разделе предполагается, что читатель знаком с синтаксисом и назначением системных команд используемой операционной системы. Описание этой области выходит за рамки настоящей книги. В данном разделе рассказывается о том, как запустить команды операционной системы из сценария PHP.

В PHP имеется множество функций, предназначенных для выполнения тех или иных действий. Например, содержимое каталога можно считать с помощью функций `opendir()` и `readdir()` (см. предыдущий раздел), а файлы можно скопировать с использованием функции `copy()`.

Однако в некоторых случаях может потребоваться выполнить некоторые действия, для которых не существует специальных функций PHP. Например, иногда нужно скопировать каталог со всем его содержимым, просмотреть или изменить текущий путь, очистить экран и т.д. В некоторых случаях может оказаться, что проще использовать команды операционной системы, например `ls` или `dir`, чем встроенные функции PHP — `opendir` или `readdir`. Вам также может понадобиться вызвать программу, написанную на другом языке программирования, например на Perl. Нет проблем! Все эти задачи можно легко решить с помощью PHP.

Сообщения об ошибках при вызове системных команд

Ни один из используемых методов PHP для вызова системных команд не поддерживает вывод или возврат сообщений при ошибках. Вы знаете, что системная команда не сработала, поскольку не получили нужный результат. Но что именно не так — неизвестно.

Для вывода сообщений об ошибках при несрабатывании системной команды необходимо добавить несколько символов в строку вызова. Таковыми для большинства операционных систем являются `2>&1`. Рассмотрим следующий пример:

```
$result = system("di c:\php");
```

В данном случае функция `system()` используется для вывода содержимого каталога `c:\php`, но системная команда `dir` написана с ошибкой. Не существует такой команды, как `di`, соответственно ничего выведено не будет. Добавим нужные символы:

```
$result = system("di c:\php 2>&1");
```

В этом случае отобразится сообщение об ошибке. В операционной системе Windows 2000 оно будет иметь следующий вид:

```
'di' is not recognized as an internal or external command, operable
program or batch file
```

Заметьте, что набор символов `2>&1` используется без пробелов.

Для вызова системных команд или внешних программ в PHP предусмотрены следующие методы.

- ✓ **Обратные одинарные кавычки** (` `). Для вызова системной команды ее имя можно заключить в обратные одинарные кавычки, и результат работы отобразится на экране.
- ✓ **Функция** `system()`. Вызывает команду операционной системы, выводит результат выполнения на экран и возвращает последнюю строку вывода.
- ✓ **Функция** `exec()`. Вызывает команду операционной системы, результат выполнения сохраняет в массиве и возвращает последнюю строку результата выполнения.
- ✓ **Функция** `passthru()`. Вызывает системную команду и отображает результат выполнения на экране.

В общем случае PHP позволяет вызывать любые команды. При этом они будут выполняться точно так же, как если бы они были вызваны непосредственно из командной строки операционной системы. Можно вызывать простые команды, такие как `ls` или `dir`, `rename` или `mv`, `rm` или `del`, можно перенаправлять результаты выполнения, вызывать две команды в одной строке и т.д., т.е. вся функциональность, которая поддерживается операционной системой, может быть доступна и в сценариях PHP. Ниже приводятся примеры системных команд, которые можно вызвать из кода PHP.

```
dir
rm badfile.txt
dir | sort
cd c:\php ; dir (Недоступна в Windows)
"cd c:\php && dir" (Для Windows 2000)
dir > dirfile
sort < unsortedfile.txt
```

Иногда процесс выполнения системной команды занимает большой промежуток времени. PHP позволяет вызвать команду в фоновом режиме (если, конечно, используемая операционная система поддерживает его), в то время как сценарий PHP будет продолжать выполняться. В этом случае результат вызова системной команды необходимо перенаправить в отдельный файл, а не сохранять в переменной сценария.

Рассмотрим теперь различные методы вызова команд операционной системы из PHP более подробно.

Использование одинарных кавычек

Самый простой способ вызвать системную команду из PHP — поместить ее в обратные одинарные кавычки. Например:

```
$result = `dir c:\php`;
```

Переменная `$result` будет содержать результат выполнения команды `dir`, т.е. информацию о содержимом каталога `c:\php`:

```
Volume in drive C has no label.
     Volume Serial Number is 394E-15E5

     Directory of c:\php

02/25/2004 10:48a         <DIR>                .
02/25/2004 10:48a         <DIR>                ..
02/25/2004 04:30p         <DIR>                dev
02/25/2004 04:30p         <DIR>                ext
02/25/2004 04:30p         <DIR>                extras
02/25/2004 04:30p                   417,792    fdftk.dll
02/25/2004 04:30p                    90,112    fribidi.dll
02/25/2004 04:30p                   346,624    gds32.dll
02/25/2004 04:30p                        70    go-pear.bat
02/25/2004 04:30p                    32,081    install.txt
02/25/2004 04:30p                   876,544    libeay32.dll
02/25/2004 04:30p                    47,027    libintl-1.dll
02/25/2004 04:30p                   165,643    libmhash.dll
02/25/2004 04:30p                   233,472    libmysql.dll
02/25/2004 04:30p                     3,208    license.txt
02/25/2004 04:30p                    57,344    msql.dll
02/25/2004 04:30p                    18,151    news.txt
02/25/2004 04:30p                   278,800    ntwdblib.dll
02/25/2004 04:30p         <DIR>                PEAR
02/25/2004 04:30p                    53,248    php-cgi.exe
02/25/2004 04:30p                    28,672    php-win.exe
02/25/2004 04:30p                    28,672    php.exe
02/25/2004 04:30p                     3,872    php.gif
02/25/2004 04:30p                    39,284    php.ini-dist
02/25/2004 04:30p                    40,899    php.ini-recommended
02/25/2004 04:30p                    40,960    php5activescript.dll
02/25/2004 04:30p                    36,864    php5apache.dll
02/25/2004 04:30p                    36,864    php5apache2.dll
02/25/2004 04:30p                    53,248    php5apache_hooks.dll
02/25/2004 04:30p                   503,320    php5embed.lib
02/25/2004 04:30p                    28,672    php5isapi.dll
02/25/2004 04:30p                    28,672    php5nsapi.dll
02/25/2004 04:30p             3,452,928 php5ts.dll
```

```
02/25/2004 04:30p                 1,224    snapshot.txt
02/25/2004 04:30p               159,744    ssleay32.dll
02/25/2004 04:30p                49,152    php_mysql.dll
              30 File(s)      7,153,163 bytes
               6 Dir(s)   251,727,872 bytes free
```

Обратные одинарные кавычки можно использовать в том случае, когда режим `safe_mode` отключен, т.е. соответствующая переменная имеет значение `Off` (это значение используется по умолчанию при установке PHP).

Использование функции `system()`

Функция `system()` выполняет системную команду, выводит результат ее выполнения на экран и возвращает последнюю строку вывода. Рассмотрим следующий пример:

```
$result = system("dir c:\php");
```

В результате вызова функции `system()` отобразится содержимое каталога `c:\php`, а переменная `$result` будет содержать последнюю строку вывода:

```
11 Dir(s)          566,263,808 bytes free
```

Использование функции `exec()`

Функция `exec()` выполняет системную команду, но при этом не выводит результат выполнения на экран. Однако он может быть сохранен в массиве, значения элементов которого — строки, сформированные в результате вызова команды операционной системы. Как и функция `system()`, `exec()` возвращает последнюю строку.

Пусть, например, необходимо узнать, сколько файлов и свободного дискового пространства содержится в каталоге. Для этого можно воспользоваться функцией `exec()` следующего вида:

```
$result = exec("dir c:\php");
```

После выполнения этой функции на экран ничего не будет выведено, а переменная `$result` будет содержать последнюю строку результата вызова системной команды `dir`:

```
11 Dir(s)          566,263,808 bytes free
```

Если же необходимо сохранить весь результат выполнения команды `dir`, следует воспользоваться следующим форматом функции `exec()`:

```
$result = exec("dir c:\php", $dirout);
```

В результате выполнения этой функции в массив `$dirout` будут помещены строки с информацией о содержимом каталога `c:\php`. Для вывода всех элементов массива воспользуемся следующим фрагментом кода:

```
foreach($dirout as $line)
{
     echo "$line\n";
}
```

В результате получим следующую информацию:

```
Volume in drive D has no label.
     Volume Serial Number is 394E-15E5

     Directory of d:\php

02/25/2004  10:48a    <DIR>          .
02/25/2004  10:48a    <DIR>          ..
```

```
02/25/2004  04:30p    <DIR>          dev
02/25/2004  04:30p    <DIR>          ext
02/25/2004  04:30p    <DIR>          extras
02/25/2004  04:30p           417,792 fdftk.dll
```

Для вывода значений определенных элементов массива `$dirout` можно воспользоваться следующим фрагментом кода:

```
echo $dirout[3];
echo $dirout[7];
```

В результате будут выведены такие строки:

```
Directory of C:\PHP
02/25/2004  04:30p    <DIR>          dev
```

Использование функции `passthru()`

Функция `passthru()` предназначена для вызова системной команды и отображения результата ее выполнения на экране. Например:

```
passthru("dir c:\php");
```

Она просто выводит результат на экран и ничего не возвращает.

При этом результат выполнения системной команды никак этой функцией не обрабатывается и поэтому может использоваться для вывода двоичных строк.

Вопросы безопасности

Вызывая системную команду, вы позволяете пользователю выполнять определенные действия на вашем компьютере. Например, системная команда `dir c:\php` вполне безобидна, в то время как вызов команды `rm /bin/*` или `del c:\*.*` может привести к непоправимым последствиям. Поэтому следует быть осторожным при использовании функций, позволяющих выполнять команды операционной системы.

Когда вы создаете сценарий PHP, в котором сами выполняете команды типа `dir` или `ls`, — ситуация под контролем. Но если для вызова системных команд используются данные, полученные от пользователей, необходимо соблюдать особую осторожность. Рассмотрим такой пример. Предположим, имеется приложение, которое на основе введенного имени пользователя создает соответствующий каталог. Пусть пользователь в поле `directoryName` некоторой формы ввел свое имя `Smith`. Для обработки введенной информации можно воспользоваться следующим фрагментом кода:

```
$directoryName = $_POST['directoryName'];
exec("mkdir $directoryName");
```

Поскольку переменная `$directoryName` имеет значение `Smith`, то в результате выполнения команды `exec("mkdir $directoryName")` будет создан каталог с именем `Smith`. Казалось бы, все нормально.

Однако предположим, что пользователь ввел в поле формы следующую строку: `Smith; rm *`. В этом случае `$directoryName = Smith; rm *` и будет выполнен такой набор команд: `mkdir Smith; rm *`. Во многих операционных системах, таких как Unix/Linux, символ точки с запятой используется для разделения команд в одной строке, т.е. выражение `mkdir Smith; rm *` эквивалентно

```
mkdir Smith
rm *
```

А вот это действительно проблема. Сначала будет создан каталог с именем `Smith`, но потом все файлы в текущем каталоге будут уничтожены!

Все это говорит о том, что следует быть крайне осторожным при вызове команд, содержащих параметры. Необходимо четко понимать, откуда поступает такая команда. Если это данные, полученные от пользователя, их необходимо проверить на корректность. Так, в предшествующем примере можно было бы проверить значение поля формы, чтобы оно содержало только буквы и числа.

Использование протокола FTP

Передача файлов с одного компьютера на другой — обычное явление в Internet. Если необходимо, чтобы ваш партнер, который находится на значительном расстоянии от вас, получил некоторый файл, это не составит проблем. Этот процесс займет всего несколько секунд.

Файл можно переслать, используя Web-сервер или протокол FTP. Способ передачи файлов с помощью Web-сервера описывался в главе 11. В этом разделе речь пойдет о протоколе FTP.

Протокол FTP не зависит от Web. Это значит, что передачу файла можно осуществить в сценарии, написанном на PHP для Web, или в любом другом сценарии. Протокол FTP позволяет получить содержимое каталога на удаленном компьютере и передать в том или ином направлении один или несколько файлов.

Протокол FTP основан на архитектуре клиент/сервер, т.е. для передачи файлов между вашим и удаленным компьютером необходимо сначала подключиться к серверу FTP, а потом передать запрос.

Язык PHP позволяет обмениваться файлами с помощью протокола FTP. В случае использования операционной системы Windows поддержка FTP включена по умолчанию. При установке PHP в операционных системах Unix/Linux/Mac следует воспользоваться параметром

```
--enable-ftp
```

В приложении A содержится более подробная информация об установочных параметрах PHP.

Подключение к серверу FTP

Для подключения к компьютеру, на котором установлен сервер FTP и с которым будет осуществляться обмен файлами, предназначена функция `ftp_connect()`.

```
$connect = ftp_connect("janet.valade.com");
```

В качестве аргумента этой функции можно также использовать IP-адрес сервера FTP.

```
$connect = ftp_connect("172.17.204.2");
```

После установки соединения с сервером FTP на нем необходимо зарегистрироваться. Для этого потребуется ввести идентификатор пользователя и пароль. При этом можно воспользоваться своими персональными данными либо же использовать общий для всех идентификатор и пароль. Многие Web-узлы именно так и поступают, предоставляя в качестве идентификатора пользователя значение `anonymous`, а в качестве пароля — адрес электронной почты. В общем случае данные пользователя следует хранить в отдельном файле и подключать его по мере необходимости. (О работе с файлами см. в главе 8.)

Для регистрации на сервере FTP в языке PHP предусмотрена функция `ftp_login()`. При этом предполагается, что у пользователя имеются соответствующие идентификатор и пароль.

```
$login_result = ftp_login($connect, $userid, $passwd);
```

Если же соединение с сервером FTP не было установлено предварительно, будет выведено следующее предупреждающее сообщение:

```
Warning: ftp_login() expects parameter 1 to be resource, boolean given in
d:\test1\test13.php on line 9
```

Однако следует заметить, что сценарий PHP в данном случае не остановится и будет продолжать выполняться. Скорее всего, это не то, что вам нужно, поскольку последующий код может использовать соединение FTP. Во избежание данной проблемы можно воспользоваться следующей конструкцией:

```
$connect = ftp_connect("janet.valade.com")
           or die("Не удалось подключиться к серверу");
$login_result = ftp_login($connect,$userid,$passwd)
           or die("Не удалось зарегистрироваться на сервере");
```

Функция `die()` позволяет остановить выполнение сценария в случае наличия ошибки.

После регистрации на сервере FTP ему можно передавать запросы различного вида, например на получение содержимого каталога или загрузку файлов. Все эти возможности рассматриваются в следующих разделах.

Получение содержимого каталога

При использовании протокола FTP одной из основных задач является получение содержимого каталога. Для этого в PHP предусмотрена функция `ftp_nlist()`, которая возвращает массив имен файлов. Ее синтаксис имеет такой вид:

```
$filesArr = ftp_nlist($connect, "data");
```

Второй параметр функции `ftp_nlist()` служит для указания имени каталога. Если же оно неизвестно, можно получить список файлов текущего каталога:

```
$directory_name = ftp_pwd($connect);
$filesArr = ftp_nlist($connect, $directory_name);
```

Для отображения всех файлов каталога можно воспользоваться следующим фрагментом кода:

```
foreach($filesArr as $value)
{
     echo $value\n;
}
```

Передача файлов с помощью протокола FTP

Для получения файла с удаленного компьютера с помощью протокола FTP в PHP предназначена функция `ftp_get()`. Ее синтаксис имеет следующий вид:

```
ftp_get($connect, "newfile.txt", "data.txt", FTP_ASCII);
```

Аргумент `newfile.txt` соответствует имени файла, которое он будет иметь после загрузки на пользовательский компьютер, а `data.txt` — имя существующего файла, находящегося на сервере FTP.

Четвертый параметр означает тип загружаемого файла: `FTP_ASCII` (файл в формате ASCII) или `FTP_BINARY` (двоичный файл). Чтобы определить тип файла, достаточно проанализировать его содержимое. Если он состоит из читаемых символов, то это файл в формате ASCII, в противном случае он является двоичным. Например, двоичными являются графические файлы

Для загрузки файла с пользовательского компьютера на удаленный узел в языке PHP предназначена функция `ftp_put()`. Ее синтаксис имеет следующий вид:

```
ftp_put($connect, "newfile.txt", "data.txt", FTP_ASCII);
```

Параметр `newfile.txt` соответствует имени файла, которое он будет иметь после загрузки на удаленный компьютер, а `data.txt` определяет имя существующего файла, который необходимо отправить на сервер FTP.

После завершения работы с соединением FTP его можно закрыть с помощью функции `ftp_close()`:

```
ftp_close($connect);
```

В качестве примера рассмотрим приведенный в листинге 13.2 сценарий PHP, который позволяет получить из заданного каталога все файлы с расширением `.txt`. Передача файлов осуществляется с использованием протокола FTP.

Листинг 13.2. Сценарий, позволяющий загружать файлы с использованием протокола FTP

```
<?php
/* Имя сценария: downloadFiles
 * Описание:     загружает все файлы каталога
 *               с расширением .txt
 */
$dir_name = "data/";
$connect = ftp_connect("janet.valade.com")
     or die("Не удалось подключиться к FTP-серверу");
$login_result = ftp_login($connect, $userID, $passwd)
     or die("Не удалось зарегистрироваться на FTP-сервере");
$filesArr = ftp_nlist($connect,$dir_name);
foreach($filesArr as $value)
{
     if(ereg("\.txt$", $value))
     {
          if(!file_exists($value))
          {
               ftp_get($connect, $value, $dir_name.$value, FTP_ASCII);
          }
          else
          {
               echo "Файл $value уже существует!\n";
          }
     }
}
ftp_close($connect);
?>
```

В данном сценарии сначала происходит получение списка файлов из каталога на удаленном компьютере, который сохраняется в массиве `$filesArr`. Цикл `foreach` используется для прохода по всем элементам этого массива и проверки, имеет ли файл расширение `.txt`. Если это так, сценарий проверяет, существует ли файл с указанным именем. Если файл уже был загружен с сервера, выводится соответствующее сообщение, в противном случае файл передается на компьютер пользователя.

Другие функции для работы с протоколом FTP

В PHP имеются и другие функции для работы с протоколом FTP, которые позволяют, например, перейти к другому каталогу на удаленном компьютере или создать новый каталог. Некоторые из них приведены в табл. 13.2.

Таблица 13.2. Функции для работы с протоколом FTP

Функция	Описание
`ftp_cdup($connect)`	Осуществляет переход к каталогу верхнего уровня по отношению к текущему
`ftp_chdir($connect, "`*имя_каталога*`")`	Изменяет каталог на удаленном компьютере
`ftp_close($connect)`	Закрывает соединение с сервером FTP

Окончание табл. 13.1

Функция	Описание
`ftp_connect("имя_сервера")`	Инициирует подключение к компьютеру. В качестве параметра *имя_сервера* можно использовать доменное имя или IP-адрес
`ftp_delete($connect, "путь/имя_файла")`	Удаляет файл на удаленном компьютере
`ftp_exec($connect, "команда")`	Выполняет системную команду на удаленном компьютере
`ftp_fget($connect, $fh, "data.txt", FTP_ASCII)`	Загружает содержимое файла, находящегося на удаленном компьютере, в открытый файл с дескриптором `$fh`. (Подробнее об обработке файлов см. в главе 12.)
`ftp_fput($connect, "new.txt", $fh, FTP_ASCII)`	Загружает содержимое открытого на стороне пользователя файла с дескриптором `$fh` на сервер FTP
`ftp_get($connect, "d.txt", "sr.txt", FTP_ASCII)`	Загружает файл, находящийся на удаленном компьютере. `sr.txt` — это имя файла, загружаемого с сервера; `d.txt` — имя результирующего файла на компьютере пользователя
`ftp_login ($connect, $userID, $password)`	Обеспечивает регистрацию на сервере FTP
`ftp_mdtm($connect, "filename.txt")`	Возвращает время последнего изменения файла
`ftp_mkdir ($connect, "имя_каталога")`	Создает новый каталог на удаленном компьютере
`ftp_nlist ($connect, "имя_каталога")`	Возвращает массив, элементами которого являются имена файлов, находящихся в удаленном каталоге
`ftp_put($connect, "d.txt", "sr.txt", FTP_ASCII)`	Загружает файл на удаленный компьютер. `sr.txt` — имя загружаемого файла; `d.txt` — имя файла после загрузки на удаленный компьютер
`ftp_pwd($connect)`	Возвращает имя текущего каталога на удаленном компьютере
`ftp_rename ($connect, "старое_имя", "новое_имя")`	Переименовывает файл, находящийся на удаленном компьютере
`ftp_rmdir($connect, "имя_каталога")`	Удаляет каталог, находящийся на удаленном компьютере
`ftp_size($connect, "filename.txt")`	Определяет размер файла, находящегося на удаленном компьютере
`ftp_systype($connect)`	Возвращает тип операционной системы, используемой файловым сервером, например Unix

Использование электронной почты

Электронная почта повсеместно используется в Internet и в приложениях PHP в частности. Электронные сообщения служат для разных целей. Например, Web-приложение может передать покупателю по электронной почте уведомление о сделанном заказе. Или в случае, когда пользователь создает на Web-узле новую учетную запись, электронное сообщение используется для верификации данных. Если же клиент забыл свой пароль, он может щелкнуть на кнопке I forgot my password (Я забыл свой пароль) и по электронной почте ему будет выслана вся необходимая информация. Электронную почту можно использовать для отправки ежемесячных сообщений подписчикам и т.д.

Язык PHP предоставляет обширный набор функций для работы с электронной почтой. Следующий раздел посвящен вопросам, связанным с отправкой электронных сообщений из сценариев PHP.

Настройка PHP для работы с электронной почтой

Электронное сообщение передается сервером исходящей почты, и для отправки сообщений из сценария PHP он должен знать имя сервера, чтобы иметь к нему доступ.

В большинстве случае сервер исходящей почты использует SMTP (Simple Mail Transfer Protocol — простой протокол передачи электронной почты). При этом не так важно, чем вы пользуетесь: локальной сетью на работе, кабельным модемом дома или услугами провайдера Internet, — все электронные сообщения отправляются сервером SMTP. Причем для доступа к серверу пользователь должен знать его параметры.

Чтобы узнать параметры сервера исходящей почты, можно воспользоваться настройками соответствующего программного обеспечения, которое применяется для отправки электронных сообщений.

Если же вам не удалось определить параметры сервера исходящей почты, следует обратиться к администратору сети, а в случае использования услуг провайдера Internet — к соответствующей компании. Имя почтового сервера может выглядеть следующим образом:

```
mail.ispname.net
```

При использовании операционной системы Linux/Unix именем почтового сервера может быть `sendmail`.

Определившись с параметрами сервера исходящей электронной почты, следует внести соответствующие изменения в конфигурационный файл `php.ini`. Для этого необходимо найти в нем следующие строки:

```
[mail function]
; Только для операционной системы Win32.
SMTP = localhost

; Только для операционной системы Win32.
;sendmail_from = me@localhost.com

; Только для операционной системы Unix.
;Можно также установить дополнительные параметры
; (по умолчанию: "sendmail -t -i").
;sendmail_path =
```

Пользователи операционной системы Windows должны изменить первые два значения. Первое из них соответствует имени сервера исходящей почты. Например:

```
SMTP = mail.ispname.com
```

Второй параметр указывает адрес электронной почты, от имени которого будут отправляться сообщения. Например:

```
sendmail_from = Janet@Valade.com
```

Третий параметр предназначен для пользователей операционной системы Unix. В большинстве случаев используемое по умолчанию значение является корректным. Однако, если что-либо не так, следует обратиться к администратору сети и узнать точные параметры.

Для пользователей операционной системы Unix. Иногда путь к почтовому серверу `sendmail` может иметь вид `/usr/sbin/sendmail` или `/usr/lib/sendmail`. Если же ваша система не использует `sendmail`, то в качестве почтового сервера применяется некоторая программная оболочка. Например, программа Qmail может находиться по адресу `/var/qmail/bin/sendmail` или `/var/qmail/bin/qmail-inject`.

Для того чтобы внесенные в файл `php.ini` изменения вступили в силу, необходимо перезапустить Web-сервер.

Отправка электронных сообщений

Для отправки электронного сообщения из сценария PHP предусмотрена функция `mail()`. Ее синтаксис имеет следующий вид:

```
mail(адрес, тема, сообщение, заголовки);
```

Функция `mail()` имеет следующие параметры:

- ✓ *адрес* (электронный адрес получателя);
- ✓ *тема* (строка, соответствующая теме электронного сообщения);
- ✓ *сообщение* (содержимое электронного сообщения);
- ✓ *заголовки* (строка, содержащая переменные заголовка электронного сообщения).

Рассмотрим такой пример:

```
$to = "janet@valade.com";
$subj = "Тест";
$mess = "Это проверка работы электронной почты";
$headers = "bcc:techsupport@mycompany.com\r\n";
$mailsend = mail($to, $subj, $mess, $headers);
```

В данном фрагменте кода электронное сообщение передается пользователю, адрес которого содержится в переменной `$to`. Причем сообщения можно отправлять нескольким адресатам:

```
$to= "janet@valade.com,me@mycompany.com";
```

Переменная `$headers` в этом примере отправляет скрытую копию сообщения на адрес технической поддержки `techsupport@mycompany.com`. В одном сообщении можно использовать несколько заголовков:

```
$header = "cc:tech@mycompany.com\r\nbcc:sales@mycompany.com";
```

Первые три параметра функции `mail()` являются обязательными, в то время как четвертый, содержащий заголовок, — нет.

Функция `mail()` возвращает значение `TRUE`, если сообщение было отправлено успешно, и `FALSE` — в противном случае. Однако следует заметить, что значение `TRUE` не гарантирует того, что сообщение обязательно дойдет до адресата.

Отправка почтовых вложений

Зачастую для обмена информацией пользователи используют вложения электронных сообщений. Например, если необходимо отправить всем клиентам контракт о сервисном обслуживании на тридцати страницах, куда эффективнее использовать вложение, чем помещать текст контракта в тело сообщения.

Для отправки вложения следует использовать заголовок специального формата. Например:

```
Content-disposition: attachment; filename=test.txt
```

В этом случае заголовок указывает почтовому серверу, что сообщение будет передано вместе с файлом `test.txt`. В следующем примере показано, как отправить короткое сообщение в качестве вложения (хотя вряд ли вам когда-то придется делать это):

```
$to = "janet@valade.com";
$subj = "Проверка использования вложения";
$mess = <<< END
Это тестовое сообщение,
отправляемое в виде вложения.
Проверьте это на практике.
END;
$headers = "Content-disposition: attachment;
filename=test.txt\n";
$headers .= "cc:sales@mycompany.com\n";
$mailsend = mail($to,$subj,$mess,$headers);
```

Сформированное сообщение имеет два заголовка: `Content-disposition` и `cc`. Причем оба они содержатся в переменной `$headers` и заканчиваются символом `\n`. Для некоторых программ электронной почты следует использовать символы `\r\n`.

При отправке электронных сообщений с вложением часто приходится передавать содержимое некоторого файла. Для этого следует считать содержимое файла в переменную, значение которой будет передано в сообщении. Более подробно о работе с файлами см. в главе 12.

При отправке электронного сообщения содержимое файла можно поместить в одну длинную текстовую строку. Это можно сделать с помощью функции `file_get_contents()`:

```
$mess = file_get_contents("имя_файла");
```

В листинге 13.3 приведен сценарий, передающий в качестве вложения текстовый файл.

Листинг 13.3. Сценарий, отправляющий текстовый файл в качестве вложения электронного сообщения

```
<?php
/* Имя сценария: mailTest
 * Описание:     передает текстовый файл в качестве
 *               вложения электронного сообщения.
 */
$filename = "mydata.txt";
$mess = file_get_contents($filename);
$to = "janet@valade.com";
$subj = "Отправка данных в качестве вложения";
$headers = "Content-disposition: attachment; filename=mydata.txt\n";
if(!$mailsend = mail($to, $subj, $mess, $headers))
{
     echo "Почта не отправлена\n";
}
else
{
     echo "Почта отправлена\n";
}
?>
```

В приведенном сценарии с помощью функции `file_get_contents()` содержимое файла считывается в текстовую строку `$mess`. Заголовок сообщения помещается в переменную `$headers`. Отправка сообщения по электронной почте осуществляется с использованием

функции `mail()`. При этом, если не удалось отправить письмо, пользователь получит соответствующее сообщение.

Чтобы отправить во вложении не только текстовые файлы, в заголовке `Content-type` электронного сообщения следует указать соответствующий тип:

```
Content-type: тип_содержимого
```

Так, для простого текстового файла в качестве значения *тип_содержимого* используется `text/plain` (в большинстве случаев это значение используется по умолчанию). Для файла HTML нужно указывать заголовок `text/html`.

Зачастую приходится отправлять двоичные файлы, к которым относятся изображения, аудио- и видеоинформация. Для некоторых из них можно использовать следующие значения заголовка `Content-type`:

```
image/gif
image/jpeg
audio/x-wav
audio/vnd.rn-realaudio
video/mpeg
video/avi
```

Файлы, создаваемые другими приложениями, также могут быть как текстовыми, так и двоичными. Например, файлы в формате RTF являются текстовыми, а файлы Word или Excel — двоичными. В общем случае неизвестно, каким программным обеспечением сгенерирован двоичный файл, поэтому в качестве заголовка `Content-type` лучше использовать значение `application/octet-stream`.

Для того чтобы двоичные файлы дошли по электронной почте в целости и сохранности, их следует зашифровать. Для этого в PHP предназначена функция

```
$mess = chunk_split(base64_encode($string));
```

Переменная `$string` инкапсулирует содержимое двоичного файла, полученное с использованием функции `fread()`.

Кроме того, при передаче закодированного файла в электронном сообщении необходимого указать это, используя заголовок

```
Content-Transfer-Encoding: base64
```

В листинге 13.4 приведен сценарий PHP, позволяющий отправлять графические файлы (двоичные и закодированные) в качестве вложений электронных сообщений.

Листинг 13.4. Сценарий PHP для передачи графических файлов в качестве вложений электронных сообщений

```
<?php
/* Имя сценария: mailGraphic
 * Описание:     передает графические файлы в качестве
 *               вложений электронных сообщений.
 */
$filename = "logo.gif";
$fh = fopen($filename, "rb");
$fileContent = fread($fh, filesize($filename));
fclose($fh);
$mess = chunk_split(base64_encode($fileContent));
$to = "janet@valade.com";
$subj = "Отправка изображения в качестве вложения";
$headers = "Content-disposition: attachment; filename=logo.gif\n";
$headers .= "Content-type: image/gif\n";
$headers .= "Content-Transfer-Encoding: base64\n";
```

```
if(!$mailsend = mail($to, $subj, $mess, $headers))
{
     echo "Почта не отправлена\n";
}
else
{
     echo "Почта отправлена\n";
}
?>
```

В приведенном сценарии закодированное содержимое файла сохраняется в переменной `$mess`. Кроме того, в сообщение включено несколько заголовков, необходимых для передачи такого рода файлов в качестве вложений.

Глава 14

Расширения PHP

В этой главе...

- Что такое расширения PHP
- Установка модуля PEAR
- Использование пакетов модуля PEAR

Архитектура PHP состоит из *ядра* (core), которое определяет базовую функциональность этого языка, и *расширений* (extension).

Следует заметить, что значительная гибкость и мощь языка PHP обеспечивается за счет использования как встроенных функций, так и функций, доступных в разнообразных расширениях. Многие из них не раз встречались в этой книге, а в приложении Б приведен список наиболее популярных функций.

Большинство полезных функций PHP содержится в расширениях, которые значительно увеличивают возможности языка. Часть базовых расширений поставляется и устанавливается по умолчанию. Остальные также включаются в дистрибутив PHP, однако для последующей работы их нужно установить дополнительно. Например, для взаимодействия с различными базами данных необходимо подключить соответствующие расширения (см. главу 12). Кроме того, на данный момент имеется множество расширений, написанных отдельными разработчиками, среди которых, в первую очередь, следует выделить модуль PEAR (PHP Extension and Application Repository — набор расширений и приложений PHP).

В данной главе изложены основные принципы использования расширений PHP для создания приложений.

Основные расширения PHP

В базовом комплекте PHP имеется несколько расширений, которые по умолчанию встроены в модуль PHP, а их функции доступны для использования. При этом нет необходимости что-либо знать об этих расширениях или предпринимать дополнительные усилия для их активизации — нужно просто пользоваться их функциональностью. (Но, если все же они вам не нужны, их можно деактивизировать, как описывается в приложении А.) Многие функции, которые доступны в различных расширениях, встречаются во всей данной книге.

Для того чтобы определить, какие расширения установлены, можно воспользоваться функцией `phpinfo()`. Сначала она выводит параметры ядра PHP, а потом всевозможных расширений. Ниже приведен список расширений, которые устанавливаются по умолчанию.

- ✓ **BCMath**. Математическая библиотека, обеспечивающая большую точность, нежели стандартные числа PHP с плавающей точкой.
- ✓ **calendar.** Библиотека, предоставляющая функции для преобразования дат из одного календарного формата (например, юлианский, григорианский, Французской республики и т.д.) в другой.
- ✓ **COM.** Библиотека, обеспечивающая доступ к объектам COM.

- ✓ **ctype**. Библиотека, предоставляющая функции для проверки типа символов (например, является ли символ буквой или знаком пунктуации).
- ✓ **ftp.** Предоставляет функции для подключения и отправки запросов серверу FTP. Эта библиотека позволяет передавать файлы между компьютерами.
- ✓ **odbc.** Библиотека, предоставляющая функции для работы с интерфейсом ODBC.
- ✓ **pcre.** Это расширение предоставляет функции для работы с регулярными выражениями в формате PCRE.
- ✓ **session.** Предоставляет функции для работы с сеансами PHP.
- ✓ **SQLite**. Предоставляет функции для хранения информации в текстовых файлах с использованием структурированного языка запросов SQL.
- ✓ **tokenizer**. Предоставляет функции, которые предназначены для анализа кода, написанного на языке PHP.
- ✓ **wddx.** Предоставляет функции, предназначенные для работы со стандартом WDDX, который основан на XML и используется для обмена информацией между различными приложениями.
- ✓ **xml.** Библиотека, использующаяся для обработки документов XML.
- ✓ **zlib**. Библиотека, позволяющая считывать и записывать файлы, заархивированные в формате `gzip`.

Приведенный выше список содержит расширения, которые на данный момент устанавливаются при инсталляции PHP по умолчанию. Однако, очевидно, что со временем этот перечень может измениться.

Если для установки PHP под управлением операционной системы Windows используются двоичные файлы, приведенные расширения скомпилированы в соответствующие двоичные файлы. В случае установки PHP в операционной системе Unix или Linux эти расширения будут подключены по умолчанию во время компиляции. Тогда же их можно и деактивизировать. Например, если нет необходимости использовать расширение, обеспечивающее поддержку сеансов PHP, следует воспользоваться специальным параметром установки, как описывается в приложении А.

```
--disable-session
```

В основном встроенные расширения обеспечивают ту функциональность, которая всегда будет востребована разработчиками PHP. Поэтому их приходится отключать очень редко.

В большинстве случаев язык программирования PHP поставляется вместе с дистрибутивом операционной системы Linux, а также входит в комплект поставки программного обеспечения многих компьютеров, использующих ее. Если вы пользуетесь услугами Web-хостинговой компании, то не устанавливаете PHP сами, а поэтому не можете знать, какие именно расширения были подключены во время компиляции. Обычно расширения, приведенные в данной главе, активизированы, но в общем случае нет никакой гарантии. Документацию по этим расширениям можно найти на официальном Web-узле PHP по адресу `www.php.net`.

Стандартные расширения PHP

Многие расширения PHP поставляются вместе с дистрибутивом, однако не все активизируются по умолчанию, т.е. у вас имеются все необходимые файлы, но поддержка тех или иных модулей не обеспечена. Для того чтобы иметь возможность использовать разнообразные функции из расширений, их следует подключить. Как правило, дополнительные расширения используются не так часто, как базовые. Разработчики PHP обеспечивают всю их функциональность, однако не компилируют по умолчанию для экономии ресурсов.

Активизация расширений PHP — задача несложная. Если PHP компилируется при установке, то в этом случае необходимо добавить соответствующий параметр для расширения. (Список таких параметров приведен в приложении А.) Большинство разработчиков, использующих такие операционные системы, как Linux, Unix и Mac, при установке компилируют модуль PHP. И только немногие выполняют ее при использовании операционной системы Windows.

Если имеется набор двоичных файлов PHP, подключить расширение можно следующим образом (без выполнения компиляции).

1. **Скопируйте необходимый файл с расширением** `.dll` **из подкаталога** `ext` **в основной каталог.**

   Подкаталог `ext` может находиться по адресу `c:\php\ext`.

2. **Откройте конфигурационный файл** `php.ini` **и отыщите строку наподобие следующей:**

   ```
   ;extension=php_gd2.dll
   ```

3. **Удалите точку с запятой в начале строки.**

Ниже приведен список наиболее популярных расширений, используемых в PHP.

- ✓ **Расширения для работы с базами данных**. Многие разработчики используют эти расширения для взаимодействия сценария PHP с различными базами данных. Как уже упоминалось ранее, расширение для работы с интерфейсом ODBC включено по умолчанию. (О расширениях для работы с базами данных см. в главе 12.)
- ✓ **Библиотека GD**. Эта библиотека является одной из самых популярных, поскольку предоставляет функции для вывода изображений (как и файлов HTML). Библиотека GD позволяет работать с изображениями в различных форматах, таких как JPEG, GIF, PNG и многих других.

Начиная с версии PHP 4.3.0, в комплект поставки модуля PHP входит встроенная версия библиотеки GD, которую использовать предпочтительнее. При этом нет необходимости подключать внешнюю библиотеку — достаточно скомпилировать PHP с установочным параметром `with-gd2`, как показано в приложении А, и не указывать путь (т.е. не использовать выражение `=DIR`).

- ✓ **Расширение PDF**. Эта библиотека предоставляет набор функций, позволяющих создавать документы в формате PDF, устанавливать шрифты, записывать текст в документ, а также добавлять графику.
- ✓ **Расширение cURL**. Эта библиотека позволяет взаимодействовать с серверами с использованием различных протоколов, таких как HTTPS, Telnet, FTP, LDAP и др.

Назначение большинства расширений — обеспечение взаимодействия PHP с разными программными продуктами. Например, расширения для работы с базами данных предоставляют набор функций для взаимодействия с базами данных в различных форматах. Очевидно, что программное обеспечение, с которым вы планируете работать, должно быть предварительно установлено. (Например, расширение для работы с базой данных Oracle будет функционировать корректно только в том случае, если Oracle будет установлен у вас на компьютере или в сети.) Иногда библиотеки для работы с тем или иным программным обеспечением уже установлены. В противном случае их необходимо предварительно загрузить и установить. При этом сперва следует обратиться к соответствующей документации, находящейся на официальном Web-узле PHP, и убедиться, что это именно то, что вам нужно. Многие расширения поставляются вместе с дистрибутивом PHP, так что нет необходимости отыскивать их

и загружать дополнительно. Такие библиотеки находятся в основном каталоге, где установлен модуль PHP. Например, для поддержки базы данных MySQL требуется наличие файлов `libmysql.dll` (для версий MySQL 4.0 и ниже) и `libmysqli.dll` (для версий MySQL 4.1 и выше). Эти файлы помещаются в основной каталог PHP, тем самым обеспечивая корректность работы функций `mysql()` и `mysqli()`.

В табл. 14.1 приведен практически полный список расширений, используемых в PHP, за исключением расширений для работы с базами данных, которые рассматривались в главе 12.

В данной таблице все имена файлов с расширением `.dll`, которые находятся в каталоге `php/ext`, начинаются с символов `php_`. Параметр `DIR` указывает путь к каталогу, где установлена библиотека. Если же он опущен, используется значение по умолчанию.

Таблица 14.1. Расширения PHP

Название расширения	Назначение	Имя файла dll	Установочный параметр
bzip2 Compression	Считывает/записывает файлы в формате bzip2	`_bz2.dll`	`--with-bz2=DIR`
ClibPDF	Создает документы в формате PDF	`_cpdf.dll`	`--with-cpdflib=DIR`
Crack	Проверяет пароль на устойчивость	`_crack.dll`	`--with-crack=DIR`
cURL	Обеспечивает работу с различными протоколами	`_curl.dll`	`--with-curl=DIR`
Domxml	Обрабатывает документы XML	`_domxml.dll`	`--with-dom=DIR`
FDF	Обрабатывает данные формы документа PDF	`_fdf.dll`	`--with-fdftk=DIR`
GD	Работает с изображениями	`_gd2.dll`	`--with-gd2`
gettext	Поддерживает естественный язык	`_gettext.dll`	`--with-gettext=DIR`
iconv	Преобразует тип кодировки символов строки	`_iconv.dll`	`--with-iconv=DIR`
IMAP	Поддерживает функции для работы с почтовым сервером IMAP	`_imap.dll`	`--with-imap=DIR`
JAVA	Поддерживает язык Java	`_java.dll`	`--with-java=DIR`
LDAP	Поддерживает функции для работы со службой каталогов LDAP	`_ldap.dll`	`--with-ldap=DIR`
Multi-byte String	Поддерживает японские (по умолчанию) и другие символы	`_mbstring.dll`	`--with-mbstring=LANG`
Mcrypt Encryption	Шифрует данные	`_mcrypt.dll`	`--with-mcrypt=DIR`
Mhash	Используется для вычисления контрольных сумм и т.д.	`_mhash.dll`	`--with-mhash=DIR`

Окончание табл. 14.1

Название расширения	Назначение	Имя файла dll	Установочный параметр
Mime type	Определяет тип содержимого документа	_mime_magic.dll	--with-mime_magic
Ming for Flash	Создает анимацию в формате Flash	_ming.dll	--with-ming
OpenSSL	Предоставляет безопасный протокол передачи данных	_openssl.dll	--with-openssl=DIR
PDF	Создает документы в формате PDF	_pdf.dll	--with-pdflib=DIR
Printer	Отправляет данные на печать (только для операционной системы Windows)	_printer.dll	Отсутствует
Shared Memory	Считывает/записывает разделяемые сегменты памяти	_shmop.dll	--enable-shmop
SNMP	Управляет объектами SNMP	_snmp.dll	--with-snmp=DIR
Sockets	Интерфейс нижнего уровня для взаимодействия с сокетами	_sockets.dll	--enable-sockets
XML-RPC	Создает серверы и клиенты XML-RPC	_xmlrpc.dll	--with-xmlrpc=DIR
XSLT	Работет с языком XSLT	_xslt.dll	--enable-xslt
Zip Files	Считывает файлы в формате zip	_zip.dll	--with-zip.dll

Очевидно, что в любой момент эти расширения могут быть исключены из этого списка и добавлены новые.

Использование модуля PEAR

Модуль PEAR (PHP Extension and Application Repository — набор расширений и приложений PHP) представляет собой структурированную библиотеку программ с открытым кодом. Дело в том, что сам программный код поставляется отдельными разработчиками (вне PEAR), а консорциум разработчиков PEAR управляет его содержимым и распространением.

Консорциум разработчиков PEAR создал метод распространения кода в так называемых *пакетах* (package). Разработчик, желающий создать расширение или приложение, должен придерживаться определенных стандартов. Например, пакет должен содержать такие компоненты, как документацию, специальные функции для обработки ошибок, а сам программный код должен отвечать требованиям PEAR. Использование стандартизированной структуры кода позволяет сделать его поддержку на протяжении значительного периода времени более эффективной. Действительно, если разработчик PEAR выиграл в лотерею и улетел на отдых на Таити, то любой другой программист может взяться за поддержку "покинутого" кода, поскольку его структура прозрачна.

Прежде чем некоторый проект будет добавлен в расширение PEAR, его принимают на рассмотрение. В PEAR добавляют только код, написанный на высоком профессиональном уровне, который удовлетворяет всем стандартам. При этом разработчик, желающий видеть свое

"творение" в модуле PEAR, должен поддерживать программный код и предоставлять соответствующую документацию. После рассмотрения разработчиками PEAR предоставленного проекта он добавляется в расширение PEAR, а его автор назначается ответственным за его поддержку.

Где можно найти модуль PEAR

По данным официального Web-узла PEAR (`http://pear.php.net`) этот модуль содержит 263 пакета, шесть из которых включены в дистрибутив PHP по умолчанию. К ним относятся следующие.

- ✓ **DB**. Представляет абстрактный уровень для работы с различными базами данных, используя один и тот же набор функций.
- ✓ **Net_Socket**. Представляет собой интерфейс Net Socket для работы с сокетами TCP.
- ✓ **Net_SMTP**. Представляет реализацию протокола SMTP (Simple Mail Transfer Protocol — простой протокол передачи почты) на основе интерфейса Net_Socket.
- ✓ **Mail**. Предоставляет набор различных функций для проверки и отправки электронной почты.
- ✓ **XML_Parser**. Представляет собой анализатор документов XML, который основан на использовании встроенного в PHP расширения XML.
- ✓ **PHPUnit**. Предназначен для проверки функций и классов, написанных на PHP.

Эти пакеты устанавливаются в каталог `PEAR/packages` либо же в `PEAR/go-pear-bundle`, расположенный в главном каталоге PHP. Причем все файлы заархивированы с помощью архиватора ZIP. Для установки необходимых пакетов можно воспользоваться установочной программой PEAR, которая описана в следующем разделе.

Описание всех пакетов PEAR можно найти на официальном Web-узле PEAR (`pear.php.net`). При этом нужные пакеты можно выбрать из соответствующих категорий или отыскать в базе данных.

Обратите внимание на две гиперссылки в левой части главной страницы: List Packages (Перечень пакетов) и Search Packages (Поиск пакетов). После щелчка на ссылке List Packages отобразится следующий список категорий.

Authentication
Benchmarking
Caching
Configuration
Console
Database
Date and Time
Encryption
File Formats
File System
Gtk Components
HTML
HTTP
Images
Internationalization
Logging
Mail
Math
Networking
Numbers
Payment
PEAR
PHP
Processing
Science
Streams
Structures
Text
Tools and Utilities
Web Services
XML

После щелчка на одной из категорий появится список относящихся к ней пакетов. В свою очередь, после щелчка на пакете отобразится страница с разнообразной информацией о нем. Так, раздел Dependencies (Зависимости) содержит перечень пакетов, которые необходимо иметь для установки выбранного вами. Раздел Download (Загрузка) содержит информацию о том, откуда можно загрузить данный пакет. Однако не стоит делать это вручную, поскольку установочная программа PEAR сделает все автоматически.

После щелчка на ссылке Search Packages появится форма, позволяющая выполнять поиск пакетов по названию, категории, по данным о разработчике и дате. В результате поиска отобразится страница со ссылками на пакеты, удовлетворяющие заданному запросу. После щелчка на одной из них появится информация о найденном пакете. Если ее окажется недостаточно, можно обратиться к руководству по использованию. Для этого необходимо щелкнуть на ссылке Manual, находящейся на левой панели страницы в разделе Documentation (Документация). После выбора языка появится содержание руководства. Раздел IV. Packages (Пакеты) содержит ссылки на категории, а те, в свою очередь, — на пакеты, содержащиеся в них. Щелчок на ссылке с именем пакета приведет к отображению страницы с полной информацией о нем.

Для того чтобы перейти с любой страницы Web-узла PEAR на главную, щелкните на ссылке PEAR в левом верхнем углу страницы.

Определившись с набором необходимых пакетов, можно перейти к их установке.

Установка модуля PEAR

Для каждого пакета PEAR существует *менеджер пакета* (package manager), обеспечивающий его администрирование и включенный в комплект поставки PHP. Установочная программа PEAR может устанавливать, удалять или обновлять пакеты. Поскольку она содержит все данные о пакетах, то позволяет выводить информацию о доступных пакетах, проверять ее, проверять зависимости между пакетами и выполнять другие действия по их администрированию.

Установка модуля PEAR при компиляции PHP

При компиляции PHP модуль PEAR и упомянутые выше шесть пакетов устанавливаются автоматически. Чтобы проверить работоспособность установочной программы PEAR, можно запустить файл `pear.php` из корневого каталога PHP. Для этого наберите в командной строке `pear`, чтобы отобразился список возможных параметров. Для того чтобы вывести список всех доступных пакетов, введите `pear list-all`. (Этот процесс может занять несколько минут, поскольку список загружает из Internet.)

Для версий PHP ниже 4.3.0 установочная программа PEAR не добавляется автоматически при компиляции PHP. Однако на Web-узле `go-pear.net` находится сценарий, позволяющий выполнять загрузку и установку компонентов модуля PEAR.

Установка модуля PEAR в операционной системе Windows

При установке модуля PHP с помощью установочной программы или файлов ZIP модуль PEAR автоматически не добавляется. Для установки этого расширения в операционной системе Windows необходимо выполнить следующие действия.

1. **Загрузите компоненты PEAR**. Для установки вручную загрузите файлы ZIP, предназначенные для операционной системы Windows, как описано в приложении A. Затем разархивируйте эти файлы в корневой каталог PHP, например `c:\php\pear`. Если вы использовали инструкции по установке, описанные в приложении A, все необходимые

компоненты модуля PEAR будут помещены именно туда. При этом устанавливать что-либо дополнительно не придется.

2. **Добавьте путь к каталогу PEAR в конфигурационном файле.** Откройте файл `php.ini` и найдите следующую строку:

   ```
   include_path = ".; c:\php\includes; c:\php\pear"
   ```

   Если эта строка не содержит путь к каталогу, где установлен модуль PEAR, его следует добавить.

3. **Загрузите и установите установочную программу PEAR.** Запустите файл `go-pear.php` (или `go-pear.bat`) в каталоге, где установлен модуль PHP. В результате выполнения этого сценария появится окно командной строки со следующим содержимым:

   ```
   Welcome to go-pear!

   Go-pear will install the 'pear' command and all the files
   needed by it. This command is your tool for PEAR installation
   and maintenance.

   Go-pear also lets you download and install the PEAR packages
   bundled with PHP: DB, Net_Socket, Net_SMPT, Mail, XML_Parser,
   PHPUnit.

   If you wish to abort, press Control-C now, or press Enter:

   (Добро пожаловать в go-pear!

   Go-pear установит команду 'pear' и все необходимые файлы. Ко-
   манда 'pear' является основным средством для установки и под-
   держки пакета PEAR.

   Go-pear также позволяет загрузить и установить пакеты PEAR для
   PHP: DB, Net_Socket, Net_SMPT, Mail, XML_Parser, PHPUnit.

   Для отмены нажмите комбинацию клавиш <Ctrl-C> или клавишу Enter:)
   ```

Вам предстоит ответить на несколько вопросов, и установочная программа PEAR будет установлена. При этом предоставляется возможность установить шесть пакетов, которые включены в дистрибутив по умолчанию.

Код сценария `go-pear.bat` содержит строку, в которой указывается путь к интерфейсу PHP CLI. В предыдущих версиях PHP этот интерфейс размещался в каталоге `/cli`. Если же строка содержит старый путь,

```
Set PHP_BIN = cli/php.exe
```

сценарий `go-pear` работать не будет. В этом случае необходимо изменить это значение на корректное.

После инсталляции установочной программы PEAR (и шести пакетов PEAR) в корневом каталоге PHP появится файл `pear.bat`. Для проверки его работоспособности наберите в командной строке `pear`, чтобы появился список возможных параметров. Для того чтобы вывести список всех доступных пакетов, введите `pear list-all`. (Этот процесс может занять несколько минут, поскольку список загружается из Internet.)

Параметр `list-all` позволяет получить доступ ко всем пакетам PEAR. При этом необходимо помнить точное название пакета, чтобы установить именно ту программу, которую вы хотели.

Установка пакета PEAR

Для установки нового пакета PEAR можно воспользоваться установочной программой PEAR (см. предыдущий раздел). Для получения полного списка всех доступных пакетов следует использовать параметр `list-all` при запуске файла `pear.bat`. В данном разделе будет показано, как добавлять пакеты, используя установочную программу PEAR.

Для установки пакета PEAR введите в командной строке следующее (в корневом каталоге PHP `c:\php`):

```
pear install имя_пакета
```

Установочная программа PEAR автоматически загрузит необходимые файлы пакета с официального Web-узла PEAR и установит его. Например, чтобы добавить пакет Mail, выполните команду

```
pear install Mail
```

При этом отобразится следующая информация:

```
downloading Mail-1.0.2.tgz ...
...done: 12,287 bytes
requires package 'Net_SMTP'
Mail: dependencies failed

(загрузка Mail-1.0.2.tgz ...
...завершено: 12,287 байт
требуется пакет 'Net_SMTP'
Mail: требуемые пакеты не найдены)
```

В этом сообщении говорится о том, что пакет Mail был успешно загружен. Однако для его функционирования требуется наличие пакета Net_SMTP, который не установлен. Соответственно пакет Mail также не будет установлен. Сначала следует добавить в модуль PEAR пакет Net_SMTP, а только потом можно устанавливать Mail.

Если выполнить требования PEAR и корректно установить пакет Mail, а затем попытаться добавить его еще раз, отобразится сообщение

```
Mail already installed

(Пакет Mail уже установлен)
```

Для того чтобы вывести список всех установленных пакетов, необходимо воспользоваться такой командой:

```
pear list
```

Для обновления пакета предназначена команда

```
pear update Mail
```

В этом случае отобразится следующая информация:

```
downloading Mail-1.0.2.tgz ...
...done: 12,287 bytes
upgrade to a newer version (1.0.2 is not newer that 1.0.2)

(загрузка Mail-1.0.2.tgz ...
...завершено: 12,287 байт
установка новой версии (1.0.2 не является более
новой версией))
```

Если попытаться установить пакет PEAR без установки соединения с сетью Internet, отобразится строка следующего вида:

```
Connection to pear.php.net:80 failed
(Не удается установить соединение с сервером pear.php.net:80)
```

Для удаления пакета из модуля PEAR используется команда

```
pear uninstall Mail
```

В результате отобразится такое сообщение:

```
uninstall ok: Mail
(удаление пакета Mail завершено)
```

Использование пакетов PEAR

После добавления пакета в модуль PEAR можно использовать его функции и методы.

Доступ к пакету PEAR

После установки пакета в корневом каталоге PEAR появится файл с расширением `.php`, имя которого соответствует названию пакета. Например, при добавлении пакета Mail будет создан файл `Mail.php`. Для того чтобы воспользоваться функциональностью пакета, в своем сценарии необходимо подключить этот файл. Рассмотрим следующий пример:

```
require_once("Mail.php");
```

В данной строке кода функция `require_once()` используется по двум причинам. Во-первых, файл `Mail.php` будет подключен к сценарию только один раз. Во-вторых, если он не будет найден, выполнение сценария прекратится.

Более подробное описание пакетов PEAR можно найти на официальном Web-узле PEAR.

Пакет DB является одним из наиболее популярных в расширении PEAR. Поэтому в качестве примера демонстрации возможностей PEAR в последующих разделах будет описан именно он. Автор не претендует на полноту изложения информации о данном пакете — будут продемонстрированы лишь основные принципы создания сценариев для взаимодействия с базами данных с помощью DB.

Пример пакета: PEAR — DB

Пакет DB предназначен для взаимодействия с различными базами данных с помощью одних и тех же функций. В главе 12 было показано, что для работы с каждой базой данных в PHP предусмотрен свой набор функций, например `mysql_connect()` или `pg_connect()`. Вроде бы все просто.

Однако проблемы возникают при смене базы данных. Предположим, что ваше приложение взаимодействует с базой данных MySQL. Однажды утром к вам заходит начальник и говорит: "Я только что приобрел новую базу данных Oracle, и все данные необходимо переместить туда". При этом вам понадобится не только переместить данные, но и найти в сценарии PHP все функции, которые предназначены для работы с MySQL, а затем заменить их на соответствующие функции для Oracle. Причем одни из этих функций будут иметь иные названия, а другие — отличный список аргументов. Вот это — настоящая проблема.

Если у вас есть стопроцентная уверенность в том, что замена базы данных никогда не произойдет, то можно об этом и не волноваться. В противном случае следует использовать функции, которые будут работать независимо от имеющейся базы данных.

Взаимодействие с базой данных

Для обеспечения взаимодействия с базой данных с помощью пакета PEAR DB выполняются те же действия, которые описаны в главе 12.

1. **Подключение к базе данных.**
2. **Отправка запросов для выполнения тех или иных действий.**
3. **Обработка информации, полученной в результате выполнения запроса.**

Для подключения к базе данных используется такая информация, как имя учетной записи, пароль, тип базы данных и т.д. Эта информация, помещенная в строку определенного формата, составляет имя источника данных DSN (Data Source Name). Используя его, DB осуществляет подключение к базе данных. Рассмотрим следующий пример:

```
$host = "localhost";
$dbuser = "admin";
$dbpasswd = "secret";
$dbname = "Sales";
$dbtype = "mysql";
$dsn = "$dbtype://$dbuser:$dbpasswd@$host/$dbname";
```

В переменной `$dsn` содержится список параметров, необходимых для корректного подключения к базе данных.

```
$db = DB::connect($dsn);
```

После установки соединения с базой данных к ней можно обращаться с SQL-запросами. Например:

```
$sql = "SELECT * from Sales";
$result = $db->query($sql);
```

Если в результате выполнения SQL-запроса будет получена некоторая информация, для доступа к ней следует воспользоваться функцией

```
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
```

В этом фрагменте кода метод `fetchRow()` помещает строку результата выполнения SQL-запроса в ассоциативный массив, ключами которого являются имена полей.

При этом очевидно, что при изменении базы данных вам придется лишь изменить значение переменной `$dbtype`. Вносить какие-либо другие изменения в сценарий не потребуется. Таким образом, немного дополнительных усилий при написании сценария позволят избежать больших хлопот, связанных с переходом к новой СУБД. На данный момент пакет PEAR DB поддерживает следующие типы баз данных.

- ✓ `mysql` (MySQL)
- ✓ `pgsql` (PostgreSQL)
- ✓ `ibase` (InterBase)
- ✓ `msql` (Mini SQL)
- ✓ `mssql` (Microsoft SQL Server)
- ✓ `oci8` (Oracle версий 7, 8, 8I)
- ✓ `odbc` (ODBC)
- ✓ `sybase` (Sybase)
- ✓ `ifx` (Informix)
- ✓ `fbsql` (FrontBase)

Например, для перехода от СУБД MySQL к Sybase следует заменить строку `$dbtype = "mysql";` на `$dbtype = "sybase";`.

Обработка ошибок

В главе 12 уже обсуждались ошибки, связанные с взаимодействием с базами данных. Например, СУБД может оказаться недоступной при попытке подключения к ней из сценария. В этом случае необходимо, чтобы сценарий остановился и вывел соответствующее сообщение об ошибке.

В пакете DB существует метод обработки ошибок `isError()`. Он проверяет значение переменной: либо оно равно `TRUE`, либо переменная содержит ошибку. В этом случае для вывода сообщения об ошибке можно воспользоваться методом `getMessage()`. Рассмотрим следующий пример:

```
$db = DB::connect($dsn);
if (DB::isError($db))
{
     die($db->getMessage());
}
```

В данном фрагменте кода метод `isError()` используется для проверки значения переменной `$db`. Если оно равно `TRUE`, т.е. подключение к базе данных прошло успешно, блок инструкций оператора `if` не выполняется. Если же переменная `$db` содержит ошибку, будет вызвана функция `die()`, которая остановит выполнение сценария и выведет сообщение, сгенерированное методом `isError()`. Например, если ошибка содержится в имени источника данных DSN, отобразится сообщение следующего вида:

```
DB Error: no such database
(Ошибка DB: база данных не существует)
```

Собирая все вместе

В листинге 14.1 приведен сценарий, предназначенный для вывода списка покупателей, информация о которых хранится в базе данных. Этот сценарий уже был реализован в главе 12 с использованием стандартного набора функций PHP для работы с базой данных PostgreSQL. В данном случае будут задействованы функции из пакета PEAR DB.

Листинг 14.1. Сценарий, выводящий список покупателей

```
<?php
/* Имя сценария: DisplayCustomers
 * Описание:     этот сценарий извлекает из базы данных
 *               информацию о покупателях и выводит ее
 *               на Web-странице.
 */
require_once("DB.php"); #подключает классы пакета PEAR DB
$host = "localhost";
$dbuser = "admin";
$dbpasswd = "secret";
$dbname = "Sales";
$dbtype = "pgsql";
$dsn = "$dbtype://$dbuser:$dbpasswd@$host/$dbname";
$db = DB::connect($dsn);
if (DB::isError($db))
{
     die($db->getMessage());
}
$sql = "SELECT * from Customer";
$result = $db->query($sql);
if (DB::isError($result))
{
     die($result->getMessage());
```

```
}
echo "<html>
      <head><title>Список покупателей</title></head>
      <body>
      <table width=\"100%\" border=\"0\">\n";
while($row = $result->fetchRow(DB_FETCHMODE_ASSOC))
{
      if (DB::isError($row))
      {
            die($row->getMessage());
      }
      echo "<tr>";
      echo "<td>{$row['lastname']}, {$row['firstname']}</td>
            <td>{$row['phone']}</td>";
      echo "</tr>\n";
      }
echo "</table></body></html>";
?>
```

В результате выполнения этого сценария будет выведен список покупателей в следующем формате:

Фамилия, имя номер телефона

Безусловным преимуществом использования пакетов PEAR является простота доступа к ним. Программа установки PEAR инсталлирует все необходимые пакеты, которые могут быть подключены в любой сценарий с помощью функции `require_once()`. Однако необходимо заметить, что каждый пакет PEAR имеет свой уникальный набор классов и функций для решения тех или иных задач. Изучение нового пакета в какой-то мере равносильно изучению нового языка программирования. При этом ранее накопленные знания о PHP могут в этом не помочь. И хотя многие пакеты PEAR имеют достаточно полную документацию, некоторые из них описаны не в полной мере.

Часть V

Великолепные десятки

"У тебя никогда не возникало такого чувства, что этот проект в любой момент может как возродиться, так и прекратить свое существование?"

В этой части...

Главы, содержащиеся в данной части, позволят вам расширить знания о языке PHP. Здесь будут даны рекомендации и предостережения для программистов на этом языке. Также будет рассказано о Web-узлах, посетив которые можно найти много полезной информации о PHP. PHP — быстро развивающийся и расширяемый язык программирования, поэтому вам, как программисту на PHP, следует не отставать от темпов его развития.

Глава 15

Десять правил, которых следует придерживаться при разработке сценариев на PHP

В этой главе...

- Описание наиболее распространенных ошибок, возникающих при написании сценариев PHP
- Анализ сообщений об ошибках

Наверное, все те ошибки, которые описываются в данной главе, так или иначе допускали и вы. Невозможно написать сценарий, не допустив при этом никаких ошибок. Однако, чтобы это не происходило слишком часто, сначала следует научиться их выявлять, а затем, взглянув на код сценария, быстро исправить. Наверняка вам часто доводилось получать подобное сообщение:

```
Parse error: parse error in c:\test.php on line 7
```

В этом случае интерпретатор PHP указывает, что строка кода содержит ошибку. При этом выводится имя сценария и номер строки, где произошла ошибка, что очень помогает при их устранении. Однако, как вы узнаете позже, интерпретатор PHP не всегда реагирует на ошибки путем вывода соответствующих сообщений.

В данной главе рассматриваются наиболее распространенные типы ошибок при создании сценариев на PHP и приводятся рекомендации по их предотвращению.

Отсутствие точки с запятой

Каждый оператор PHP должен завершаться символом точки с запятой (;). Интерпретатор PHP считывает содержимое строк до тех пор, пока не встретит этот символ. Рассмотрим следующий пример:

```
$test = 1
echo $test;
```

Приведенный фрагмент кода не имеет смысла, поскольку содержимое двух строк на языке PHP будет интерпретироваться как один оператор, что приведет к выводу следующего сообщения об ошибке:

```
Parse error: parse error in c:\test.php on line 2
```

Ошибка, связанная с отсутствием точки с запятой, является достаточно распространенной.

Недостаточное количество знаков равенства

При сравнении двух величин необходимо использовать два расположенных подряд символа равенства (==). Одна из наиболее распространенных ошибок — использование одного такого символа. Эта ошибка легко объяснима: еще с первого класса вы научились использовать один знак равенства для таких выражений, как 2+2=4. Эту ошибку не так уж легко выявить, поскольку PHP никаких сообщений об этом не выводит. Единственное, что при этом наблюдается, — это странное поведение сценария (например, наличие бесконечных циклов или невыполнение инструкций условного оператора `if`). Можно долго наслаждаться выполнением следующего фрагмента кода, который содержит бесконечный цикл:

```
$test = 0;
while ($test = 0)
{
      $test++;
}
```

Опечатка в имени переменной

Это еще один тип ошибок, который не приводит к выводу никаких сообщений. Если в имени переменной сделана опечатка, то интерпретатор PHP воспринимает переменную как только что созданную и выполняет с ней все необходимые действия. Ниже приведен еще один пример бесконечного цикла.

```
$test = 0;
while ($test = 0)
{
      $Test++;
}
```

Помните, что в языке PHP переменные с именами `$Test` и `$test` различаются.

Отсутствие символа доллара

Часто забывают о символе доллара ($) в имени переменной. Но в этом случае PHP подскажет вам, где следует искать ошибку:

```
Parse error: parse error in test.php on line 7
```

Ошибки, связанные с кавычками

Ошибки, связанные с кавычками, могут быть обусловлены их количеством или неверным типом. Например, в следующем примере их слишком много:

```
$test = "<table width="100%">";
```

В этом случае вторые двойные кавычки (перед числом 100) будут интерпретироваться PHP как закрывающие. Соответственно PHP будет воспринимать символ 1 в качестве оператора, что, конечно же, не будет иметь никакого смысла. Существуют два способа исправления ошибок такого рода:

```
$test = "<table width='100%'>";
```

и

```
$test = "<table width=\"100%\">";
```

В следующем примере кавычек уже слишком мало:

```
$test = "<table width='100%'>;
```

Интерпретатор PHP будет считывать последующие строки сценария как часть текстовой строки до тех пор, пока не встретит символ закрывающей двойной кавычки ("). В этом случае номер строки, который будет выведен в сообщении об ошибке, не будет соответствовать строке сценария, которая в действительности содержит ошибку (пропущенный символ ").

Ошибки также возможны и в тех случаях, когда вместо одинарных кавычек используются двойные и наоборот. В главе 5 подробно рассматривалось различие в использовании этих символов.

Вывод невидимых символов

Некоторые функции, такие как `header()`, необходимо вызывать перед любыми операторами вывода. После операторов вывода такого рода функции будут работать некорректно. Рассмотрим следующий пример:

```
<html>
<?php
     header("Location: http://company.com");
?>
```

Дескриптор `<html>` размещен вне фрагмента PHP и соответственно отправляется в качестве HTML-вывода. Это значит, что приведенный сценарий не сработает. А вот в этом случае все уже корректно:

```
<?php
     header("Location: http://company.com");
?>
<html>
```

Рассмотрим еще один пример, ошибку в котором не так уж легко заметить:

```
 <?php
     header("Location: http://company.com");
?>
```

Дело в том, что приведенный фрагмент кода содержит символ пробела перед открывающим дескриптором PHP. Естественно, он будет выведен на экран, хотя пользователь этого и не заметит. Таким образом, функция `header()` не будет выполнена корректно, поскольку перед ее вызовом уже была выведена информация. Это достаточно распространенная ошибка, выявить которую — задача не из легких.

Нумерация элементов массива

В языке PHP нумерация элементов массива начинается с нуля (0), хотя некоторые пользователи предполагают, что с единицы (1). Это фундаментальное отличие обусловливает разную интерпретацию кода сценария разработчиком и самим интерпретатором PHP. Рассмотрим такой пример:

```
$test = 1;
while ($test <= 3)
{
     $array[]=$test;
```

```
    $test++;
}
echo $array[3];
```

В результате выполнения сценария ничего выведено не будет (в том числе не появится и сообщение об ошибке). Разработчик может предположить, что что-то не так. Но на самом деле все верно. Просто PHP создаст массив

```
$array[0] = 1
$array[1] = 2
$array[2] = 3
```

т.е. элемент `$array[3]` не будет существовать.

Включение операторов PHP

При подключении файла в сценарий PHP с помощью функции `include()` предполагается, что его содержимое будет интерпретироваться как код PHP, т.е. на место подключения из файла будет вставлен фрагмент кода на языке PHP. Однако на самом деле все обстоит немного иначе. Рассмотрим файл `file1.inc`, содержащий следующий набор инструкций:

```
if ($test == 1)
    echo "Привет";
```

Вставим его содержимое в другой файл.

```
<?php
$test = 1;
include("file1.inc");
?>
```

Разработчик ожидает, что в результате выполнения этого сценария отобразится строка `Привет`. Однако на Web-странице отобразится

```
if ($test == 1) echo "Привет";
```

т.е. содержимое файла `file1.inc` будет интерпретироваться как код HTML. Чтобы изменить ситуацию, в этот файл следует добавить дескрипторы PHP:

```
<?php
if ($test == 1)
    echo "Привет";
?>
```

Недостающая пара

В PHP круглые и фигурные скобки всегда должны содержать как открывающую, так и закрывающую части. Отсутствие одной из них приведет к ошибке. Рассмотрим следующий пример:

```
if (isset($test)
```

В этом выражении недостает одной закрывающей круглой скобки. Надо сказать, что не так уж легко проследить за тем, чтобы все блоки инструкций содержали закрывающие элементы, особенно при использовании вложенных конструкций. В качестве примера рассмотрим следующий:

```
while ($test<=3)
{
if ($test2 != "да")
{
if ($test3 > 4)
```

```
{
echo "вперед";
}
}
```

Нетрудно заметить, что открывающих фигурных скобок — три, а закрывающих — две. А если представить себе, что таких строк кода может быть около ста? В этом случае достаточно непросто определить, в чем же проблема, особенно, если программист считает, что последняя закрывающая фигурная скобка соответствует циклу `while`, а на самом деле она относится к оператору `if`. В больших сценариях очень трудно проследить за ошибками такого рода.

В этих случаях очень полезно использовать отступы и комментарии:

```
while ($test<=3)
{
     if ($test2 != "да")
     {
          if ($test3 > 4)
          {
               echo "вперед";
          } #закрывающая скобка оператора if, содержащего переменную
$test3
     } #закрывающая скобка оператора if, содержащего переменную $test2
} #закрывающая скобка цикла while
```

Путаница с круглыми и фигурными скобками

Автор не уверен, является ли эта проблема существенной только для него, или от нее страдают и другие программисты. Интерпретатор PHP без проблем отличает круглые скобки от фигурных, но человеку, особенно после десяти часов программирования, это становится делать все сложнее. Не составляет труда перепутать (и {. В этом случае PHP выведет соответствующее сообщение об ошибке.

Глава 16

Десять жизненно необходимых Web-ресурсов

В этой главе...

- Где найти статьи и учебные пособия по языку PHP
- Где взять библиотеки кода на языке PHP

Язык программирования PHP поддерживается большим сообществом разработчиков. Поэтому в Internet можно найти множество соответствующих ресурсов. В данной главе приводятся наиболее полезные из них. Помните, что вы не одиноки.

Официальный Web-узел PHP

На официальном Web-узле PHP можно найти последние версии PHP и полное справочное руководство, а также все, что необходимо знать о PHP.

`www.php.net`

Списки рассылки PHP

Если на протяжении нескольких дней вам не удается найти ошибку в сценарии, который с завидным постоянством не хочет запускаться, — обратитесь к спискам рассылки PHP (PHP Lists). Их постоянно посещают сотни разработчиков, которые знают о PHP практически все. Только оставьте там свое сообщение с вопросом, и вы сразу же получите на него ответ, не успев даже убрать руки с клавиатуры. При этом существуют разные категории списков рассылки PHP, например `php-general` (посвящен общим вопросам, связанным с PHP), `php-db` (посвящен вопросам взаимодействия PHP с базами данных), `php-install` (посвящен вопросам установки PHP), `php-windows` (посвящен работе PHP под управлением операционной системы Windows) и т.д. Существует возможность подписаться на такие списки рассылки.

Однако получение информации по подписке может оказаться достаточно трудоемким. В этом случае можно получать по сто сообщений в день. Но, если вы только начинаете программировать на PHP, анализ вопросов и предлагаемых решений может значительно вам помочь в изучении PHP. В крайнем случае можно просто подписаться на сообщения о выпуске новых версий PHP, что происходит не так часто.

`www.php.net/mailing-lists.php`

Ядро Zend

Ядро Zend является основой PHP. На этом Web-узле можно найти много интересной информации о PHP, а также статьи, последние новости, списки проводимых семинаров и даже предложения о работе.

`http://zend.com`

Web-ресурс PHP Builder

Этот Web-узел содержит множество ссылок на ресурсы для программистов PHP. Здесь можно узнать о последних новостях PHP и найти полезные статьи. Кроме того, Web-узел PHP Builder содержит примеры программ на PHP, которые разделены на следующие категории: базы данных, календари, игры, изображения и т.д.

`www.phpbuilder.com`

Web-ресурс Black Beans

Если бы можно было пользоваться только одним Web-узлом, то Web-ресурс Black Beans слыл бы достойным кандидатом на эту роль. Здесь содержится множество полезных статей о PHP, разнообразные форумы, группы новостей, средства для создания сценариев PHP и многое другое.

`www.black-beans.com.br/php_home_eng.htm`

PHP для начинающих

Этот Web-узел будет особенно полезен для начинающих разработчиков

`www.phpbeginner.com`

Web-ресурс PHP Dev Center

На этом Web-узле содержатся очень полезные статьи и обучающие программы по PHP как для новичков, так и для опытных разработчиков.

`www.onlamp.com/php`

Web-узел PHPMac.com

Этот Web-узел ориентирован на использование языка PHP на платформе Mac. Подобного рода информацию найти намного сложнее, чем, например, статьи об использовании PHP под управлением операционных систем Windows или Linux/Unix.

`www.phpmac.com`

Редакторы PHP

Этот Web-узел предоставляет набор редакторов и интегрированных средств разработки (IDE — Integrated Development Environment) для создания приложений на PHP. На данный момент он насчитывает около 105 наименований как бесплатных, так и коммерческих программных продуктов. На узле также содержатся комментарии и замечания пользователей.

`http://phpeditors.linuxbackup.co.uk`

Web-ресурс SourceForge.net

Этот Web-узел является самым большим хранилищем приложений открытого кода и приложений для Internet. Чтобы найти сценарии, написанные непосредственно на языке PHP, следует выполнить такие действия.

1. **Щелкните на вкладке** Software Map **основной страницы Web-узла.**
2. **В колонке справа выберите ссылку** Programming Languages.
3. **В колонке слева появится список языков программирования, из которого нужно выбрать элемент** PHP.

На данный момент узел насчитывает около 7000 проектов, связанных с языком PHP.

`www.sourceforge.net`

Бесплатные Web-хостинговые услуги

Данный Web-узел содержит список Web-хостинговых компаний, предоставляющих бесплатные услуги по поддержке PHP. При этом каждая компания имеет свой рейтинг.

`www.oinko.net/freephp`

Web-узел автора настоящей книги

Существует Web-узел, который обеспечивает поддержку данной книги. Здесь можно найти примеры программного кода из книги, а также список исправленных ошибок. На этом Web-узле автор предлагает последние новости о PHP, а также список наиболее полезных Web-ресурсов.

`http://janet.valade.com`

Часть VI

Приложения

"Да, Unicenter имеет автоматическую функцию помощи. Но для чего?"

В этой части...

В приложениях вы найдете инструкции по установке модуля PHP в различных операционных системах, а также исчерпывающий список наиболее важных функций. Приложение Б написано таким образом, что вам не составит труда найти нужную функцию, в том числе те вопросы, которые не рассматривались в данной книге.

Приложение А

Установка PHP

В данном приложении содержатся рекомендации по установке PHP в операционных системах Windows, Unix/Linux и Mac как для разработки Web-приложений, так и для использования интерфейса командной строки CLI (Command Line Interface). Несмотря на то что модуль PHP можно использовать совместно с различными Web-серверами, в данном приложении основное внимание уделяется Web-серверам Apache и IIS (Microsoft Internet Information Server), которые используются на 90% Web-узлов. Информацию об использовании языка PHP под управлением других операционных систем или Web-серверов можно найти на официальном Web-узле `www.php.net`.

В этом приложении рассматривается процесс установки PHP 5. Имейте в виду, что рекомендации по установке более ранних версий могут несколько отличаться. Поэтому более полную информацию по этому вопросу лучше поискать в файле `install.txt`, который входит в комплект поставки PHP.

Установка модуля PHP в системах Unix/Linux

Модуль PHP может использоваться либо в качестве модуля Web-сервера Apache для разработки Web-приложений, либо в качестве автономного независимого модуля. Для получения полной функциональности необходимо установить две разные версии PHP (для работы с сервером Apache и PHP CLI). Ниже содержатся рекомендации по установке обеих версий.

Для установки PHP нужно загрузить исходные файлы, скомпилировать их, а затем установить скомпилированные модули. Этот процесс не такой уж и сложный, как кажется на первый взгляд. В последующих нескольких разделах содержатся подробные пошаговые инструкции, которые могут существенно упростить весь процесс установки. При этом все предварительные этапы лучше выполнить заранее, поскольку чем лучше вы подготовлены, тем легче и быстрее пройдет весь процесс в целом.

Для пользователей операционной системы Linux. При установке большинства версий системы Linux Web-сервер Apache и модуль PHP устанавливаются автоматически, что позволяет избежать множества потенциальных проблем. Кроме того, вместе с системой Linux поставляются как исходные файлы PHP, так и файлы в формате RPM. (Их можно найти на установочном компакт-диске.) Но если модуль PHP автоматически установлен вместе с операционной системой или путем установки файлов RPM, то вы не сможете управлять его конфигурацией. Например, очень часто версия PHP CLI автоматически не устанавливается, или в файлах RPM отсутствует поддержка требуемой для работы баз данных. Кроме того, при автоматической установке зачастую используются параметры, которые в действительности могут никогда вам не понадобиться. Учитывайте также тот

факт, что в состав установочного комплекта системы Linux и файлов RPM, как правило, входит не самая новая версия PHP. Поэтому самый эффективный способ установки PHP заключается в использовании соответствующих исходных файлов. Именно этим вопросам и посвящаются последующие разделы.

Перед установкой модуля PHP в системах Unix/Linux

Перед использованием модуля PHP и Web-сервера Apache для создания и поддержки Web-узлов его нужно предварительно установить. Во многих версиях систем Unix/Linux сервер Apache устанавливается по умолчанию. Однако перед установкой модуля PHP удостоверьтесь в следующем.

- ✓ **Используется версия Web-сервера Apache 1.3.0 или более новая**. Для проверки используемой версии Web-сервера Apache воспользуйтесь следующей командой:

  ```
  httpd -v
  ```

  Перед использованием этой команды перейдите в каталог, в котором находится файл `httpd`.

  С точки зрения обеспечения безопасности лучше всего использовать версию Apache 1.3.27 или выше.

  В настоящее время совместное использование модуля PHP и сервера Apache версии 2 возможно лишь в экспериментальном режиме. Поэтому для поддержки коммерческих Web-приложений лучше использовать сервер Apache 1.3.

  Установлен модуль Apache `mod_so`. В большинстве случаев это именно так. Для получения списка всех подключенных модулей Apache выполните следующую команду:

  ```
  httpd -l
  ```

  Как и в предыдущем случае, перед выполнением этой команды нужно перейти в соответствующий каталог. В результате выполнения будет выведен достаточно длинный список модулей, основным из которых (для модуля PHP) является `mod_so`. Если этот модуль отсутствует, Web-сервер Apache необходимо переустановить с использованием параметра `--enable-module=so`.

  Установлена утилита `apxs` (или `apxs2` для Apache 2). Как правило, утилита `apxs` устанавливается вместе с Web-сервером Apache автоматически. Удостоверьтесь в наличии файла `apxs` (который может находиться в каталоге `/usr/sbin/apxs`). Если его удалось найти, значит, утилита `apxs` установлена. Если установка Apache осуществлялась с использованием RPM-файлов, то утилита `apxs`, скорее всего, по умолчанию будет отсутствовать. Дело в том, что файлы RPM обычно поставляются в двух версиях. Одна из них определяет базовую функциональность сервера Apache, а вторая предоставляет дополнительные средства разработки. Возможно, для установки утилиты `apxs` придется воспользоваться второй версией.

Установка модуля PHP в системах Unix/Linux

Для установки модуля PHP версии 5 в системе Unix/Linux выполните следующие действия.

1. **Зайдите на официальный Web-узел PHP** `www.php.net`.
2. **Перейдите в раздел** Downloads.

3. **Выберите файл с исходным кодом последней версии PHP. На момент написания данной книги такой версией была 5.0.0.**

   Откроется диалоговое окно.

4. **Выберите режим сохранения файла.**

   Откроется диалоговое окно, в котором можно указать путь сохранения файла.

5. **Выберите каталог, в котором нужно сохранить файл с исходным кодом (например, `/usr/src`). Затем щелкните на кнопке** Save.

6. **После завершения загрузки файла перейдите в каталог, в котором он был сохранен (например, с помощью команды `cd- /usr/src`).**

   Вы увидите файл, имя которого начинается с символов `php-`, после которых указана версия PHP. Файл имеет расширение `tar.gz` и представляет собой архив. В нем содержится много других файлов, заархивированных с помощью команды `tar`. (Такие файлы называются также *tarball*.)

7. **Распакуйте загруженный файл.**

   Это можно сделать с помощью команды

   ```
   gunzip -c php-5.0.0.tar.gz | tar -xf -
   ```

   В результате будет создан каталог с именем `php-5.0.0`.

8. **Перейдите в каталог, который был создан в результате распаковывания файла.**

   Для этого воспользуйтесь командой

   ```
   cd php-5.0.0
   ```

9. **Введите команду `configure`.**

   Для этого введите в командной строке команду `./configure` с необходимым набором параметров. Если модуль PHP устанавливается для совместного использования с Web-сервером Apache, введите следующую команду:

   ```
   ./configure --with-apxs
   ```

   При этом на экран будет выведено множество строк. Подождите завершения команды `configure` (это может занять несколько минут).

   Наряду с приведенным параметром можно использовать и многие другие, позволяющие подключить поддержку баз данных или изменить рабочие каталоги PHP. Параметры команды `configure` более подробно описываются ниже, в разделе "Параметры установки для операционных систем Unix/Linux/Mac".

Если путь к папке, где установлена утилита `apxs`, не найден, отобразится соответствующее сообщение об ошибке. В этом случае необходимо определить точное месторасположение утилиты `apxs` (например, с помощью команды `find / -name apxs`) и добавить соответствующий путь к параметру `with-apxs`. В этом случае команда `configure` может выглядеть следующим образом:

```
./configure --with-apxs=/usr/sbin/apxs
```

При использовании Web-сервера Apache версии 2 необходимо использовать утилиту `apxs2`.

10. **Введите команду `make`.**

При этом будет выведено большое количество строк. Подождите завершения выполнения команды `make`. Это может занять несколько минут.

По умолчанию устанавливаются обе версии PHP, CGI и CLI. Взаимодействие модуля PHP с Web-сервером Apache обеспечивается файлом `libphp5.so`. В свою очередь, основным файлом для версии PHP CLI является `php`, расположенный в каталоге PHP (например, `user/local/php`). Если какую-либо из версий PHP устанавливать нет необходимости, измените настроечные параметры команды `configure`, которые описываются ниже, в разделе "Параметры установки для операционных систем Unix/Linux/Mac".

11. **Введите команду** `make install`.

    В результате выполнения этой команды все необходимые файлы переместятся в соответствующие каталоги. Например, версия PHP для Web будет установлена в каталоге Apache (например, `/usr/local/apache/libexec/libphp5.so`).

Альтернативный метод установки вместе с сервером Apache

Иногда установить модуль PHP с использованием утилиты `apxs` не удается. В этом случае следует воспользоваться альтернативным методом установки, который и описывается в данном разделе. Несмотря на то что утилита `apxs` в большинстве случаев работает корректно и позволяет быстро установить модуль PHP, альтернативный способ иногда окажется совсем нелишним. Ниже приведены соответствующие пошаговые инструкции.

1. **Зайдите на официальный Web-узел PHP** `www.php.net`.
2. **Перейдите в раздел** Downloads.
3. **Выберите файл с исходным кодом последней версии PHP. На момент написания данной книги это была версия 5.0.0.**

   Откроется диалоговое окно.
4. **Выберите режим сохранения файла.**

   Откроется диалоговое окно, в котором можно указать путь, по которому сохранен файл.
5. **Укажите каталог, в котором нужно сохранить файл с исходным кодом (например,** `/usr/src/php`**). Затем щелкните на кнопке** Save.
6. **После завершения загрузки файла перейдите в каталог, в котором он был сохранен (например, с помощью команды** `cd- /usr/src/php`**).**

   Загруженный файл начинается с символов `php-`, после которых следует номер версии PHP. Файл имеет расширение `tar.gz`.
7. **Распакуйте загруженный файл.**

   Это можно сделать с помощью команды

   ```
   gunzip -c php-5.0.0.tar.gz | tar -xf -
   ```

   В результате будет создан каталог с именем `php-5.0.0` с несколькими подкаталогами.
8. **Повторите шаги 1–5 еще раз, однако на этот раз для загрузки файла с исходным кодом Web-сервера Apache. Загрузите этот файл в каталог, где ранее был разархивирован файл с исходным кодом PHP.**

   Исходный код сервера Apache можно найти по адресу `httpd.apache.org`.

   Ниже в качестве примера рассматривается установка сервера Apache версии 1.3.27. На момент написания данной книги это была его последняя версия.

В настоящее время использовать PHP совместно с Web-сервером Apache 2 не рекомендуется. Для того чтобы проверить, изменились ли соответствующие рекомендации, посетите официальный Web-узел PHP.

9. **Распакуйте архив с исходным кодом сервера Apache. Для этого воспользуйтесь командой**

   ```
   gunzip apache_1.3.27.tar.gz | tar -xf -
   ```

   В результате выполнения всех предыдущих действия должно быть создано два каталога `php-5.0.0` и `apache_1.3.27` с набором подкаталогов.

10. **Введите команду** `cd apache_1.3.27`.

11. **Введите команду** `./configure`.

    В данном случае входные параметры команды ни на что не влияют. Дело в том, что так выполняется предварительная настройка Web-сервера Apache, которую нужно выполнить перед настройкой модуля PHP. Позже эта команда будет использоваться повторно с указанием соответствующих параметров. Дождитесь завершения выполнения команды `./configure`.

12. **Введите команду** `cd ../php-5.0.5`.

13. **Введите команду**

    ```
    ./configure --with-apache=../apache_1.3.27
    ```

    Если для набора этой команды необходимо воспользоваться двумя строками, то в конце первой из них следует поставить обратную косую черту (\).

    Кроме приведенного выше параметра, при запуске команды `configure` можно использовать и другие опции включения поддержки баз данных или изменения рабочих каталогов PHP. Более подробно они описываются ниже, в разделе "Параметры установки для операционных систем Unix/Linux/Mac".

14. **Введите команду** `make`.

    Отобразится подробный листинг. Дождитесь завершения выполнения команды `make` (это может занять несколько минут).

15. **Введите команду** `make install`.

    Ожидание ее завершения много времени не потребует.

16. **Введите команду** `cd ../apache_1.3.27` **для возврата в корневой каталог сервера Apache.**

17. **Для повторной настройки Web-сервера Apache введите команду**

    ```
    ./configure --prefix=/www --activate-module=src/modules/php5/
    libphp5.a
    ```

    Если эту команду нужно набрать в двух строках, в конце первой из них поставьте обратную косую черту. Также обязательным является наличие символа пробела. Между двумя параметрами не забудьте поставить как минимум один пробел.

18. **Введите команду** `make`.

    Отобразится подробный листинг. Дождитесь завершения команды `make` (для этого может потребоваться несколько минут).

    Последний шаг, который нужно выполнить, зависит от того, был ли сервер Apache ранее установлен, или это его первая установка.

19. **При первой установке Web-сервера Apache выполните команду** `make install`.

Если же сервер Apache ранее был уже установлен и запущен, выполните следующие действия.

- **Завершите выполнение сервера Apache.**

  Для этого можно запустить специальный сценарий, созданный при установке сервера. Обычно он называется `apachectl` и содержится в подкаталоге `bin` основного каталога Apache, например `/usr/local/apache/bin`, `/sbin` или `/usr/sbin`. Этот файл может также находиться в каталоге со сценариями автозагрузки (например, `/etc/rc.d/init.d`). Для завершения работы сервера Apache введите команду `apachectl stop`.

- **Найдите новый файл `httpd`, который был создан при выполнении п. 18.**

  Скорее всего, этот файл будет находиться в одном из подкаталогов основного каталога `apache`, например `/usr/src/php/apache_1.3.27/bin/httpd`.

- **Найдите старый файл `httpd`, который существовал до установки.**

  Этот файл можно найти в каталоге `/usr/local/apache/bin`, `/sbin` или `/usr/sbin`.

- **Замените старый файл новым.**

Перед копированием сделайте резервную копию старого файла `httpd`.

Если вы хотите обновить используемую версию PHP до более новой версии, следует воспользоваться именно этим методом. При первой установке его применять нельзя.

Установка модуля PHP в системе Mac

После выхода PHP 4.3 использовать модуль PHP в системе Mac стало так же просто, как и в системах Unix/Linux. При этом модуль PHP может использоваться как совместно с Web-сервером Apache (для поддержки Web-приложений), так и в качестве независимого интерпретатора. Для обеспечения полной функциональности необходимо установить две отдельные версии PHP, PHP для Web и PHP CLI. Ниже даны инструкции по установке обеих из них.

Для установки PHP нужно загрузить исходные файлы, скомпилировать их, а затем установить скомпилированные модули. Этот процесс не такой уж и сложный, как кажется на первый взгляд. В следующих разделах содержатся подробные пошаговые инструкции, которые могут существенно упростить весь процесс установки. При этом все предварительные этапы лучше выполнить заранее, поскольку чем лучше вы подготовлены, тем легче и быстрее пройдет весь процесс в целом.

Перед установкой модуля PHP в системе Mac

Перед использованием модуля PHP и Web-сервера Apache для создания и поддержки Web-узлов его нужно предварительно установить. Во многих версиях системы Mac сервер Apache устанавливается по умолчанию. Однако перед установкой модуля PHP удостоверьтесь в следующем.

- ✓ **Используется версия Web-сервера Apache 1.3.0 или более новая**. Для проверки используемой версии Web-сервера Apache воспользуйтесь командой

  ```
  httpd -v
  ```

  Перед использованием этой команды перейдите в каталог, в котором находится файл `httpd`.

  С точки зрения обеспечения безопасности лучше всего использовать версию Apache 1.3.27 или выше.

  В настоящее время совместное использование модуля PHP и сервера Apache версии 2 возможно лишь в экспериментальном режиме. Поэтому для поддержки коммерческих Web-приложений лучше использовать сервер Apache 1.3.
- ✓ **Установлен модуль Apache `mod_so`.** В большинстве случаев это именно так. Для получения списка всех подключенных модулей Apache выполните команду

  ```
  httpd -l
  ```

  Как и в предыдущем случае, перед выполнением этой команды нужно перейти в соответствующий каталог. В результате выполнения будет выведен достаточно длинный список модулей, основным из которых (для модуля PHP) является `mod_so`. Если этот модуль отсутствует, Web-сервер Apache необходимо переустановить с использованием параметра `--enable-module=so`.
- ✓ **Установлена утилита `apxs`.** В большинстве случаев утилита `apxs` устанавливается вместе с Web-сервером Apache автоматически. Удостоверьтесь в наличии файла с именем `apxs` (который может находиться в каталоге `/usr/sbin/apxs`). Если его удалось найти, значит, утилита `apxs` установлена.
- ✓ **Установлены файлы с компакт-диска со средствами разработки.** Обычно этот вспомогательный компакт-диск входит в основной комплект поставки операционной системы Mac OS X. Если его найти не удалось, все необходимые средства можно загрузить с Web-узла `developer.apple.com/tools/macosxtools.html`.

Установка модуля PHP в системе Mac

Для установки модуля PHP в операционной системе Mac выполните следующие действия.

1. **Зайдите на официальный Web-узел PHP (`www.php.net`).**
2. **Перейдите в раздел** Downloads.
3. **Выберите файл с исходным кодом последней версии PHP. На момент написания данной книги такой версией была 5.0.0.**

   Откроется диалоговое окно.
4. **Выберите режим сохранения файла.**

   В появившемся диалоговом окне нужно указать путь, где будет сохранен файл.
5. **Перейдите в каталог, в котором нужно сохранить файл с исходным кодом (например, `/usr/src`), а затем щелкните на кнопке** Save.
6. **После завершения загрузки файла перейдите в каталог, в котором он был сохранен (например, с помощью команды `cd- /usr/src`).**

   Вы увидите файл, имя которого начинается с символов `php-`, после которых указана версия PHP. Файл имеет расширение `tar.gz` и представляет собой архив, который может быть автоматически распакован программой Stuffit Expander. При этом появится каталог `php-5.0.0`. Если все именно так и произошло, можно сразу перейти к шагу 8.

7. **Распакуйте загруженный файл.**

   Это можно сделать с помощью команды

   ```
   tar xvfz php-5.0.0.tar.gz
   ```

   При этом будет создан каталог с именем `php-5.0.0` и несколькими подкаталогами.

8. **Перейдите в новый каталог, который был создан при разархивировании файла.**

   Для этого можно воспользоваться, например, командой

   ```
   cd php-5.0.0
   ```

9. **Введите команду** `configure`.

   Для этого следует ввести команду `./configure` с набором необходимых параметров, среди которых обязательными являются следующие.

   - **Параметры местоположения.** Поскольку в системе Mac файлы размещаются в каталогах, которые отличаются от используемых модулем PHP по умолчанию, их следует указать вручную. Воспользуйтесь следующим набором параметров:

     ```
     --prefix=/usr
     --sysconfdir=/etc
     --localstatedir=/var
     --mandir=/usr/share/man
     ```

   - **Параметр** `zlib`: `--with-zlib`.

   - **Параметр Apache**. Если модуль PHP устанавливается для совместного использования с Web-сервером Apache, укажите также параметр `--with-apxs`.

     Таким образом, для корректной настройки нужно воспользоваться командой `configure` со следующими параметрами:

     ```
     ./configure --prefix=/usr --sysconfdir=/etc
     --localstatedir=/var --mandir=/usr/share/man
     --with-apxs
     ```

     Если для набора этой команды требуются две строки, в конце первой из них поставьте обратную косую черту (\).

     Эта команда предоставляет подробный листинг. Дождитесь завершения ее выполнения (это может занять несколько минут).

   Если путь к папке, где установлена утилита `apxs`, не найден, то будет выведено соответствующее сообщение об ошибке. В этом случае необходимо определить точное месторасположение утилиты `apxs` (например, с помощью команды `find / -name apxs`) и добавить соответствующий путь к параметру `with-apxs` (`--with-apxs=/usr/sbin/apxs`).

   Если вы используете Web-сервера Apache 2, воспользуйтесь командой `apxs2`.

   Наряду с приведенными опциями можно использовать и другие, предназначенные для активизации поддержки баз данных или изменения различных установок PHP. Более подробно они описаны ниже, в разделе "**Параметры установки для операционных систем** Unix/Linux/Mac".

10. **Введите команду** `make`.

    Эта команда предоставит подробный листинг. Дождитесь завершения этой команды (на это может уйти несколько минут).

По умолчанию устанавливаются обе версии PHP, CGI и CLI. Взаимодействие модуля PHP с Web-сервером Apache обеспечивается файлом `libphp5.so`. В свою очередь, основным файлом для версии PHP CLI является `php`, расположенный в каталоге PHP (например, `user/local/php`). Если какую-либо из версий PHP устанавливать не нужно, измените параметры настройки команды `configure`, которые описаны ниже, в разделе "**Параметры установки для операционных систем** Unix/Linux/Mac".

11. **Введите команду `sudo make install`.**

Параметры установки для операционных систем Unix/Linux/Mac

В предыдущих разделах описывался пошаговый процесс установки модуля PHP в системах Unix/Linux/Mac. Однако следует заметить, что при установке этого модуля можно использовать различные параметры командной строки, а некоторые из них могут пригодиться и вам. Например, вам может понадобиться поддержка различных баз данных, таких как MySQL или Oracle, или вы захотите установить отдельные компоненты PHP в каталоги, отличные от используемых по умолчанию. Кроме того, может возникнуть потребность и в использовании дополнительного программного обеспечения.

Для активизации всех перечисленных возможностей в процессе установки PHP нужно указать соответствующие параметры командной строки. Просто добавьте их при вызове команды `configure` в п. 13 для систем Unix/Linux или при выполнении п. 9 для системы Mac. При этом порядок этих параметров не играет никакой роли. В табл. А.1 приведены параметры командной строки, которые в процессе установки модуля PHP используются чаще всего. Для того чтобы получить полный перечень параметров, перейдите в каталог, в котором установлен модуль PHP, и введите команду `configure -help`.

Таблица А.1. Параметры командной строки для настройки модуля PHP в системах Unix/Linux

Параметр	Назначение
`prefix=`*PREFIX*	Устанавливает *PREFIX* в качестве основного каталога PHP. По умолчанию *PREFIX*`=/usr/local`
`infodir=`*DIR*	Устанавливает документацию в каталог *DIR*. По умолчанию *DIR*=*PREFIX*`/info`
`mandir=`*DIR*	Устанавливает файлы справочного руководства в каталог *DIR*. По умолчанию *DIR*=*PREFIX*`/man`
`with-config-file-path=`*DIR*	Задает путь *DIR*, в котором будет размещаться конфигурационный файл `php.ini`. Этот файл требуется для корректного функционирования модуля PHP. По умолчанию *DIR*`=/usr/local/lib`
`disable-cgi`	Отменяет установку двоичных файлов PHP CGI
`disable-cli`	Отменяет установку двоичных файлов PHP CLI
`disable-libxml`	Отключает поддержку XML
`enable-debugger`	Включает поддержку отладчика
`enable-ftp`	Включает поддержку протокола FTP
`enable-url-include`	Позволяет с помощью функции `include()` извлекать файлы из каталогов HTTP и FTP, а также из каталога `include`

Параметр	Назначение
`with-msql=`*DIR*	Включает поддержку mSQL. *DIR* — это исходный каталог mSQL, которому по умолчанию соответствует каталог `/usr/local/Hughes`
`with-mysql=`*DIR*	Включает поддержку MySQL версии 4.0 или ниже. Параметр *DIR* — это исходный каталог MySQL, которому по умолчанию соответствует каталог `/usr/local/Hughes`
`with-mysqli=`*DIR*	Включает поддержку MySQL версии 4.1 или выше. Параметр *DIR* определяет путь к файлу `mysql_config`, установленному вместе с MySQL. Для него путь по умолчанию не задан
`with-openssl=`*DIR*	Включает поддержку протокола OpenSSL для безопасного сервера. Требуется установленная версия OpenSSL 0.9.5 или выше
`with-oracle=`*DIR*	Включает поддержку Oracle. Параметру *DIR* по умолчанию соответствует каталог, заданный в переменной окружения `ORACLE_HOME`
`with-pgsql=`*DIR*	Включает поддержку баз данных PostgreSQL. Параметр *DIR* задает исходный каталог PostgreSQL. По умолчанию используется каталог `/usr/local/pgsql`
`with-servlet=`*DIR*	Включает поддержку сервлетов. Параметр *DIR* определяет основной каталог JSDK. При этом расширение Java должно быть подключено как dll-библиотека
`with-xml`	Включает поддержку XML

Настройка сервера Apache и модуля PHP в системах Unix/Linux/Mac

Модуль PHP использует конфигурационный файл `php.ini`, в котором определены различные параметры настройки, которые в случае необходимости можно изменить. Этот файл считывается каждый раз при загрузке модуля PHP и по умолчанию размещается в каталоге `/usr/local/lib/php.ini`. Однако, как уже упоминалось в предыдущем разделе, в процессе установки этот путь можно изменить с помощью соответствующего параметра. Если в указанном каталоге файл `php.ini` не найден, используются значения, установленные по умолчанию.

Файл `php.ini` необходимо скопировать в требуемый каталог вручную. Исходный конфигурационный файл `php.ini-dist` (с параметрами по умолчанию) можно найти в корневом каталоге PHP. Для того чтобы скопировать файл `php.ini-dist` в требуемый каталог, воспользуйтесь командой

```
cp php.ini-dist /usr/local/lib/php.ini
```

или

```
sudo cp php.ini-dist /usr/local/lib/php.ini
```

в операционной системе Mac.

После этого в файл `php.ini` можно вносить необходимые изменения, хотя значения, установленные по умолчанию, также обеспечат корректное функционирование модуля PHP. Многие параметры и соответствующие им значения рассматриваются в различных разделах данной книги, материал которых имеет к ним непосредственное отношение. Например, можно изменить значения параметров в файле `php.ini` и таким образом модифицировать процедуру обработки ошибок и отображения соответствующих сообщений. Эти вопросы подробно рассматривались в главе 4.

Если модуль PHP используется совместно с Web-сервером Apache, то его нужно настроить на корректное распознавание файлов с расширением `.php`. Для этого выполните следующие действия.

1. **Найдите конфигурационный файл сервера Apache** `httpd.conf`.

   Он может находиться в каталоге `/etc` или `/usr/local/apache/conf`. В операционной системе Mac этот файл можно поискать в каталоге `/etc/httpd`. Для корректного функционирования модуля PHP в файл `httpd.conf` Web-сервера Apache нужно внести соответствующие изменения.

2. **Добавьте в файл** `httpd.conf` **строку, обеспечивающую загрузку модуля PHP.**

   Для этого найдите строку с директивой `LoadModule` и модифицируйте ее следующим образом:

   ```
   LoadModule php5_module libexec/libphp5.so
   ```

   Удостоверьтесь в том, что такая строка имеется в файле `httpd.conf`. Если ее в нем нет, нужно внести соответствующие изменения. Если в начале строки с директивой `LoadModule` содержится символ решетки (#), удалите его.

3. **Добавьте в файл** `httpd.conf` **строку с расширениями файлов, которые могут содержать код на языке PHP.**

   Сначала найдите строки с директивой `AddType`. Таких строк может быть несколько. (Каждая из них может использоваться различным программным обеспечением.) В частности, для модуля PHP нужно добавить строку

   ```
   AddType application/x-httpd-php .php
   ```

   Удостоверьтесь в том, что такая строка имеется в файле `httpd.conf`. Если в начале строки содержится символ решетки, удалите его. Директива `AddType` указывает серверу, в файлах с каким расширением (`.php`) следует искать код PHP. По мере необходимости можно добавить и другие расширения. При работе в операционной системе Mac эту строку необходимо добавить в конец файла `httpd.conf`.

4. ***Только для пользователей операционной системы Mac.*** **Если ранее уже был установлен модуль PHP, входящий в комплект поставки системы Mac, закомментируйте следующие строки:**

   ```
   LoadModule hfs_apple_module
   libexec/httpd/mod_hfs_apple.so
   AddModule mod_hfs_apple.c
   ```

   Для этого в начало каждой строки достаточно добавить символ решетки (#).

5. **Запустите или перезапустите заново Web-сервер Apache.**

   Для запуска или перезапуска Web-сервера Apache можно воспользоваться специальным сценарием, который устанавливается при его установке. Он может называться `apachectl` или `httpd.apache` и содержаться в каталоге `/bin` или `/usr/local/apache/bin`. Например, если сценарий называется `apachectl`, то для запуска сервера Apache введите команду `apachectl start`, для его перезапуска — команду `apachectl restart`, а для останова — команду `apachectl stop`. При работе в операционной системе Mac нужно воспользоваться также дополнительной командой `sudo`, например `sudo apachectl restart`. Иногда при перезапуске сервера Apache новые параметры не вступают в силу. В этом случае остановите работу сервера, а затем запустите его заново. Зачастую сервер Apache запускается сразу же при включении компьютера. Поэтому при возникновении любых проблем можно просто выключить компьютер, а затем включить его заново.

После любых изменений в конфигурационном файле `php.ini` Web-сервер Apache необходимо перезапустить.

Установка PHP в системе Windows

Модуль PHP можно использовать также в системах Windows 98/Me (хотя это и не рекомендуется) и Windows NT/2000/XP. Его можно установить для использования совместно с Web-сервером (версия PHP CGI) для поддержки Web-узлов или для создания независимых сценариев (PHP CLI). Для функционирования двух этих различных версий языка PHP требуются разные исполняемые файлы. Можно установить одновременно обе версии либо только одну из них.

В системе Windows модуль PHP можно установить двумя способами: автоматически, с помощью программы установки, или вручную, воспользовавшись архивным файлом `.zip`. Конкретный способ установки зависит от того, для чего вы планируете применять язык PHP. При выборе метода установки руководствуйтесь следующими рекомендациями.

- ✓ **Автоматическая программа установки** позволяет установить только версию PHP CGI. При этом можно указать, какие компоненты PHP вам требуются. Например, интерфейс ODBC предназначен для взаимодействия с базами данных (такими как MS Access) или с расширением SQlite, которое используется для хранения данных в текстовых файлах. (Вопросы взаимодействия PHP с базами данных подробно рассматривались в главе 12.) Можно также подключить расширения для работы с языком XML, протоколом FTP и объектами COM. Однако данный метод установки не позволяет инсталлировать версию PHP CLI.

  Этот простой и быстрый метод установки нужно использовать в следующих случаях.

  - Модуль PHP планируется использовать только для разработки и поддержки Web-приложений, а версия PHP CLI не требуется.
  - Модуль PHP используется для генерации кода HTML или обработки данных HTML-форм.
  - Вам не требуется взаимодействие с базами данных или вполне достаточно интерфейса ODBC.
  - Все поставленные задачи позволяет решить встроенная функциональность модуля PHP.

- ✓ **Установка вручную.** Все установочные файлы содержатся в архивном файле `.zip`. Этот файл нужно загрузить, разархивировать, а затем разместить полученные файлы в соответствующих каталогах.

  При установке вручную модуль PHP обладает большей функциональностью. Если вам понадобился некоторый компонент PHP (например, интерфейс PHP CLI или расширение для работы с базой данных), то его придется установить вручную. При этом все необходимые файлы можно найти в загруженном архивном файле `.zip`. Различные расширения языка PHP подробно рассматривались в главе 14.

Автоматическая установка PHP CGI

Приведенная ниже последовательность действий позволит установить модуль PHP в системе Windows, предназначенный для разработки Web-приложений. Не забывайте о том, что при этом версия PHP CLI не устанавливается.

1. **Введите в браузере URL-адрес** `www.php.net`.
2. **Перейдите в раздел** Downloads.
3. **Перейдите в раздел** Windows Binaries. **Щелкните на соответствующей ссылке, чтобы загрузить программу установки последней версии PHP (на момент написания книги — 5.0.0).**
4. **В результате будет выведен список зеркальных Web-узлов, с которых можно загрузить требуемый файл. Выберите Web-узел, расположенный к вам ближе всего.**

   Откроется диалоговое окно.
5. **Выберите режим сохранения файла.**

   В открывшемся диалоговом окне укажите путь к каталогу, в котором будет сохранен загружаемый файл.
6. **Перейдите в каталог, в котором нужно сохранить программу установки PHP, а затем щелкните на кнопке** Save.

   После окончания загрузки файла перейдите в каталог, в котором он был сохранен. Имя загруженного файла должно содержать символы `php`, номер версии PHP и строку `-installer.exe`, например `php5.0.0-installer.exe`.
7. **Если используемый Web-сервер запущен (кроме серверов IIS (Internet Information Server) или PWS (Personal Web Server)), завершите его работу.**
8. **Перейдите в каталог с загруженным файлом и дважды щелкните на его имени** (`php5.0.0-installer.exe`).

   Появится окно, показанное на рис. А.1.

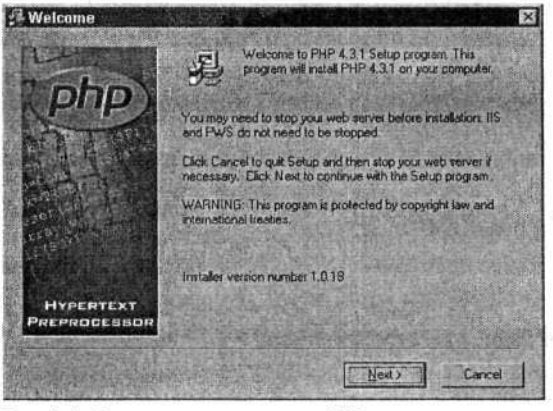


Рис. А.1. Окно мастера установки PHP

9. **Щелкните на кнопке** Next.

   В новом диалоговом окне появится текст с лицензией.
10. **Щелкните на кнопке** I agree.

    Появится диалоговое окно, в котором можно выбрать тип установки.

11. **Выберите тип Standard и щелкните на кнопке Next.**

    В появившемся окне можно выбрать каталог, в который будет установлен модуль PHP.

12. **Если вас вполне устраивает установка PHP в каталог по умолчанию (`c:\php`), щелкните на кнопке Next. В противном случае щелкните на кнопке Browse, укажите подходящий каталог и после этого также щелкните на кнопке Next.**

    В появившемся диалоговом окне можно выбрать параметры электронной почты.

13. **В диалоговом окне настройки электронной почты содержатся два поля, в которые можно ввести адрес используемого сервера SMTP или адрес электронной почты, который будет использоваться при отправке почтовых сообщений из сценариев PHP. Если эти данные вам неизвестны, оставьте в этих полях значения, установленные по умолчанию.**

    Позднее эти изменения можно внести и в конфигурационный файл модуля PHP.

14. **Щелкните на кнопке Next.**

    В появившемся диалоговом окне (рис. А.2) содержится список Web-серверов, совместно с которыми можно использовать модуль PHP.

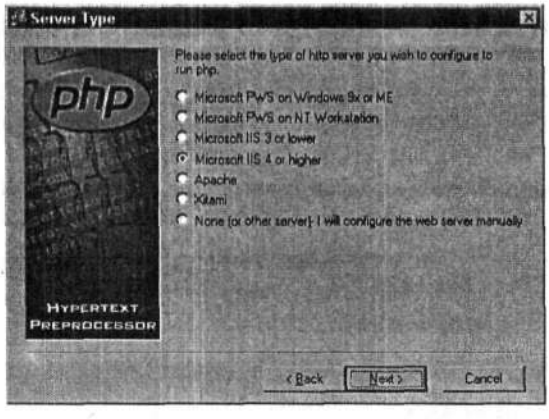


Рис. А.2. В диалоговом окне Server Type программы установки PHP можно выбрать используемый Web-сервер

15. **Выберите используемый Web-сервер. Если в списке такого сервера нет, выберите значение None.**

16. **Щелкните на кнопке Next.**

    Появится диалоговое окно Ready. Это свидетельствует о том, что программа установки готова к инсталляции модуля PHP.

17. **Для запуска процесса установки щелкните на кнопке Next.**

    После завершения установки модуля PHP 5.0.0 появится диалоговое окно со всей необходимой информацией. Внимательно прочтите это сообщение. Если понадобится, перезапустите Web-сервер. Например, при использовании Web-сервера Apache диалоговое окно будет иметь вид, как на рис. А.3.

    Получение этого сообщения вовсе не означает, что сервер Apache не установлен. Просто для обеспечения его корректного взаимодействия с модулем PHP все настройки необходимо выполнить вручную. Как это сделать, вы узнаете из последующих разделов.

Возможно, в скором будущем программа установки PHP будет вносить изменения в конфигурационный файл сервера Apache автоматически, тем самым избавляя разработчиков от лишних хлопот.

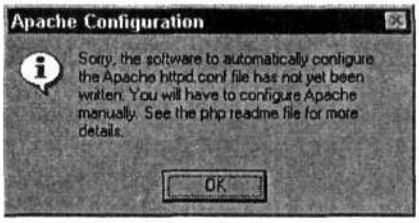


Рис. А.3. Сообщение после завершения установки PHP в случае использования Web-сервера Apache

Установка модуля PHP вручную

Для того чтобы вручную установить модуль PHP 5 в системе Windows, выполните следующие действия.

1. **Зайдите на Web-узел** `www.php.net`.
2. **Перейдите в раздел** Downloads.
3. **Перейдите в раздел** Windows Binaries **и щелкните на ссылке для загрузки архивного файла** `.zip` **последней версии PHP (на момент написания книги — 5.0.0).**

Если вам известен формат сжатия файлов `.bz` и вы знаете, как с ним работать, можно загрузить именно такой файл. Он имеет меньший размер и соответственно загрузится гораздо быстрее. Однако учитывайте тот факт, что такой формат сжатия поддерживается не всем программным обеспечением.

4. **В результате будет выведен список зеркальных Web-узлов, с которых можно загрузить требуемый файл. Выберите Web-узел, расположенный к вам ближе всего.**

   Откроется диалоговое окно.

5. **Выберите режим сохранения файла.**

   В открывшемся диалоговом окне можно указать путь к каталогу, в котором будет сохранен загружаемый файл.

6. **Перейдите в каталог, в котором нужно сохранить загружаемый файл, а затем щелкните на кнопке** Save.

   После окончания загрузки файла зайдите в выбранный каталог и найдите там загруженный файл. Его имя начинается с символов `php`, после них следует номер версии PHP и строка `-win32.zip`, например `php-5.0.0-win32.zip`.

7. **Разархивируйте загруженный** `zip`**-файл в каталог, в который вы хотите установить модуль PHP, например** `c:\php`.

   После двойного щелчка на имени `zip`-файла появится диалоговое окно программы, которая используется для работы с архивными файлами, например WinZip или Pkzip. Выберите режим разархивирования файла и каталог, в котором нужно разместить установочные файлы.

Вполне возможно, что при разархивировании zip-файла будет создан каталог php-5.0.0-win32. В этом случае его имя можно заменить на более простое, например php. Тогда после установки файлов PHP на диск c:\ они будут размещены в каталоге c:\php. Такой выбор удобен тем, что во многих конфигурационных файлах необходимо указывать путь к каталогу, где установлен модуль PHP, а в большинстве случаев по умолчанию используется именно каталог c:\php.

Не стоит устанавливать PHP в каталог, в имени которого содержится пробел, например Program Files/PHP. Иногда из-за этого могут возникать проблемы.

После выполнения всех перечисленных выше шагов на вашем жестком диске будет содержаться каталог со всеми необходимыми для работы файлами PHP.

8. **В корневом каталоге PHP содержится файл** php.exe**, предназначенный для работы с интерфейсом командной строки PHP CLI. В случае необходимости его можно переместить в любой другой каталог.**

Настройка модуля PHP и Web-сервера в системе Windows

Для настройки работы модуля PHP предназначен специальный конфигурационный файл. Кроме того, для того чтобы Web-сервер корректно обрабатывал сценарии PHP, необходимо внести соответствующие изменения и в конфигурационный файл Web-сервера. Эти вопросы и рассматриваются в данном разделе.

Настройка PHP в системе Windows

Модуль PHP использует конфигурационный файл php.ini, в котором можно задать различные параметры настройки. Он считывается каждый раз при запуске PHP. Если использовалась программа автоматической установки PHP, то этот файл создается автоматически. В противном случае файл php.ini необходимо скопировать в требуемый каталог вручную. Исходный конфигурационный файл php.ini-dist (с параметрами по умолчанию) можно найти в корневом каталоге PHP. В зависимости от используемой операционной системы его необходимо переписать в один из следующих каталогов.

- ✓ windows (в системах **Windows 98/Me/XP**)
- ✓ winnt (в системах **Windows NT/2000**)

Если ранее был установлен модуль PHP (например, версии 4.3), сделайте резервную копию старого конфигурационного файла php.ini, а используемые настройки перенесите в новый файл.

При использовании Web-сервера IIS одно из значений по умолчанию необходимо изменить. Для этого найдите в файле php.ini следующую строку:

```
;cgi.force_redirect = 1
```

В этой строке укажите значение 0 и уберите символ комментариев (;) в ее начале:

```
cgi.force_redirect = 0
```

В большинстве случаев используемые по умолчанию параметры (кроме упомянутого выше) обеспечивают корректную работу модуля PHP. Однако по мере необходимости их можно изменить. Большинство параметров конфигурационного файла php.ini рассматривались на протяжении всей книги. Например, рекомендации по настройке режима вывода сообщений об ошибках см. в главе 4.

Настройка Web-сервера для работы с PHP

Для того чтобы Web-сервер корректно обрабатывал сценарии PHP, необходимо внести изменения в его конфигурационный файл и обеспечить корректное распознавание файлов сценариев PHP. Если модуль PHP был установлен с использованием автоматической программы инсталляции и в системе установлен Web-сервер IIS или PWS (кроме IIS 6), то вся настройка выполняется автоматически. Если же вы используете Web-сервер Apache или модуль PHP был установлен вручную, то все изменения в конфигурационный файл придется внести самостоятельно.

Настройка Web-сервера Apache

Для настройки Web-сервера Apache выполните следующие действия.

1. **Найдите конфигурационный файл Web-сервера** `httpd.conf`.
2. Для этого выберите команду Start⇨Programs⇨Apach HTTP Server⇨Configure Apache Server⇨Edit Configuration (Пуск⇨Программы⇨Apache HTTP Server⇨Configure Apache Server⇨Edit Configuration).
3. Если по каким-то причинам требуемая команда в меню отсутствует, то конфигурационный файл можно поискать на жестком диске. Обычно этот файл содержится в подкаталоге `conf` корневого каталога Apache (например, `c:\Program Files\Apache group\Apache\conf`). Откройте его в любом текстовом редакторе, таком как Notepad или WordPad.
4. **Укажите в конфигурационном файле** `httpd.conf`**, чтобы файлы с кодом на языке PHP передавались интерпретатору PHP. Для этого измените значения двух следующих директив.**

   - `ScriptAlias`. Директива `ScriptAlias` позволяет задать имя-псевдоним полного пути к каталогу, где установлен модуль PHP. Для этого найдите в файле `httpd.conf` строку с директивой `ScriptAlias`. Таких строк может быть несколько, каждая из которых предназначена для различных программных продуктов. Найдите строку, относящуюся к модулю PHP (по необходимости эту строку нужно добавить):

     ```
     ScriptAlias /php/ "c:/php/"
     ```

     Первый аргумент определяет псевдоним (`/php/`), а второй — путь к модулю PHP (`c:/php/`).

   - `Action`. Директива `Action` используется для связывания всех файлов с типом `application/x-httpd-php` с интерпретатором `php-cgi.exe`. В соответствующей строке укажите следующее значение (если такая строка отсутствует, ее нужно создать):

     ```
     Action application/x-httpd-php "/php/php-cgi.exe"
     ```

     Обратите внимание, что в приведенной строке используется имя-псевдоним, определяемое директивой `ScriptAlias`. При изменении значения директивы `ScriptAlias` сервер Apache будет искать файл `php-cgi.exe` в другом каталоге.

При указании путей лучше использовать косую черту (`/`). Web-сервер Apache сможет обрабатывать их без особых проблем.

5. **В конфигурационном файле** `httpd.conf` **укажите, в каких файлах может содержаться код PHP.**

Для этого найдите в файле `httpd.conf` строку с директивой `AddType`. (Этих строк тоже может быть несколько.) Найдите нужную строку, а при ее отсутствии — создайте.

```
AddType application/x-httpd-php .php
```

Посмотрите на эту строку. Если в ее начале содержится символ решетки (#), удалите его. Теперь для обработки файлов с расширением `.php` Web-сервер Apache будет обращаться к интерпретатору PHP. По необходимости можно задать несколько таких расширений.

6. **Запустите или перезапустите заново Web-сервер.**

   Для этого в системе Windows NT/2000/XP выберите команду Start⇨Programs⇨Apach HTTP Server⇨Control Apache Server (Пуск⇨Программы⇨Apache HTTP Server⇨Control Apache Server) и запустите Web-сервер как службу.

   В системе Windows 98/Me выберите команду Start⇨Programs⇨Apach Web Server⇨Management (Пуск⇨Программы⇨Apach Web Server⇨Management).

   Иногда после перезапуска сервера новые параметры не вступают в силу. В таком случае завершите работу сервера Apache, а затем запустите его заново. Зачастую сервер запускается при включении компьютера. Поэтому при возникновении каких-либо проблем выключите компьютер, а затем включите.

Настройка сервера IIS вручную

Если модуль PHP был установлен с помощью программы автоматической инсталляции, то при использовании IIS-сервера версии 5 и ниже все необходимые настройки также выполняются автоматически. При установке модуля PHP вручную дело обстоит иначе: IIS-сервер придется настраивать самостоятельно. Кроме того, ручная настройка версии IIS 6/Windows Server 2003 понадобится в любом случае.

Для настройки IIS-сервера на использование модуля PHP выполните следующие действия.

1. **Откройте управляющую консоль сервера IIS.**

   Для этого выберите команду Start⇨Programs⇨Administrative Tools⇨Internet Services Manager (Пуск⇨Программы⇨Средства администратора⇨Управление службами Internet) или Start⇨Settings⇨Control Panel⇨Administrative Tools⇨Internet Services Manager (Пуск⇨Установки⇨Панель управления⇨Средства администратора⇨Управление службами Internet).

2. **Щелкните правой кнопкой мыши на вашем Web-узле.**
3. **В контекстном меню выберите команду** Properties (Свойства).
4. **Перейдите во вкладку** Home Directory (Домашний каталог).
5. **Щелкните на кнопке** Configuration (Настроить).
6. **Перейдите во вкладку** App Mappings.
7. **Щелкните на кнопке** Add (Добавить).
8. **В поле** Executable **введите путь к интерпретатору PHP, например** `c:\php\php-chi.exe`.
9. **В поле** Extension (Расширение) **введите расширение** `.php`.

   Файлы с этим расширением будут обрабатываться интерпретатором PHP.

10. **Установите флажок** Script Engine.
11. **Щелкните на кнопке** OK.

Повторите шаги 6–10, если необходимо добавить другие расширения файлов, которые будут ассоциироваться с интерпретатором PHP, например `.phtml`.

Приложение Б

Встроенные функции PHP

Вся мощь языка PHP обусловлена множеством предоставляемых им функций, наиболее полезные из которых приведены в данном приложении.

Некоторые из функций уже рассматривались в данной книге. В этом случае вы увидите ссылку на соответствующую главу, в которой та или иная функция рассматривалась более подробно.

Функции для работы с массивами

В этом разделе рассматриваются функции для работы с массивами.

array()

Создает новый массив (см. главу 6).

Синтаксис: `$array = array(`*`ключ=>значение, ключ=>значение, ключ=>значение, ...`*`);`

array_count_values()

Возвращает массив, содержащий количество повторений значений в исходном массиве.

Синтаксис: `$array_out = array_count_values($orig_array);`

Например, если в массиве `$orig_array` содержатся значения

```
$orig_array[a] = Джон
$orig_array[b] = Мэри
$orig_array[c] = Джон
$orig_array[d] = Джоуз
```

то в возвращаемом функцией `array_count_values()` массиве `$array_out` будут содержаться следующие элементы:

```
$array_out[Джон] = 2
$array_out[Мэри] = 1
$array_out[Джоуз] = 1
```

array_diff()

Возвращает массив элементов из исходного массива *$массив1*, которые не содержатся в остальных массивах *$массив2*, *$массив3* и т.д. (см. главу 6).

Синтаксис: `$array_out = array_diff(`*`$массив1, $массив2, $массив3, ...`*`);`

array_intersect()

Возвращает массив с элементами, которые присутствуют в двух или более массивах.

Синтаксис: `$simArarray = array_intersect(`*`$массив1, $массив2, ...`*`);`

array_keys()

Возвращает массив ключей исходного массива $orig_array. Если указан параметр ключ_поиска, то в результирующий массив будут включены только те значения, которые соответствуют выражению ключ_поиска.

Синтаксис: `$array_out = array_keys($orig_array, "ключ_поиска");`

Например, если в массиве $orig_array содержатся значения

```
$orig_array[a] = CA
$orig_array[b] = OR
$orig_array[c] = TX
```

то в возвращаемом массиве $array_out будут содержаться следующие элементы:

```
$array_out[0] = a
$array_out[1] = b
$array_out[2] = c
```

Если же значение параметра ключ_поиска равно OR, т.е. функция array_keys() вызывается следующим образом:

```
$array_out = array_keys($orig_array, OR);
```

то массив $array_out будет содержать элементы

```
$array_out[0] = b
```

array_merge()

Объединяет два или более массивов. Если несколько элементов имеют один и тот же нечисловой ключ, то в результирующий массив будет добавлен лишь последний из них (см. главу 6).

Синтаксис: `$bigArray = array_merge($массив1, $массив2, ...);`

array_merge_recursive()

Объединяет два или более массивов. Если несколько элементов имеют один и тот же нечисловой ключ, то в результирующем массиве будет содержаться массив со всеми этими элементами (см. главу 6).

Синтаксис: `$bigArray = array_merge_recursive($массив1, $массив2, ...);`

array_pop()

Возвращает и удаляет последний элемент массива.

Синтаксис: `$element = array_pop($orig_array);`

array_push()

Добавляет указанные элементы в конец массива и возвращает его новый размер.

Синтаксис: `$new_size = array_push($orig_array, "эл1", "эл2", "эл3");`

array_reverse()

Меняет на обратный порядок следования элементов массива $orig_array.

Синтаксис: `$array_out = array_reverse($orig_array);`

array_search()

Производит поиск *значения* в массиве $orig_array и, если оно найдено, возвращает его ключ, в противном случае — значение FALSE.

Синтаксис: `$key = array_search("значение", $orig_array);`

array_slice()

Создает новый массив $subArray с *количеством* элементов массива $orig_array, начиная с позиции *начало* (см. главу 6).

Синтаксис: `$subArray = array_slice($orig_array, начало, количество);`

array_sum()

Возвращает сумму элементов исходного массива (см. главу 6).

Синтаксис: `$sum = array_sum($orig_array);`

array_unique()

Удаляет из исходного массива повторяющиеся элементы (см. главу 6).

Синтаксис: `$array_out = array_unique($orig_array);`

arsort()

Сортирует массив $orig_array по значениям в обратном порядке с сохранением индексных связей (см. главу 6).

Синтаксис: `arsort($orig_array);`

asort()

Сортирует массив $orig_array по значениям с сохранением индексных связей (см. главу 6).

Синтаксис: `asort($orig_array);`

compact()

Создает массив из заданных значений ($var1, $var2 и т.д.). Этими значениями могут быть как текстовые строки, так и отдельные массивы.

Синтаксис: `$array_out = compact($var1, $var2, ...);`

count()

Возвращает число элементов в массиве $orig_array (см. главу 6).

Синтаксис: `$size = count($orig_array);`

current()

Возвращает текущий элемент массива $array (см. главу 6).

Синтаксис: `$value = current($array);`

end()

Перемещает внутренний указатель массива $array на последний элемент (См. главу 6.)

Синтаксис: `$value = end($array);`

explode()

Создает массив подстрок исходной строки. При разбиении используется символ-разделитель *sep* (зачастую в качестве такого символа используется запятая или символ табуляции) (см. главу 6).

Синтаксис: `$array_out = explode("sep", $string);`

extract()

Создает набор переменных по одной для каждого элемента исходного массива. В качестве имени переменной используется ключ элемента (см. главу 6).

Синтаксис: `extract($array);`

implode()

Возвращает строку, содержащую значения элементов массива, разделенных символом-разделителем *sep* (см. главу 6).

Синтаксис: `$string = explode($array, "sep");`

in_array()

Производит поиск *значения* в массиве `$array`. Если оно найдено, возвращается значение `TRUE`, в противном случае — `FALSE`.

Синтаксис: `$bool = in_array("значение", $array);`

key()

Возвращает ключ текущего элемента массива `$array`.

Синтаксис: `$key = key($array);`

key_exists()

Производит поиск элемента с *ключом* в массиве `$array`. Если такой элемент найден, возвращается значение `TRUE`, в противном случае — `FALSE`.

Синтаксис: `$bool = key_exists("ключ", $array);`

ksort(), krsort()

Сортируют исходный массив по ключам. Функция `ksort()` обеспечивает сортировку в прямом порядке, в `krsort()` — в обратном порядке (см. главу 6).

Синтаксис: `ksort($array); krsort($array);`

natsort(), natcasesort()

Сортируют исходный массив по значениям в "естественном" порядке. Например, результатом сортировки может быть `n1, n2, n12, n25`, а не `n1, n12, n2, n25`, как при обычной сортировке. Функция `natcasesort()` выполняет сортировку с учетом регистра символов.

Синтаксис: `natsort($array); natcasesort($array);`

next()

Перемещает внутренний указатель массива `$array` к следующему элементу (см. главу 6).

Синтаксис: `next($array);`

prev()

Перемещает внутренний указатель массива `$array` к предыдущему элементу (см. главу 6).

Синтаксис: `prev($array);`

range()

Создает массив с элементами из указанного диапазона. Диапазон может быть числовым (например, `1-10`) или символьным (`a-m` или `m-a`).

Синтаксис: `$array_out = range(начало, конец);`

reset()

Устанавливает внутренний указатель массива `$array` на его первый элемент (см. главу 6).

Синтаксис: `reset($array);`

sizeof()

Возвращает число элементов массива `$array` (см. главу 6).

Синтаксис: `$size = sizeof($array);`

sort(), rsort()

Сортируют исходный массив по значениям. Функция `sort()` сортирует элементы по возрастанию, а `rsort()` — по убыванию (см. главу 6).

Синтаксис: `sort($array); rsort($array);`

Функции для работы с датой и временем

В этом разделе содержатся функции для работы с датой и временем.

checkdate()

Возвращает значение TRUE, если дата задана корректно, и FALSE — в противном случае.

Синтаксис: `checkdate(месяц, день, год);`

date(), gmdate()

Возвращает строку, соответствующую временной метке `$timestamp` в формате Unix, которая отформатирована в соответствии со строкой *формат*. Функция `gmdate()` возвращает среднее время по Гринвичу.

Синтаксис: `$formatted_date = date("формат", $timestamp);`

getdate()

Создает массив с такими элементами, как секунды, минуты, месяцы, дни и т.д., которые соответствуют заданной временной метке `$timestamp` в формате Unix.

Синтаксис: `$array_date = getdate($timestamp);`

localtime()

Создает массив с такими элементами, как секунды, минуты, месяцы, дни и т.д., которые соответствуют местному времени.

Синтаксис: `$array_date = localtime($timestamp);`

microtime()

Возвращает время в секундах и микросекундах, прошедшее с 1 января 1970 года.

Синтаксис: `$time_out = microtime();`

mktime(), gmmktime()

Возвращает временную метку Unix для заданной даты. Функция `gmmktime()` возвращает среднее время по Гринвичу (см. главу 5).

Синтаксис: `$timestamp = mktime(часы, минуты, секунды, месяц, день, год);`

time()

Возвращает временную метку Unix для текущего времени (см. главу 5).

Синтаксис: `$timestamp = time();`

Функции для работы с файловой системой

В этом разделе рассматриваются функции для работы с файловой системой.

basename()

Извлекает полное имя файла из указанного *пути* (см. главу 13).

Синтаксис: `$filename = basename("путь");`

chdir()

Изменяет текущий каталог на *путь_к_каталогу* (см. главу 13).
Синтаксис: `chdir("путь_к_каталогу");`

chgrp()

Изменяет группу, которой принадлежит файл.
Синтаксис: `chgrp("путь_к_файлу", "группа");`

chmod()

Изменяет уровень доступа к файлу.
Синтаксис: `chmod("путь_к_файлу", "восьмеричное_число");`

chown()

Изменяет владельца файла.
Синтаксис: `chown("путь_к_файлу", "новый_владелец");`

closedir()

Закрывает каталог с дескриптором `$dh` (см. главу 13).
Синтаксис: `closedir($dh);`

copy()

Создает копию файла (см. главу 13).
Синтаксис: `copy("старый_файл", "новый_файл");`

dirname()

Извлекает из заданного *пути* имя каталога (см. главу 13).
Синтаксис: `$directory_name = dirname("путь");`

dis_total_space()

Возвращает общий объем диска в байтах.
Синтаксис: `$space = dis_total_space("путь");`

disk_free_space()

Возвращает объем свободного дискового пространства в байтах.
Синтаксис: `$free = disk_free_space("путь");`

fclose()

Закрывает открытый файл с дескриптором `$fh` (см. главу 12).
Синтаксис: `fclose($fh);`

feof()

Возвращает значение TRUE при достижении конца файла с идентификатором `$fh` (см. главу 12).
Синтаксис: `feof($fh);`

fgetc()

Возвращает один символ (текущий) из файла с дескриптором `$fh` и перемещает указатель на следующий символ.
Синтаксис: `$char = fgetc($fh);`

fgetcsv()

Считывает строку не больше заданной *длины* символов из файла, разбивает ее с учетом символа-разделителя *sep* и возвращает массив с полученными подстроками (см. главу 12).
Синтаксис: `$array_out = fgetcsv($fh, длина, "sep");`

fgets(), fgetss()

Считывает из файла с идентификатором `$fh` строку *длиной* не больше заданного значения. Кроме того, функция `fgetss()` удаляет дескрипторы (см. главу 12).

Синтаксис: `$line = fgets($fh, длина); $line = fgetss($fh, длина);`

file()

Возвращает массив, каждый элемент которого содержит отдельные строки заданного файла (см. главу 12).

Синтаксис: `array_lines = file($fh);`

file_exists()

Проверяет существование файла (см. главу 13).

Синтаксис: `$bool = file_exists("путь_к_файлу");`

fileatime()

Возвращает время последнего доступа к файлу (см. главу 13).

Синтаксис: `$timestamp = fileatime("путь_к_файлу");`

filectime()

Возвращает время создания файла (см. главу 13).

Синтаксис: `$timestamp = filectime("путь_к_файлу");`

filemtime()

Возвращает время последней модификации файла (см. главу 13).

Синтаксис: `$timestamp = filemtime("путь_к_файлу");`

fileowner()

Возвращает идентификатор пользователя-владельца файла (см. главу 13).

Синтаксис: `$userID = fileowner("путь_к_файлу");`

fileperms()

Возвращает разрешения, связанные с файлом.

Синтаксис: `$perms = fileperms("путь_к_файлу");`

filesize()

Возвращает размер файла в байтах (см. главу 13).

Синтаксис: `$size = filesize("путь_к_файлу");`

filetype()

Возвращает тип файла, например файл или каталог (см. главу 13).

Синтаксис: `$type = filetype("путь_к-файлу");`

flock()

Блокирует файл (см. главу 12).

Синтаксис: `filelock("путь_к_файлу");`

fopen()

Открывает файл в заданном *режиме* и возвращает его дескриптор (см. главу 12).

Синтаксис: `$fh = fopen("путь_к_файлу", режим);`

fputs()

Записывает строку в файл. Возвращает количество записанных байт или значение `FALSE`, если возникла ошибка (см. главу 12).

Синтаксис: `$result = fputs($fh, "текст", длина);`

fread()

Считывает из файла заданное *количество* байт, если раньше не будет достигнут его конец (см. главу 13).

Синтаксис: `$file_content = fread($fh, количество);`

fscanf()

Считывает содержимое файла и возвращает отформатированную строку. (О способах форматирования см. в главе 13.)

Синтаксис: `$string = fscanf($fh, "формат", $v1, $v2, ...);`

fseek()

Перемещает указатель в файле с идентификатором `$fh` на заданное *число* байт. Параметр *режим* может принимать следующие значения: `SEEK_SET` (смещение осуществляется от начала файла), `SEEK_CUR` (с текущей позиции) и `SEEK_END` (от конца файла к его началу).

Синтаксис: `fseek($fh, число, режим);`

fwrite()

Записывает строку *текста* в файл с дескриптором `$fh`. Параметр *длина* является необязательным (см. главу 12).

Синтаксис: `$bytes_written = fwrite($fh, "текст", длина);`

getcwd()

Возвращает полный путь, соответствующий текущему каталогу.

Синтаксис: `$current_directory = getcwd();`

getlastmod()

Возвращает дату последней модификации текущего сценария.

Синтаксис: `$timestamp = getlastmod();`

is_dir()

Возвращает значение `TRUE`, если *путь* задает каталог, и `FALSE` — в противном случае (см. главу 13).

Синтаксис: `$bool = is_dir("путь");`

is_file()

Возвращает значение `TRUE`, если *путь* определяет обычный файл, и `FALSE` — в противном случае (см. главу 13).

Синтаксис: `$bool = is_file("путь");`

is_readable()

Возвращает значение `TRUE`, если *путь* определяет доступный для чтения файл, и `FALSE` — в противном случае (см. главу 13).

Синтаксис: `$bool = is_readable("путь_к_файлу");`

is_uploaded()

Проверяет, был ли файл загружен через Web с использованием форм.

Синтаксис: `$bool = is_uploaded("путь_к_файлу");`

is_writable()

Возвращает значение `TRUE`, если *путь_к_файлу* задает доступный для записи файл, и `FALSE` — в противном случае (см. главу 13).

Синтаксис: `$bool = is_writable("путь_к_файлу");`

mkdir()

Создает новый каталог. *Режим* в восьмеричном формате определяет связываемые с ним разрешения.

Синтаксис: `mkdir("путь_к_новому_каталогу", режим);`

move_uploaded_file()

Перемещает загруженный файл из временного каталога в постоянный (см. главу 11).

Синтаксис: `move_uploaded_file("имя_файла", "путь");`

opendir()

Открывает каталог и возвращает соответствующий дескриптор (см. главу 13).

Синтаксис: `$dh = opendir("путь_к_каталогу");`

passthru()

Вызывает системную команду и выводит результат (см. главу 13).

Синтаксис: `passthru("системная_команда");`

pathinfo()

Возвращает ассоциативный массив с информацией о пути. В этом массиве содержатся три элемента: `dirname` (каталог), `basename` (полное имя файла) и `extension` (расширение).

Синтаксис: `$array_dir = pathinfo("путь");`

readdir()

Считывает одно имя файла из открытого каталога (см. главу 13).

Синтаксис: `$filename = readdir($dh);`

readfile()

Считывает файл и выводит его содержимое на стандартное устройство вывода. Возвращает количество прочитанных байтов. Эта функция позволяет использовать и адреса URL.

Синтаксис: `$numberOfBytesRead = readfile("путь_к_файлу");`

rename()

Переименовывает файл.

Синтаксис: `rename("старое_имя", "новое_имя");`

rewind()

Перемещает указатель файла с дескриптором `$fh` на его начало.

Синтаксис: `rewind($fh);`

rmdir()

Удаляет каталог (см. главу 13).

Синтаксис: `rmdir("путь_к_каталогу");`

tmpfile()

Создает временный файл с уникальным именем, открывает его с правом записи и возвращает соответствующий дескриптор.

Синтаксис: `$fh = tmpfile();`

touch()

Устанавливает время модификации файла. Если *время* не задано, используется текущее время. Если файл не существует, создается новый файл.

Синтаксис: `$bool = touch("путь_к_файлу", время);`

unmask()

Изменяет разрешения, заданные по умолчанию, в соответствии с заданной *маской* и возвращает старое значение. По завершении работы сценария старые значения восстанавливаются.

Синтаксис: `$old_mask= unmask(маска);`

unlink()

Удаляет файл (см. главу 13).

Синтаксис: `unlink("путь_к_файлу");`

Функции для работы с протоколом HTTP и электронной почтой

В этом разделе рассматриваются функции для работы с протоколом HTTP и электронной почтой.

get_browser()

Позволяет получить информацию о текущем клиентском браузере или о браузере с заданным *именем*.

Синтаксис: `$string = get_browser("имя");`

get_meta_tags()

Создает массив, в элементах которого содержатся имена атрибутов всех дескрипторов <meta>, обнаруженных в заданном файле.

Синтаксис: `$array_tags = get_meta_tags("путь_к_файлу");`

header()

Отправляет заголовок HTTP Web-серверу (см. главу 10).

Синтаксис: `header("заголовок_HTTP");`

mail()

Отправляет сообщение по электронной почте из сценария PHP (см. главу 13).

Синтаксис: `$success = mail("кому", "тема_сообщения", "сообщение", "заголовки");`

parse_url()

Возвращает массив, в элементах которого содержатся различные элементы URL-адреса, такие как имя узла, путь, порт, имя пользователя и т.д.

Синтаксис: `$array_url = parse_url("адрес_URL");`

setcookie()

Устанавливает данные cookie (см. главу 10).

Синтаксис: `setcookie("имя", "значение", время_жизни, "путь", "домен", уровень_безопасности);`

Математические функции

В этом разделе приведены основные математические функции. Такие функции, как cos (косинус), tan (тангенс) или pi (значение числа π), в данном разделе не представлены.

abs()

Возвращает абсолютную величину числа.
Синтаксис: `$absolute = abs(число);`

bindec()

Преобразует двоичное число в десятичное.
Синтаксис: `$number_decimal = bindec(двоичное_число);`

exp()

Возвращает значение константы e, возведенное в *степень*.
Синтаксис: `$number = exp(степень);`

floor()

Округляет *число* до ближайшего меньшего целого.
Синтаксис: `$int = floor(число);`

hexdec()

Преобразует шестнадцатеричное число в десятичное.
Синтаксис: `$number_decimal = hexdec(число);`

log()

Возвращает натуральный логарифм от заданного числа.
Синтаксис: `$log = log(число);`

log10()

Возвращает логарифм по основанию 10 от заданного числа.
Синтаксис: `$log10 = log10(число);`

max()

Возвращает максимальное число из заданного множества. В качестве входного параметра этой функции можно передать массив.
Синтаксис: `$num_large = max($array);` или `$num_large = max(число1, число2, ...);`

min()

Возвращает минимальное число из заданного множества. В качестве входного параметра этой функции можно передать массив.
Синтаксис: `$num_min = min($array);` или `$num_min = min(число1, число2, ...);`

number_format()

Форматирует *число* на основе заданного десятичного символа-разделителя и разделителя тысяч. По умолчанию используются десятичная точка (.) и запятая (,) (см. главу 5).
Синтаксис: `$formatted = number_format(число, "десятичный_символ", "разделитель_тысяч");`

octdec()

Преобразует восьмеричное число в десятичное представление.
Синтаксис: `$number_decimal = octdec(число);`

pow()

Возвращает *число*, возведенное в *степень*.
Синтаксис: `$result = pow(число, степень);`

rand()

Генерирует случайную величину из заданного диапазона.
Синтаксис: `$number_rand = rand(`*минимум, максимум*`);`

round()

Округляет *число* с точностью до заданного количества десятичных цифр.
Синтаксис: `$result = round(`*число, количество_цифр*`);`

sqrt()

Возвращает квадратный корень от заданного аргумента.
Синтаксис: `$square_root = sqrt(`*число*`);`

srand()

Инициализирует генератор случайных чисел с указанным начальным значением.
Синтаксис: `srand(`*начальное_значение*`);`

Функции для работы с параметрами PHP

В этом разделе приведены функции для работы с параметрами PHP.

getenv()

Возвращает значение переменной окружения.
Синтаксис: `$environment_value = getenv("`*имя_переменной_окружения*`");`

getlastmod()

Возвращает время последней модификации текущего сценария.
Синтаксис: `$timestamp = getlastmod();`

ini_get()

Возвращает значение конфигурационного *параметра*.
Синтаксис: `$string = ini_get("`*параметр*`");`

ini_set()

Устанавливает значение конфигурационного *параметра* (см. главу 4).
Синтаксис: `ini_set("`*параметр*`", "`*значение*`");`

phpinfo()

Предоставляет информацию о версии и параметрах модуля PHP (см. главу 4).
Синтаксис: `phpinfo();`

phpversion()

Возвращает текущую версию PHP.
Синтаксис: `$version = phpversion();`

putenv()

Добавляет переменную окружения. Обычно *значение* задается в виде *имя_переменной=значение*.
Синтаксис: `putenv("`*значение*`");`

Строковые функции

В этом разделе приведены функции для работы со строками.

addslashes()

Возвращает строку без одинарных/двойных кавычек, обратной косой черты и символа \0.
Синтаксис: `$string_escaped = addslashes("строка");`

base64_encode(), base64_decode()

Шифрует/декодирует строку с использованием алгоритма base64 (см. главу 13).
Синтаксис: `$string_encoded = base64_encode("строка");`

chop()

Удаляет пробелы в конце строки.
Синтаксис: `$chopped = chop("строка");`

chr()

Возвращает символ ASCII, соответствующий заданному коду.
Синтаксис: `$char = chr(код_ASCII);`

count_chars()

Возвращает ассоциативный массив, ключами которого являются символы строки `$string`, а значениями — их количество в заданной строке. Необязательный параметр *критерий* позволяет выбрать символы только с нулевой или ненулевой частотой появления и т.д. По умолчанию в ассоциативном массиве размещаются все символы строки `$string`.
Синтаксис: `$array = count_chars($string, критерий);`

echo()

Выводит одну или более строк (см. главу 3).
Синтаксис: `echo строка1, строка2, строка3, ...;`

ereg(), eregi()

Выполняет поиск подстроки в строке `$string`, которая соответствует заданному шаблону. При этом функция `eregi()` не учитывает регистр символов (см. главу 7).
Синтаксис: `$bool = ereg("шаблон", $string);`

ereg_replace(), eregi_replace()

Находит фрагмент строки `$string`, соответствующий указанному шаблону, и заменяет его новой строкой. Функция `ereg_replace()` зависит от регистра символов, а функция `eregi_replace()` — нет (см. главу 7).
Синтаксис: `$new_string = ereg_replace(ereg_replace("шаблон", "новая_строка", $string);`

explode()

Создает массив, элементами которого являются части строки `$string`, отделенные друг от друга символом-разделителем.
Синтаксис: `$array_out = explode("разделитель", $string);`

htmlentities()

В строке `$orig_string` переводит все объекты HTML в специальные символы.
Синтаксис: `$string_out = htmlentities($orig_string);`

htmlspecialchars()

В строке `$string` преобразует специальные символы в объекты HTML, как, например, & — в &amp.

Синтаксис: `$string_out = htmlspecialchars($string);`

implode()

Объединяет элементы массива `$array` в одну строку и разделяет их с помощью *разделителя*.

Синтаксис: `$string_out = implode($array, "разделитель");`

nl2br()

В заданной строке `$string` вставляет дескриптор `<br />` перед каждым символом перевода строки (`\n`).

Синтаксис: `$string_out = nl2br($string);`

ord()

Возвращает ASCII-значение первого символа строки.

Синтаксис: `$integet = ord("строка");`

parse_url()

Создает ассоциированный массив, в элементах которого содержатся различные части URL-адреса.

Синтаксис: `$array = parse_url("адрес_URL");`

print()

Выводит строку, число или значение переменной.

Синтаксис: `print параметр;`

printf()

Выводит строку в заданном формате (см. главу 5).

Синтаксис: `printf("формат", аргумент1, аргумент2, аргумент3, ...);`

split(), spliti()

Создает массив, в элементах которого размещаются результаты разбиения строки `$string` на отдельные фрагменты с учетом заданного регулярного выражения *шаблон*. Функция `spliti()` не учитывает регистр символов.

Синтаксис: `$array = split("шаблон", $string);`
`$array = spliti ("шаблон", $string);`

sprintf()

Возвращает строку, преобразованную в соответствии с заданным *форматом* (см. главу 5).

Синтаксис: `$string = sprintf("формат", аргумент1, аргумент2, аргумент3, ...);`

str_pad()

Удлиняет исходную строку `$string` до заданного размера.

Синтаксис: `$string = sprintf($string, длина, "заполнитель");`

str_repeat()

Возвращает строку `$string`, продублированную заданное *число* раз.

Синтаксис: `$string_out = str_repeat($string, число);`

str_replace()

Заменяет все вхождения *старого_текста* в строке `$string` на *новый_текст*.

Синтаксис: `$string_out = str_replace("старый_текст", "новый_текст", $string);`

strchr(), strrchar()

Функция `strchr()` возвращает часть строки `$string`, начиная с первого вхождения *символа* и до ее конца, а `strrchar()` — начиная с заданного символа и до начала строки.

Синтаксис: `$string_part = strchr($string, "символ");`

strcmp(), strcasecmp()

Эти функции выполняют сравнение двух строк. Если строка `$str1` длиннее `$str2`, возвращается значение 1, в противном случае — значение –1. Если обе строки имеют одинаковую длину, возвращается значение 0. Функция `strcmp()` учитывает регистр символов, а `strcasecmp()` — нет.

Синтаксис: `strcasecmp($str1, $str2);`

strcspn()

Возвращает позицию первого вхождения заданного символа в строку `$string`.

Синтаксис: `$int = strcspn($string, "символ");`

strip_tags()

Удаляет из строки `$string` дескрипторы HTML и PHP. Необязательный параметр *разрешенные_дескрипторы* позволяет задать символы, которые удаляться не будут (см. главу 10).

Синтаксис: `$string_stripped = strip_tags($string, "разрешенные_дескрипторы");`

strlen()

Возвращает длину строки `$string` в символах.

Синтаксис: `$length = strlen($string);`

strpos(), strrpos()

Функция `strpos()` возвращает позицию первого вхождения заданного символа в строку `$string`, а `strrpos()` — последнего.

Синтаксис: `$integer = strpos($string, "символ");`
`$integer = strrpos($string, "символ");`

strstr(), stristr()

Возвращает часть строки `$string`, начиная с первого появления указанного символа и до ее конца. Функция `strstr()` учитывает регистр символов, а функция `stristr()` — нет.

Синтаксис: `$str_part = strstr($string, "символ");`
`$str_part = stristr($string, "символ");`

strtolower(), strtoupper()

Преобразуют строку `$string` в нижний и верхний регистр соответственно.

Синтаксис: `$str_lower = strtolower($string);`
`$str_upper = strtoupper($string);`

strtr()

Эта функция заменяет в строке $string каждую подстроку *старый_текст* на подстроку *новый_текст*.

Синтаксис: `$string_out = strtr($string, "новый_текст", "старый_текст");`

substr()

Возвращает часть строки $string с заданной позиции и длиной *количество_символов*.

Синтаксис: `$string_new = substr($string, начало, количество_символов);`

substr_replace()

Заменяет часть строки $string *новой_строкой* с начальной позиции и длиной *количество_символов*.

Синтаксис: `$string_new = substr_replace($string, "новая_строка", начало, количество_символов);`

trim(), ltrim(), rtrim()

Функция trim() удаляет все пробелы в начале и в конце строки $string, ltrim() — только в начале, а rtrim() — только в конце.

Синтаксис: `$string_new = trim($string);`

ucfirst()

Переводит первый символ строки $string в верхний регистр.

Синтаксис: `$string_new = ucfirst($string);`

ucwords()

Переводит в верхний регистр первые символы всех слов строки $string.

Синтаксис: `$string_new = ucwords($string);`

wordwrap()

Вставляет в строку $string символ перевода строки (\r\n) через указанное количество символов и до конца строки.

Синтаксис: `$string_new = wordwrap($string, количество_символов);`

Функции для работы с переменными

В этом разделе приведены функции для работы с переменными.

empty()

Определяет, присвоено ли переменной $varname какое-либо значение (см. главу 7).

Синтаксис: `$bool = empty($varname);`

get_defined_classes()

Создает массив с именами всех классов, используемых в сценарии (в том числе и в подключаемых файлах).

Синтаксис: `$array_classes = get_defined_classes();`

get_defined_constants()

Возвращает ассоциированный массив с именами всех констант.

Синтаксис: `$array_constants = get_defined_constants();`

get_defined_functions()

Возвращает ассоциированный массив с именами всех функций.

Синтаксис: `$array_functions = get_defined_functions();`

get_defined_vars()

Возвращает массив с именами всех переменных.

Синтаксис: `$array_vars = get_defined_vars();`

isset()

Проверяет, существует ли переменная `$varname` (см. главу 7).

Синтаксис: `$bool = isset($varname);`

print_r()

Выводит содержимое переменной `$varname` (см. главу 4).

Синтаксис: `print_r($varname);`

putenv()

Добавляет переменную окружения. Обычно *параметр* задается в виде *имя_переменной=значение*.

Синтаксис: `putenv("параметр");`

serialize()

Преобразует данные в двоичную строку. Эта функция оказывается полезной при сохранении данных в файле или в базе данных. В качестве параметра `$variable` можно использовать любой тип данных, включая объект или функцию.

Синтаксис: `$string_ser = serialize($variable);`

unserialize()

Восстанавливает сериализованные данные в первоначальном виде.

Синтаксис: `$variable = unserialize($string_ser);`

unset()

Удаляет переменную с именем `$varname` (см. главу 4).

Синтаксис: `unset($varname);`

var_dump()

Выводит содержимое переменной (см. главу 4).

Синтаксис: `var_dump($varname);`

Предметный указатель

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Э

Я

Шпаргалка

Специальные символы, используемые в шаблонах

Символ	*Назначение*	*Пример*	*Соответствует шаблону*	*Не соответствует шаблону*
^	Начало строки	^э	экзамен	математический экзамен
$	Конец строки	н$	экзамен	экзамены
.	Любой одиночный символ	..	до, от Более длинные слова также соответствуют этому шаблону, поскольку включают текстовую строку из двух символов	а, 2
?	Предшествующий символ является не обязательным	ger?m	germ, gem	geam
( )	Группирует буквенные символы в последовательность, которую в точности должна содержать строка	g(er)m	germ	Gem, grem
[ ]	Включает набор необязательных символов	g[er]m	gem, grm	germ, gel
[^]	Определяет все символы, кроме указанных	g[^er]m	gym, gum	gem, grem, germ
-	Включает диапазон всех символов, заключенных между двумя (ряд возможных символов)	g[a-c]m	gam, gbm, gcm	gdm, gxm, gal
+	Один или несколько наборов предшествующих символов	bldg[1-3]+	bldg111, bldg132	bldg, bldg555
*	Один, ни одного или несколько вхождений предшествующего символа	ge*m	gm, geeem	germ, grm
{n}	Повторение *n* раз	ge{5}m	geeeeem	geeeem, geeeeeem
	Определяет диапазон повторений символа (символов)	a{2,5}	aa, aaa, aaaa, 145aaaaa	1, a3
\	Определяет буквенный символ	g*m	g*m	gem, germ
(| |)	Набор альтернативных строк	(Сэм|Салли)	Салли	Сара, Салмон

BESTSELLING BOOK SERIES

PHP5 для "чайников"™

СЕРИЯ КОМПЬЮТЕРНЫХ КНИГ ОТ ДИАЛЕКТИКИ

Шпаргалка

Создание массивов

Метод создания массива	Пример кода	Результат
Использование пустых скобок	`$colors[] = "красный";` `$colors[] = "синий";`	`$colors[1] = красный;` `$colors[2] = синий;`
Использование ключей	`$paint['house'] = "синий";` `$paint['barn'] = "красный";`	`$paint[house] = синий;` `$paint[barn] = красный;`
Использование функции `array`	`$colors = array("синий", "красный");`	`$colors[1] = синий;` `$colors[2] = красный;`
Использование функции `array`	`$colors = array(5 => "синий", "красный");`	`$colors[5] = синий;` `$colors[6] = красный;`
Использование функции `array` с ключами	`$paint = array("barn" => "красный", "house" => "зеленый");`	`$paint[barn] = красный;` `$paint[house] = зеленый;`

Многомерный массив	`$paint['house']['tall'] = "желтый";`
	`$paint['barn']['tall'] = "белый";`
	`$paint['house']['short'] = "синий";`
	`$paint['barn']['short'] = "красный";`
		`$paint[house][tall] = желтый;`
		`$paint[barn][tall] = белый;`
		`$paint[house][short] = синий;`
		`$paint[barn][short] = красный;`

Операции сравнения

Операция	Описание
`==`	Равнозначны ли значения двух переменных?
`===`	Одинаковы ли как значения, так и типы двух переменных?
`>`	Больше ли первое значение, чем второе?
`>=`	Верно ли, что первое значение не меньше второго?
`<`	Меньше ли первое значение, чем второе?
`<=`	Верно ли, что первое значение не больше второго?
`!=`	Не равны ли значения двух переменных?
`!==`	Не одинаковы ли значения или типы данных двух переменных?
`<>`	Не одинаковы ли значения или типы данных двух переменных?

Правила именования переменных PHP

- Должны начинаться с символа доллара (`$`)
- Могут быть любой длины
- Могут включать буквы, цифры и символы подчеркивания
- Должны начинаться с буквы или символа подчеркивания, но не цифры
- Символы верхнего и нижнего регистров различаются

Научно-популярное издание

Джанет Валейд

PHP 5 для "чайников"

В издании использованы карикатуры
американского художника Рича Теннанта

Литературный редактор	*И.А. Попова*
Верстка	*В.И. Бордюк*
Художественные редакторы	*В.Г. Павлютин, Т. Тараброва*
Корректоры	*З.В. Александрова, Л.А. Гордиенко, Л.В. Чернокозинская*

Издательский дом "Вильямс".
101509, Москва, ул. Лесная, д. 43, стр. 1.

Подписано в печать 07.06.2005. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 25,8. Уч.-изд. л. 18,72.
Тираж 3000 экз. Заказ № 1879.

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати, телерадиовещания
и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.