2

Написание простых программ

Читатель, вероятно, к этому моменту уже установил PHP и все остальные программные компоненты, необходимые для того, чтобы приступить к использованию PHP 5. Кроме того, читатель должен знать, что большинство создаваемых им программ предназначены для работы на Web-сервере и в них используются HTML- или XHTML-страницы для отображения пользовательского интерфейса и результатов обработки данных в браузере (для PC-пользователей браузером, скорее всего, будет Internet Explorer).

В данной главе рассматриваются основополагающие принципы написания PHP-программ, взаимодействие с которыми пользователь осуществляет через браузер. Здесь описаны основные аспекты правильного написания PHP-программ, в частности, вставки PHP-кода в HTML- или XHTML-страницы, использование нескольких широко распространенных PHP-функций (таких как `echo`, `date()`, `strlen()` и т.д.), а также создание и использование переменных. В данной главе представлено несколько примеров разработки простых программ, демонстрирующих типичное использование PHP 5 в простых Web-страницах, а также кратко описана работа PHP-программ.

Материал данной главы знакомит читателя с операторами и выражениями, а также с исключительно полезными переменными, которые называются массивами. Массивы, как и обычные переменные, применяются для хранения данных, однако они имеют ряд функций, которые делают их весьма мощным средством для различных типов обработки данных.

В этой главе представлены фундаментальные понятия для материала последующих глав, поскольку почти вся обработка данных в PHP-программах зависит непосредственно от умения программиста соответствующим образом именовать, а также использовать переменные и связанные с типами данных PHP-функции и особенно массивы.

Создание PHP-программы

Для начала напишем Web-страницу, которая будет отображаться в любом существующем браузере. Для этого выполните следующие действия.

1. Откройте программу Блокнот или любой другой доступный текстовый редактор и наберите в нем следующий HTML-код:

```
<html>
  <head>
    <title>Web-страница</title>
  </head>
  <body>
    Этот текст появляется в окне браузера
  </body>
</html>
```

2. Сохраните данный файл с каким-либо именем и расширением `.htm` (например, `simple01.htm`).
3. Откройте файл в браузере. Результат показан на рис. 2.1.

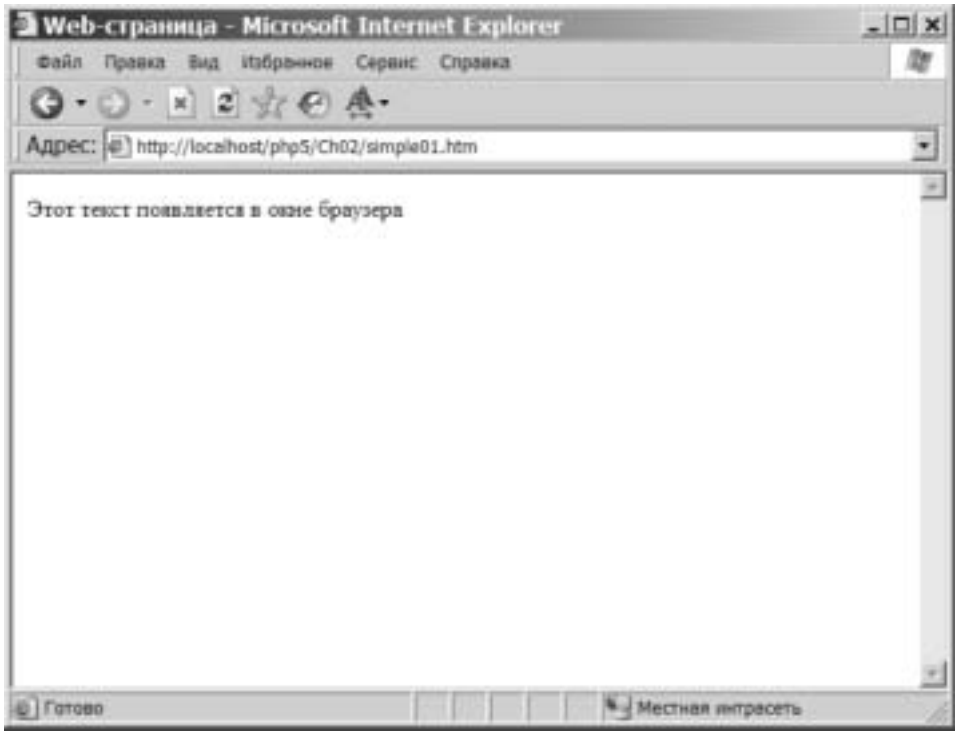

Рис. 2.1.

4. В зависимости от инсталляции и настройки, описанной в главе 1, загрузите файл в соответствующий каталог сервера (если Web-сервер работает на локальной машине, то файл следует просто скопировать в каталог, обслуживаемый

Web-сервером). Снова откройте файл в браузере, используя HTTP-адрес для локального узла. Результат должен выглядеть аналогично.

5. Теперь замените строку, начинающуюся с "Этот текст ...", следующими строками:

```
Этот текст представляет данные, полученные в результате работы PHP 5: Сегодня
<?php
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" . date("Y",time());
echo $todaysdate;
?>
```

6. Сохраните файл, скопируйте его на сервер, если необходимо, и обновите страницу в браузере. Возможно, ничего не изменилось, если расширение файла не было изменено с `.htm` на `.php`. Очевидно, что для того чтобы Web-сервер передал данный файл PHP-процессору, во-первых, он (сервер) должен определить, что данный файл является PHP-файлом, а определить это можно по расширению файла. Во-вторых (предположим, что файл имеет соответствующее расширение), PHP-процессор выбирает для обработки разделы PHP-кода путем *синтаксического анализа* этого файла и поиска в нем PHP-тегов (`<?php` и `?>`), а затем *выполняет* PHP-код. Синтаксический анализ означает, что PHP-процессор считывает отдельные команды и проверяет их на синтаксические ошибки (синтаксические ошибки — весьма распространенное явление при создании PHP-кода). Под выполнением кода следует понимать просто фактическую обработку кода PHP-процессором.

7. Измените расширение файла на `.php` и введите его адрес в браузере. На этот раз код должен работать и в браузере должна появиться текущая дата в конце строки текста.

Некоторые детали

В PHP, как и в любом другом языке программирования, существует множество команд, ключевых слов, операторов, языковых конструкций и функций. PHP-процессор ищет эти элементы в процессе синтаксического анализа и, если синтаксис корректен, то обрабатывает их, а затем возвращает результат обработки. Когда выполняется Web-приложение, результаты должны быть совместимы с HTML или языками написания сценариев на Web-странице, отображающей результаты. Поэтому часто результаты обработки включают в себя HTML-теги (подробнее о том, как работает HTML, рассказывается в главе 3).

Функция `echo`, по сути, не является функцией — она представляет собой конструкцию языка, т.е. ей не требуется передавать параметры в скобках, она не возвращает никаких значений (однако она делает именно то, что предполагается — отправляет в браузер строковое значение), а, кроме того, имеет несколько других ограничений, не свойственных функциям. Конструкция `echo` — не единственная языковая конструкция в PHP; другим примером является `unset`. В данном случае важно понимать то, что `echo` отправляет браузеру строку; в последующих главах различия между языковыми конструкциями и функциями рассматриваются более подробно.

Итак, команда `echo` отправляет пользовательскому браузеру строковые данные. В состав этих данных может включаться HTML-код, но если его нет, то пользователь увидит в браузере текст (браузер интерпретирует такие данные как текстовый файл

и "подставляет" HTML-код, чтобы данные отображались как обычно). В следующем простом примере `echo` отправляет строковые данные, сформированные на основании текущей даты и записанные в переменную `$todaysdate`:

```
echo $todaysdate;
```

Функция `date()` встроена в PHP, т.е. нет необходимости создавать или копировать эту функцию в разрабатываемой PHP-программе; она доступна для использования в любой момент. Она вызывается так же, как и функции в большинстве языков — указывается имя функции, за которым следуют круглые скобки. В скобках указываются аргументы — значения, выражения или другие функции, возвращающие значения, которые затем используются функцией `date()` для вычисления окончательного результата. Рассмотрим следующую строку из приведенного ранее примера программы:

```
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" . date("Y",time());
```

Первый аргумент `"m"` сообщает функции `date()` о том, что она должна ввернуть порядковый номер месяца. В качестве второго аргумента используется функция `time()`, которая возвращает текущее время. Функция `date()` обрабатывает текущее время и извлекает из него номер месяца в виде значения, состоящего из двух символов. Функцию `date()` можно использовать снова, для того чтобы получить день и месяц из значения, возвращаемого функцией `time()`.

Как работает PHP-код

Как следует из предыдущего примера, для запуска PHP-программ внутри Web-страницы необходимо выполнение следующих основных требований:

- наличие Web-страницы, с которой может взаимодействовать пользователь;
- файл с расширением `.php`;
- опознаваемые PHP-теги;
- синтаксически корректный PHP-код.

Рассмотрим каждое требование подробнее.

Web-страница (пользовательский интерфейс)

Важно четко представлять себе, как отображается Web-страница, а также то, как работает PHP 5, поскольку отображаемая страница — это все, что видит конечный пользователь PHP-программы. Уместно подчеркнуть, что, хотя отображаемый вывод часто называют HTML-кодом, HTML сдает свои позиции языку XHTML, а браузеры поддерживают обработку другого подобного вывода, например, XML с XSLT. Несмотря на то, что в данной книге для обозначения вывода, отправляемого Web-сервером, используется аббревиатура HTML, под выводом также следует понимать код других языков, который может быть визуализирован браузером.

Можно писать PHP-программы так, чтобы PHP-код встраивался непосредственно в HTML-код, или же можно по мере необходимости ссылаться на HTML-код из PHP-кода. В любом случае весь вывод, отправляемый конечному пользователю, должен представлять собой HTML-код.

Почему? Попытаемся открыть `.php` не как Web-страницу, а как файл (для этого в браузере необходимо использовать пункт меню Файл⇨Открыть и выбрать файл на жестком диске). При этом в браузере PHP-код не отображается (браузеры игнорируют

PHP-теги и все, что находится между ними), однако, воспользовавшись меню Вид⇨ Просмотр HTML-кода, можно увидеть необработанный PHP-код. Это связано с тем, что ни Web-сервер, ни PHP-процессор не обрабатывали данный файл до того, как он был открыт в браузере.

Расширения файлов

Если вы уже создавали Web-страницы, скорее всего, вы знакомы с расширениями `.htm` и `.html`, а также, возможно, `.shtml`. С помощью данных расширений браузер распознает тип открываемого файла. Web-сервер также способен распознавать расширения файлов, и если он встречает `.php`-файл, то передает его PHP-машине для обработки. Расширение `.php` появилось не в PHP 5, а является стандартным расширением PHP-файлов для большинства Web-серверов.

Существует возможность так сконфигурировать Web-сервер, чтобы он отправлял PHP-процессору файлы с другими расширениями (например, `.htm` или `.html`). При использовании такой конфигурации обычные HTML-страницы будут обрабатываться PHP-машиной (в дополнение ко всем PHP-страницам), хотя обычные HTML-файлы не изменяются в результате этой обработки (это просто вносить некоторые издержки). Так как файлы, отправленные пользователям, будут иметь расширения `.htm` или `.html` вместо `.php`, пользователь не узнает, что для серверной обработки данных файлов используется PHP. Добавлять эти расширения или нет — дело выбора разработчика, однако данный способ рекомендуется в тех случаях, когда от пользователя необходимо скрыть PHP-природу Web-приложения.

PHP-разделители

Разделители используются в различных типах кода для указания блоков кода, данных и т.д. Разделители представляют собой специальные символы, которые сообщают программе синтаксического анализа начало и окончание данных. Так, в формате с разделяющими запятыми разделителями являются запятые. Программа или процессор, анализирующий синтаксис потока данных, "знает", что между двумя соседними запятыми должны быть данные соответствующего типа.

Та же идея применяется к PHP-коду, встроенному в Web-страницу. Стандартными разделителями для PHP 5 являются последовательности символов `<?php` и `?>`. Можно также использовать в качестве разделителей последовательности `<?` и `?>`, но `<?php` и `?>` более предпочтительны — они стандартизированы в репозитории PHP-расширений и приложений (PHP Extension and Application Repository — PEAR), который является хорошим источником стандартного PHP-кода и предметом обсуждения главы 14.

Как и другие настройки в PHP 5, разделители, которые будут распознаваться PHP-машиной, можно установить или расширить путем редактирования конфигурационного файла. Например, можно заставить PHP 5 распознавать в качестве разделителей символы `<%` и `%>`. Такие разделители называются ASP-разделителями, поскольку они соответствуют разделителям, используемым для написания ASP-кода, встраиваемого в Web-страницы. (ASP или Active Server Pages (активные серверные страницы) — технология Microsoft, подобная PHP.)

Кроме того, ограничивать PHP-код можно с помощью HTML-тегов сценариев, например:

```
<script language="PHP">Здесь расположен PHP-код</script>
```

Корректный PHP-код

Как и в любом другом языке программирования, PHP-код должен быть корректно написан. Когда PHP-программа выполняется на Web-сервере и проходит через PHP-процессор, любые ошибки в коде приводят к тому, что пользователь видит страницу с сообщением об ошибке.

Итак, очевидно, что PHP-код должен быть корректно написан. Вместе с тем синтаксически корректное написание программы (и ее выполнение без сообщений об ошибках) не гарантирует того, что программа выдает "правильный" ответ — в коде могут присутствовать логические ошибки. (Эта тема более подробно обсуждается в главе 5.)

Кстати, код в рассматриваемой здесь простой PHP-программе работает прекрасно.

Общие маркеры в коде

В PHP используется несколько символов для указания конца строк и ограничения блоков кода. Следует отметить, что обе строки кода программы заканчиваются точкой с запятой (`;`):

```
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" .
date("Y",time());
echo $todaysdate;
```

В PHP-коде:

- выражения заканчиваются точкой с запятой (`;`);
- блоки кода заключаются в фигурные скобки (`{}`);
- комментарии в коде начинаются с символов `//` (для однострочных комментариев) или начинаются с `/*` и заканчиваются `*/` (для многострочных комментариев).

Ниже показано, как эти маркеры выглядят в блоке *псевдокода* (фиктивный код, который используется для описания обработки в PHP или иллюстрации какой-либо идеи):

```
<?php
//разместим здесь оператор echo
echo "небольшое количество псевдокода";
if ($var1 == $var2) {
    //что-то сделать
    /* это многострочный комментарий
    делаем что-то
    делаем что-то другое
    */
}
?>
```

Очень важно помнить об этих требованиях, поскольку в противном случае PHP-процессор будет генерировать и отображать сообщения о синтаксических ошибках. Вероятнее всего, запомнить эти требования будет труднее всего программистам, работавшим с Visual Basic или ASP, поскольку в этих языках точка с запятой и фигурные скобки не используются, а комментарии начинаются с апострофа (а не с `//`).

Как работают PHP-программы в Web-среде

В отличие от настольных приложений, которые запускаются на локальных системах, когда активизируются `.exe`-файлы, PHP-программы в сети выполняются, когда Web-серверу поступает запрос. Запрос пытается заставить Web-сервер получить и отправить запрашиваемый файл, но перед тем как будет сформирован ответ, PHP-процессор имеет возможность обработать PHP-код в данном файле.

В этой главе обсуждается работа написанных на PHP Web-программ. Можно запускать PHP-программы из командной строки, если имеется доступ к системе, на которой PHP установлен. Эта тема рассматривалась в главе 1.

Часто умалчивают об одном важном факте: в один момент времени может работать только один PHP-файл (поскольку только один файл может одновременно запрашиваться с сервера), а это означает, что даже если на сервере существует множество PHP-файлов, каждый из них должен функционировать как одна небольшая программа. Можно подключать к этой программе другие PHP-файлы, используя конструкции `include` или `require` (как это сделать, будет показано далее), однако на самом деле при этом код внешних файлов только копируется, а в качестве программы выполняется всего один файл. Это не является серьезным ограничением, но об этом стоит упомянуть, потому что все данные и переменные теряются каждый раз после обработки одной страницы и выполнения HTTP-запроса. Существует способ сохранения данных между запросами страницы (с помощью сеансов), но нужно четко понимать идею "один файл — одна программа".

Любую программу, которая взаимодействует с сервером, можно назвать клиентской программой, а любую программу, которая обслуживает клиентские программы, можно назвать сервером. На практике некоторые программы работают и как клиенты, и как серверы. Однако широко распространенные программы, такие как браузеры, FTP-программы и e-mail-программы, являются клиентскими и для выполнения своих функций они инициируют соединение с сервером. Серверы, к которым они обычно подключаются, являются Web-серверами, FTP-серверами и почтовыми серверами соответственно.

Клиент-серверная связь важна для PHP, так как весь PHP-код выполняется на сервере, тогда как HTML и/или JavaScript-код внутри Web-страниц передается клиенту нетронутым для обработки на стороне последнего. Технически можно отправлять браузеру для обработки нетронутый PHP-код, однако это не будет работать, так как у большинства пользователей нет возможности обработки PHP-кода в браузере. Одним из главных преимуществ, связанных с тем, что PHP-код обрабатывается на сервере, является то, что в отличие от использования JavaScript, конечный пользователь не имеет возможности просмотреть исходный код программы.

Web-соединения: Internet-протоколы и HTTP

Internet-протоколы определяют формат для всех Internet-соединений между компьютерами. Это означает, что для того чтобы один компьютер мог обмениваться данными с другим компьютером через Internet, оба компьютера должны использовать для этого обмена данными один и тот же язык. Для передачи файлов используется FTP (File Transfer Protocol — протокол передачи файлов), а для Web-коммуникаций применяется HTTP (HyperText Transfer Protocol — протокол передачи гипертекста).

В нескольких последующих разделах представлено введение в Internet-протоколы и описывается, как они способствуют обмену данными в Web-среде. Хорошее понимание происходящего между браузером и сервером является неотъемлемым для Web-программирования, потому что внутри запросов и ответов, которые пересылаются от клиента к серверу и обратно, имеется множество полезных данных, и эти данные можно перехватывать и использовать в программе.

TCP/IP

Internet обеспечивает обмен данными между многими взаимосвязанными Internet-узлами. Узлами в Internet являются все компьютеры или устройства, имеющие IP-адреса (четыре числа, разделенных точками, например, 64.71.134.49). Основным протоколом (а фактически набором сетевых протоколов), используемым для форматирования предназначенных к отправке данных, является TCP/IP (Transmission Control Protocol/Internet Protocol — протокол управления передачей/Internet-протокол). TCP/IP представляет собой просто метод описания *информационных пакетов* (индивидуально передаваемые через сеть блоки битов), так чтобы их можно было передавать по телефонным или кабельным сетям или T1-каналам от одного узла к другому, пока они не достигнут заданного пункта назначения.

Одним из преимуществ протокола TCP/IP является то, что он может очень быстро направить информацию по другому маршруту, если определенный узел или маршрут вышел из строя или работает недостаточно быстро. Когда пользователь дает браузеру команду получить какую-либо страницу, браузер, используя TCP, делит эту инструкцию на части (превращает в пакеты). TCP — транспортный протокол, обеспечивающий для данной инструкции надежный формат передачи. Данный протокол гарантирует, что все сообщение корректно разбирается и упаковывается для передачи (а также что оно корректно распаковывается и собирается в единое целое после того, как достигнет пункта назначения).

Прежде чем пакеты данных будут отправлены через сеть, необходима их адресация (пакеты должны включать в себя IP-адреса отправителя и получателя). Поэтому второй протокол, который называется протоколом передачи гипертекста (или HTTP), добавляет в них адресные метки, так чтобы TCP/IP "знал", куда следует направлять данную информацию. HTTP — протокол, используемый World Wide Web при транспортировке данных от одной машины к другой — если URL-адрес предваряется последовательностью http://, то это означает, что используется протокол HTTP. TCP/IP можно представить себе как почтовую службу, которая осуществляет маршрутизацию и передачу писем, а HTTP — штампы и адреса на письмах (данных), которые гарантируют, что письма попадают куда следует.

Сообщения, передаваемые от браузера к Web-серверу, называются *HTTP-запросами*. Получив такой запрос (фактически запрос на какую-либо Web-страницу или файл), Web-сервер проверяет свои хранилища данных в поисках соответствующей страницы. Если страница найдена, то содержащийся в ней HTML-код разделяется сервером (с помощью TCP) на пакеты, которые адресуются браузеру (с помощью HTTP) и отправляются обратно через сеть. Если Web-сервер не может найти необходимую страницу, то в ответ он генерирует страницу, содержащую сообщение об ошибке (в данном случае Error 404: Page Not Found (Ошибка 404: Невозможно найти страницу)), разделяет ее на пакеты и отправляет браузеру. Сообщения, которые отправляются Web-сервером браузеру, называются *HTTP-ответами*.

HTTP-протокол

Рассмотрим работу протокола HTTP более подробно. Отправляемый Web-серверу запрос содержит не только необходимый URL-адрес. Как часть запроса передается множество дополнительных сведений. То же верно и для ответа — кроме самой страницы сервер отправляет обратно браузеру также дополнительную информацию.

Бо́льшая часть информации, которая передается внутри HTTP-сообщения, генерируется автоматически, пользователю нет необходимости непосредственно иметь

с ней дело, поэтому и разработчик не должен беспокоиться о передаче такой информации. И все же необходимо помнить, что эта дополнительная информация передается между машинами как часть HTTP-запросов и ответов, причем PHP-сценарий позволяет непосредственно влиять на точное содержание передаваемых данных.

Независимо от вида сообщения (запрос клиента или ответ сервера) каждое HTTP-сообщение имеет один и тот же формат, состоящий из трех разделов: строка запрос/ответ, HTTP-заголовок и HTTP-тело. Содержимое этих частей зависит от того, является ли сообщение запросом или ответом.

HTTP-запрос

HTTP-запрос, который браузер отправляет Web-серверу, содержит строку запроса, заголовок и тело. Ниже приведен пример строки запроса и заголовка.

```
GET /testpage.htm HTTP/1.1
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: www.wrox.com
Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730
User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)
```

Строка запроса

Первой строкой в каждом HTTP-запросе является *строка запроса (request line)*, содержащая три блока информации:

- HTTP-команда, которая называется метод (например, `GET` или `POST`);
- путь от сервера к запрашиваемому клиентом ресурсу;
- номер версии протокола HTTP (например, HTTP 1.1).

Ниже приведен пример строки запроса:

```
GET /testpage.htm HTTP/1.1
```

Метод используется для того, чтобы указать серверу, как обрабатывать данный запрос. В следующей таблице описывается три наиболее распространенных метода.

Метод	*Описание*
GET	Запрос на информацию, расположенную по определенному URL-адресу. Большинство запросов в Internet — GET-запросы (когда пользователь нажимает на гиперссылку, генерируется GET-запрос). Информация, запрашиваемая данным запросом, может быть любой — от HTML- или PHP-страницы, до вывода JavaScript или Perl-программы и т.д. Браузер может отправлять серверу некоторые ограниченные данные в форме расширения URL-строки
HEAD	То же, что и `GET`-метод, однако `HEAD`-метод запрашивает только HTTP-заголовок без данных
POST	Указывает на то, что данные отправляются серверу как часть HTTP-тела (например, поля формы). Эти данные затем передаются программе обработки данных на Web-сервере

Протокол HTTP поддерживает большое количество других методов, включая `PUT`, `DELETE`, `TRACE`, `CONNECT` и `OPTIONS`. Как правило, эти методы менее распространены, поэтому они выходят за рамки материала данной книги. Подробнее данные методы описаны в документе RFC 2068, который можно найти на сайте `www.rfc.net`.

Заголовок HTTP-запроса

Следующей порцией отправляемой информации является HTTP-заголовок. В нем содержатся сведения о том, документы каких типов клиент принимает от сервера, тип браузера, запросившего страницу, дата и общая конфигурационная информация. Заголовок HTTP-запроса содержит информацию, которая разделяется на три различных категории:

- общий заголовок: общая информация либо о клиенте, либо о сервере;
- заголовок объекта: информация о передаваемых между клиентом и сервером данных;
- запрос: информация о клиентской конфигурации и различных типах принимаемых документов.

Ниже приведен пример заголовка запроса:

```
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: www.wrox.com
Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730
User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)
```

Очевидно, что HTTP-заголовок составляется из нескольких строк, каждая из которых содержит описание блока информации в HTTP-заголовке, а также значение этого блока.

В HTTP-заголовок может входить множество различных строк и большинство из них является необязательными, поэтому HTTP требует указывать окончание передачи заголовочной информации. Для этого используется пустая строка.

Тело HTTP-запроса

Если в строке запроса HTTP используется метод `POST`, то в HTTP-теле содержатся любые данные, которые отправляются серверу, например, данные, введенные пользователем в HTML-форму (соответствующие примеры будут приведены далее). В противном случае тело HTTP-запроса пустое, как в данном примере.

HTTP-ответ

HTTP-ответ отправляется клиентскому браузеру от сервера и состоит из строки ответа, заголовка и тела. Ниже приведен пример строки ответа и заголовка.

```
HTTP/1.1 200 OK                                          //строка состояния
Date: Fri, 31st Oct 2003, 18:14:33 GMT                   //общий заголовок
Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //заголовок ответа
Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT     //заголовок объекта
                                                         //пустая строка (конец
                                                         заголовка)
```

Строка ответа

Строка ответа содержит только два блока информации:

- номер версии HTTP;
- код HTTP-запроса, который указывает на успешное или безуспешное выполнение данного запроса.

Например, такая строка ответа

```
HTTP/1.1 200 OK
```

возвращает код HTTP-состояния 200, соответствующий сообщению `OK`, которое означает успешное выполнение запроса и то, что в ответе содержится затребованная

страница или данные от сервера. Если строка ответа содержит код HTTP-состояния 404 (уже упоминавшийся в этой главе), это значит, что Web-сервер не смог найти необходимый ресурс. Значения кодов ошибок представляют собой трехзначные числа, где первая цифра указывает класс ответа. Существует пять классов ответа, которые описаны в приведенной ниже таблице.

Класс кода	*Описание*
100–199	Информационный; указывает на то, что запрос в настоящее время обрабатывается
200–299	Отмечает, что Web-сервер успешно получил и выполнил данный запрос
300–399	Указывает на то, что запрос не был выполнен ввиду того, что необходимая информация была перемещена
400–499	Означает клиентскую ошибку (т.е. запрос был неполным, некорректным или невозможным)
500–599	Означает серверную ошибку (запрос был корректным, но сервер не смог его выполнить)

Заголовок ответа

Заголовок HTTP-ответа аналогичен рассмотренному ранее заголовку запроса. В HTTP-ответе заголовочная информация также разделяется на три типа:

- общий заголовок: содержит общую информацию либо о клиенте, либо о сервере;
- заголовок объекта: содержит информацию о данных, отправляемых между клиентом и сервером;
- ответ: содержит информацию об отправляющем данный ответ сервере, а также о возможности обработки этого ответа.

Данный заголовок состоит из множества строк, в нем также используется пустая строка, указывающая на его окончание. Ниже приведен пример заголовка; в комментариях указаны названия строк.

```
Date: Fri, 31st Oct 2003, 18:14:33 GMT              //общий заголовок
Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //заголовок ответа
Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT     //заголовок объекта
                                                    //пустая строка (конец
                                                      заголовка)
```

Назначение первой строки вполне очевидно. Во второй строке, `Server`, указывается программное обеспечение используемого Web-сервера. Так как в данном примере запрашивается файл, находящийся на данном Web-сервере, в третьей строке указывается время последней модификации запрашиваемой страницы.

Заголовок может содержать гораздо больше информации или информация может отличаться в зависимости от того, какой ресурс запрашивается. Более подробно различные типы заголовочной информации описаны в RFC 2068 (разделы 4.5, 7.1 и 7.2).

Тело ответа

Если запрос был успешным, то тело HTTP-ответа содержит HTML-код (вместе с каким-либо сценарием, выполняемым на стороне клиента), готовый для интерпретации браузером. Если запрос был безуспешным, то возвращается код ошибки.

Запуск PHP-сценариев посредством HTTP-запроса

Фактически все клиентские приложения (не только браузеры), способные отправлять HTTP-запросы Web-серверу, могут активизировать и запускать PHP-программы. На самом деле необязательно, чтобы файл отображал пользователю какой-либо вывод (т.е. разработчик не обязательно должен включать код в Web-страницу). Если Web-серверу отправляется корректно сформированный HTTP-запрос, обращающийся к файлу, содержащему PHP-код, и данный файл имеет соответствующее расширение, то PHP-программа запустится.

Web-сервер

Если Web-серверное программное обеспечение правильно настроено для работы в используемой на сервере операционной системе и для поддержки PHP, то можно ожидать, что HTTP-запросы для файлов, содержащих PHP-код, будут правильно обрабатываться, и что PHP-программы будут работать.

PHP-процессор

Язык PHP, по сути, состоит из функциональных модулей, языкового ядра (которое называется Zend Engine и к моменту написания имеет версию 2.0), а также интерфейса к Web-серверу. Этот интерфейс позволяет PHP обмениваться данными с Web-сервером. Функциональные модули снабжают PHP многими ценными возможностями, хотя Zend Engine (ядро языка) выполняет сложную работу по анализу, трансляции и выполнению поступающего кода (функции процессора Zend несколько шире, но идея заключается именно в этом). Важно отметить, что PHP-код транслируется в момент запуска PHP-программы на сервере. Это значительно упрощает работу программиста, устраняя необходимость заранее транслировать код специально для каждой машины, на которой данный код предположительно должен выполняться.

Использование переменных в PHP

Переменные используются практически в каждом языке программирования. Трудно представить себе возможность обработки данных без использования какой-либо формы переменных. Переменные являются одной из наиболее важных структур в программировании. Обычно они легко создаются и используются. В PHP переменные легко отличить, поскольку они начинаются со знака доллара ($). Поэтому если в PHP-файле ввести знак доллара с последующим именем, то получится PHP-переменная.

В PHP переменные не обязательно объявлять и инициализировать, также нет необходимости устанавливать для них типы данных, поскольку PHP является языком с так называемой слабой типизацией (loosely typed language) (подробнее данная тема рассматривается в разделе “Строгая и слабая типизация данных” далее в настоящей главе). Переменная создается при включении ее имени в выражение и одновременно присвоении ей какого-либо значения. В первом примере данной главы переменной `$todaysdate` присваивалось значение текущей даты.

```
$todaysdate = date("m",time()) . "-" . date("d",time()) . "-" .
 date("Y",time());
```

Создание переменных

Можно отметить несколько специфических для любого языка вопросов, связанных с переменными, а именно:

- именование;
- тип данных;
- область видимости.

Рассмотрим их по порядку.

Именование переменных

Переменные предназначены для хранения данных с целью их обработки. Переменными они называются потому, что значения хранящихся в них данных могут изменяться в зависимости от их обработки.

Фактически переменная состоит из двух частей: имени переменной и значения переменной. Поскольку переменные в коде используются достаточно часто, лучше всего назначать переменным такие имена, которые можно легко понять и запомнить. Как и в других языках программирования, в PHP существует несколько правил, регламентирующих именование переменных:

- имя переменной начинается со знака доллара ($);
- первым символом после знака доллара должна быть буква или символ подчеркивания;
- оставшимися символами имени могут быть буквы, цифры или символы подчеркивания без ограничений.

Имена переменных чувствительны к регистру символов (`$Variable` и `$variable` — две абсолютно разные переменные), а имена длиннее 30 символов непрактичны. В данной книге используется несколько полезных соглашений по написанию кода. Принятие какого-либо хорошего набора соглашений, скорее всего, покажется читателю стоящей идеей. Подробно соглашения по написанию сценариев рассматриваются в главе 6, однако по мере рассмотрения имен переменных все же будет представлено несколько соглашений по написанию кода.

Ниже приведен пример именованной переменной в PHP-программе:

```
$my_first_variable = 0;
```

Почему в данном случае переменной присваивается значение 0? В PHP нет особого смысла в указании имени переменной без присвоения ей тем или иным способом какого-либо значения, поскольку сам акт именования в программе создает эту переменную. Поэтому в данном случае создается переменная и ей присваивается значение 0. Естественно, можно создавать переменные и присваивать им любые значения, а не только нуль.

В некоторых языках программист ограничен, а иногда вообще не имеет возможности использовать переменную без явного предварительного ее объявления (создания). Однако PHP позволяет использовать переменные в любой точке программы, для этого нужно лишь указать их имена. И все же такое благо может оказаться обманчивым; если случайно по ошибке одно и то же имя переменной будет использовано дважды, то никакого сообщения об ошибке не будет, а в программе может возникнуть трудно обнаружимый дефект. Вместе с тем, в большинстве случаев такая возможность очень полезна и хорошо работает.

Типы данных

Другим вопросом при создании переменной является ее тип или тип хранящихся в ней данных. Что такое тип переменной? Тип переменной описывает тип хранящихся в ней данных. Читатели, которые уже работали с базами данных, вероятно, заметили, что полям в таблице базы данных часто назначается тип данных и этот тип данных позволяет различать строки, числа, даты булевы значения и т.д.

Тип данных какого-либо элемента определяет разновидность обработки, которую можно применить к элементу, а также объем необходимой для его хранения памяти. Например, если используется элемент данных строкового типа (`string`), имеющий значение `1995`, и если язык программирования не способен автоматически интерпретировать типы данных и модифицировать их в соответствие с контекстом, в котором они используются, то невозможно прибавить значение `1995` к другой строке со значением `5` и ожидать при этом, что в результате получится число `2000`. (Кстати, PHP способен автоматически интерпретировать и модифицировать типы данных; эта тема позднее рассматривается более подробно).

Вместо этого возникнет ошибка типа данных или, возможно, две строки будут объединены в одну строку и в результате получится значение `19955`, т.е. совсем не то, что ожидалось.

Строгая и слабая типизация данных

В предыдущем примере были нарушены правила использования типов данных в языке со строгой типизацией, в результате чего возникала ошибка. Однако PHP является языком со слабой типизацией данных, и поэтому он уберегает программиста от ошибок такого рода, "понимая" намерения программиста и автоматически исправляя типы данных.

Понятие *строгой типизации* языка *(strongly typed language)* означает, что язык требует явного объявления типов переменных и сгенерирует ошибку, если попытаться использовать для переменных некорректные операторы, или выдаст некорректные результаты (т.е. не те, которые ожидаются). Языки со *слабой типизацией (loosely typed)* не требуют объявления типа переменной и автоматически конвертируют типы переменных в зависимости от контекста, в котором эти переменные используются, и операций над их значениями.

PHP — слабо типизированный язык, однако он позволяет при необходимости проверять типы данных, а также устанавливать и использовать типы данных. Несмотря на то, что явно объявлять переменные и назначать им типы данных не обязательно, существует возможность определить, какой тип данных назначен переменной в процессе обработки, а также в случае необходимости привести переменные к определенному типу данных.

Типы данных в PHP

Несмотря на слабую типизацию, PHP в действительности поддерживает многие распространенные простые и структурированные типы данных. Простые типы данных содержат диапазон значений, которые можно упорядочить в одном измерении (строки, числа, булевы значения и др.), а в число структурированных типов данных включаются массивы и объекты. В PHP имеется восемь простых типов, которые описаны в следующей таблице.

Тип данных	*Описание*
Boolean (булев)	Скалярный тип; либо `True` либо `False`
Integer (целый)	Скалярный тип; целое число
Float (вещественный)	Скалярный тип; возможно, число с десятичными разрядами
String (строковый)	Скалярный тип; последовательность символов
Array (массив)	Сложный тип; упорядоченная таблица (содержащая имена и связанные с ними значения)
Object (объект)	Сложный тип; тип, который может содержать свойства и методы
Resource (ресурс)	Специальный тип; содержит ссылку на внешний ресурс, например, дескриптор открытого файла
NULL (нуль)	Специальный тип; в качестве значения может содержать только `NULL`, это означает, что переменная явно не содержит никакого значения

Программисты используют такие понятия, как скалярный, сложный и специальный, для обозначения характеристик типов данных. "Скалярный" означает, что значения такого типа данных могут быть упорядоченными по какой-либо шкале. Например, числа упорядочиваются от наименьшего к наибольшему, а символы упорядочиваются по алфавиту. "Сложный" означает, что данные состоят из множества элементов; например, массивы содержат индексы и связанные с ними значения. "Специальный" означает специальное число или значение, имеющее важный смысл для приложения, например, дескриптор файла.

Массивы описываются далее в данной главе, в последующих главах подробно рассматриваются объекты (и новые объектно-ориентированные свойства PHP).

Преобразование типов данных в PHP

В обычных обстоятельствах программисту редко приходится преобразовывать значение переменной из одного типа в другой. Однако иногда это полезно, например, когда требуется убедиться, что используется определенный тип данных, или при подготовке вывода, который будет использоваться другой программой. В PHP включены встроенные функции *приведения (casting)* (или установки) типов.

PHP-функцию `gettype()` можно использовать для определения текущего типа переменной, а функция `settype()` преобразует переменную в заданный тип. Например, в приведенном ниже коде в качестве значения переменной устанавливается целое число, затем тип меняется на строковый, при этом каждый раз распечатывается тип данных. Конкретные символы, составляющие значение, остаются одними и теми же, изменяется лишь тип данных:

```
$my_var = 1995; //$my-var содержит числовое значение
echo "Текущий тип переменной ". gettype($my_var) . "<br>";
$my_var = settype($my_var, "string");
//$my_var теперь имеет строковый тип
echo "Текущий тип переменной " . gettype($my_var);
```

PHP-функция `gettype()` возвращает строковое значение, описывающее тип переданной функции переменной (например, `string`, `integer` и т.д.). В PHP также имеются функции, которые проверяют определенный тип данных, например, `is_string`, `is_int` и др. Эти функции следует применять всегда, когда требуется проверить определенный тип, не сравнивая при этом строку, возвращенную функцией `gettype()` (например, `integer`), с предполагаемой строкой (также `integer`).

Область видимости переменной

В рассмотренных ранее примерах кода подразумевалось, что если создается переменная путем назначения ей имени, а затем данной переменной присваивается значение, то эту переменную можно использовать как угодно долго и когда угодно для передачи ее в любые функции обработки данных. В некоторой степени так оно и есть, однако имеется ряд ограничений. Эти ограничения связаны с *областью видимости (scope)* переменной.

Область видимости переменной соответствует пространству в коде, где эта переменная (а фактически ее значение) доступна для манипуляции. Как уже отмечалось, большинство переменных доступны в любой точке PHP-программы, однако в описанной формально функции (подробное обсуждение функций приведено в главе 6) переменные являются *локальными*, т.е. они распознаются и используются в пределах данной функции. Ниже приведен пример простой функции:

```
$my_data = "Внешние данные";
function send_data() {
$my_data = "Внутренние данные";
echo $my_data;
}
send_data(); //отправляет внутренние данные пользователю
echo $my_data; //отправляет внешние данные пользователю
```

Данная функция просто возвращает данные. При вызове функции (предпоследняя строка кода) пользователю отправляется строка `Внутренние данные`.

Однако если переменная `$my_data` выводится вне функции, как показано в последней строке кода, пользователю отправляется строка `Внешние данные`, поскольку переменная `$my_data` вне функции и переменная `$my_data` внутри функции представляют собой разные переменные, несмотря на то, что они имеют одинаковые имена. Область видимости переменной `$my_data` внутри функции называется локальной для данной функции. Кроме того, как только функция завершает свою работу, ее внутренняя переменная `$my_data` уничтожается, а ее значение теряется.

Ключевое слово global

Существует способ получить доступ к внешним переменным изнутри функции. Если переменная объявляется с ключевым словом `global`, то она будет доступна внутри функции, как показано в следующем примере (в главе 6 использование ключевого слова `global` описано более подробно):

```
$my_data = "Внешние данные";
function send_data() {
global $my_data;
echo $my_data;
}
send_data(); //отправляет пользователю внешние данные
echo $my_data; //также отправляет пользователю внешние данные
```

Статические переменные

Если при создании переменной внутри функции (т.е. область действия переменной локальна для данной функции и обычно теряется после завершения работы функции) используется ключевое слово `static`, то эта переменная и ее значение будет сохраняться между вызовами функции. Подобная возможность полезна в определенных ситуациях, когда желательно знать, сколько раз была вызвана данная функция. Например,

предположим, что требуется запретить вызывать какую-либо функцию для возвращения записей из базы данных более 100 раз. Добиться этого можно, задав статическую переменную, значение которой увеличивается на единицу при каждом вызове функции. Ниже приведен соответствующий пример (с небольшим количеством псевдокода):

```
function get_record() {
   static $counter = 0;
   $counter++;
   //проверить условие $counter < 100
   //если условие выполняется, то запустить код,
   //извлекающий запись из базы данных
   //если $counter = или > 100, то echo "Записей больше нет"
}
```

Каждый раз при вызове функции значение переменной `$counter` увеличивается на единицу и сохраняется до следующего вызова функции.

Определение констант

Кроме переменных в PHP можно определять другой вид контейнеров значений — *константы (constants)*. Константы, как очевидно из названия, могут быть определены (с помощью функции `define()`) в PHP-программе только однажды, а их значения невозможно изменять и они не могут быть неопределенными. Константы отличаются от переменных тем, что в начале имен у них нет символа доллара, а в остальном они именуются так же, как и переменные.

Константы могут содержать только скалярные значения, например, булевы или целые числа, числа с плавающей точкой и строки (но не массивы или объекты). К константам можно обращаться из любой точки программы, не обращая внимания на область действия, а их имена чувствительны к регистру символов. Чтобы определить константу, используется функция `define()`, которой в качестве параметров передается имя константы и ее значение, см. пример ниже:

```
define("my_constant", "1995");
//константа my-constant всегда содержит строковое значение "1995"
echo my_constant; //отправляет пользователю строку "1995"
// (обратите внимание, строку, а не целое число)
```

Операторы и выражения

Обработка данных в PHP, как и в других языках программирования, осуществляется с помощью операторов и выражений. Операторы представляют собой символы, которые указывают PHP, какую операцию необходимо выполнить, а выражения являются отдельными группами переменных и операторов, предназначенных для вычисления результатов.

PHP-операторы

В PHP значение переменной присваивается с помощью знака равенства:

```
$my_data = "Hello";
```

Знак равенства является *оператором*. Операторы используются для выполнения обработки значений переменных. В данном случае знак равенства называется *оператором присваивания*, так как с его помощью только что созданной переменной присваивается строковое значение.

В большинстве языков программирования используется много операторов; некоторые из них выполняют арифметические действия, как в простых уравнениях, другие оперируют строками или датами, а третьи осуществляют другие функции. Все они производят обработку значений переменных.

Операторы, которым нужен только один операнд, называются унарными; например, оператор ++ может присоединяться справа к имени переменной (операнду) для увеличения ее значения на единицу. Поскольку этот оператор может быть помещен как перед именем переменной, так и после него, говорят, что он допускает префиксную и постфиксную нотацию.

Операторы, которым требуется два операнда, называются бинарными; знак равенства в выражении `$my_data = "Hello"` представляет собой бинарный оператор. Данный оператор помещается между операндами, поэтому такая форма записи называется инфиксной нотацией.

В некоторых языках имеются тернарные операторы. Например, PHP позволяет использовать оператор `?`, который представляет собой сокращенное выражение `if`. Для его использования сначала пишется выражение, за которым следует знак вопроса, а затем два возможных результата, разделенных двоеточием (например, запись "выражение ? результат01 : результат02", означает "если выражение справедливо, то результатом является результат01, иначе результат02"). Поскольку оператор помещается между операндами, такая форма записи также называется инфиксной нотацией.

PHP-выражения

Выражения представляют собой любой код, который вычисляется в значение. Присвоение переменной значения само по себе является выражением, хотя часто выражения рассматриваются как уравнения (например, $a = $b + $c, где $b + $c — выражение).

Следовательно, $a = 5 — выражение, поскольку оно вычисляется в значение 5. Известно, что запись вида $a = $b + $c означает сложение $b и $c, а затем присвоение результирующего значения переменной $a. Можно создавать выражения произвольной сложности и для получения результатов применять любые операторы к любым соответствующим значениям. По сути, основная часть PHP-функциональности связана с обработкой (вычислением) выражений.

Одним из ключевых моментов при вычислении выражения, особенно сложного выражения, является приоритет операторов (как и в арифметике или математике часто имеет значение то, какая часть выражения вычисляется первой). Существует приоритет операций по умолчанию, а управлять приоритетом можно с помощью круглых скобок. Например, поскольку приоритет оператора умножения (*) выше, чем у оператора сложения (+), то выражение 2 + 2 * 12 будет равно 26 (часть 2 * 12 вычисляется первой по умолчанию, а затем к произведению прибавляется 2), тогда как результатом выражения (2 + 2) * 12 является 48 (скобки приводят к тому, что сначала выполняется сложение, а затем полученная сумма умножается на 12).

Типы операторов

В PHP доступны следующие типы операторов:

Тип	*Описание*
Арифметические	Выполняют обычные арифметические операции, такие как сложение и вычитание
Присваивания	Присвоение значения переменной
Битовые	Выполняют операции над отдельными битами целого числа
Сравнения	Сравнивают значения методом Буля (возвращается true или false)
Операторы контроля ошибок	Влияют на обработку ошибок (в PHP 5 появилось несколько новых операторов)
Выполнения	Приводят к выполнению команд, как если бы они были shell-командами
Инкрементные/ декрементные	Инкрементируют или декрементируют переменные
Логические	Булевы операторы, такие как И, ИЛИ и НЕ, которые можно использовать для включения или исключения (эта тема подробнее рассматривается в главе 4)
Строковые	Выполняют конкатенацию (объединение) строк
Для работы с массивами	Выполняют операции над массивами (например, добавление значений или разделение массива)

Справочную информацию по каждому оператору можно получить на сайте `www.php.net` (перечень операторов дан в разделе документации). По мере необходимости в данной книге рассматривается некоторая специфика используемых операторов.

Строковые операторы и функции

Существует только один строковый оператор: точка (`.`). Вместе с тем в PHP имеется множество строковых функций, которые позволяют эффективно манипулировать строками. В последующих разделах обсуждается оператор конкатенации и работа нескольких строковых функций.

Использование оператора конкатенации

Оператор конкатенации (`.`) может использоваться между строковыми значениями с целью их объединения. Ниже показан пример конкатенации в PHP-программе:

```
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

Следует отметить, что в значение `$whole_name` между значениями переменных `$first_name` и `$last_name` путем конкатенации добавляется пробел `" "` (пробел между кавычками). Пробелы перед и после каждого оператора конкатенации необязательны, но они позволяют сделать код более простым для чтения. Следующий код работает точно так же:

```
$whole_name = $first_name." ".$last_name;
```

Чтобы сделать результат более читабельным при отображении на Web-странице, в возвращаемый ответ добавляются HTML-теги (теги и , выделяющие текст жирным шрифтом):

```
echo "Имя плюс фамилия = <b>$whole_name</b>";
```

Если нет необходимости включать в HTML-код специальные символы, такие как кавычки, то можно просто вставлять HTML-теги в текст, который впоследствии будет правильно отформатирован браузером при отображении страницы. В данном случае переменная $whole_name была вставлена непосредственно в строку. Во многих языках программирования это невозможно, но PHP достаточно развитый язык и автоматически использует значение переменной, а не ее имя, когда она помещена в строку. Чтобы отобразить в строке имя переменной, необходимо экранировать знак доллара (добавив перед ним символ обратной косой черты):

```
echo "First name plus last name = <b> \$whole_name</b>";
```

Использование функции strlen()

Функция strlen() определяет длину строки. Она подсчитывает и возвращает количество всех символов в строке. В следующем примере общее число символов записывается в переменную с именем $string_length:

```
$string_length = strlen($whole_name);
```

Примечательно, что использование оператора конкатенации для объединения строки (например, "длина строки с именем") с числовым значением (например, длиной строки, содержащейся в $string_length) приводит к тому, что весь результат будет иметь строковый тип данных.

Эта функция полезна для подсчета количества символов в строке, например, при проверке данных, которые должны записываться в базу данных.

Использование функции strstr()

Функция strstr() извлекает любую часть строки, которая находится после первого вхождения определенного символа, или строку внутри другой строки. В следующем примере в значении переменной $whole_name (Иван Петров) функция strstr() ищет первое вхождение символа пробела, а затем возвращает всю оставшуюся после него часть строки. Дополнительные вхождения искомой подстроки внутри строки ничего не меняют. Если искомая подстрока не найдена внутри просматриваемой строки, то функция возвращает значение FALSE.

```
$part_after_space = strstr($whole_name, " ");
echo "Часть строки после пробела - <b>" .
$part_after_space . "</b>";
```

Использование функции strpos()

Функция strpos() используется для того, чтобы определить, существует ли в просматриваемой строке искомая подстрока. Данная функция возвращает числовое значение, представляющее собой позицию, с которой начинается искомая подстрока (если такая подстрока есть). В следующем примере поиск подстроки a в значении переменной $whole_name (Иван Петров) возвращает номер позиции — 2. Можно было бы предположить, что возвращаемым значением будет 3, так как буква a в имени стоит третьей, однако, как и во многих других языках программирования, в PHP часто используется нумерация значений, начиная с 0, а не с 1, поэтому возвращается позиция 2.

```
$letter_position = strpos($whole_name, "o");
echo "Позиция буквы &quot;а&quot;: <b>" .
$letter_position . "</b>";
```

В данном случае PHP изучает строку `Иван Петров` так, как если бы она была массивом символов (на самом деле так оно и есть в PHP), а затем для возвращения позиции любого символа использует индексы массива (`0`,`1`,`2`,`3`,`4` и т.д.). (Массивы рассматриваются далее в настоящей главе).

Использование функции chr()

Функция `chr()` возвращает строковый символ для переданного ей в качестве аргумента десятичного ASCII-значения. Таблицы ASCII-символов можно найти в Internet, и часто они весьма удобны в использовании, особенно для специальных символов. Например, ASCII-код для символа перевода строки равен 10, а для возврата каретки 13. В действительности таких символов на клавиатуре нет, и чтобы включить их в строку, необходимо воспользоваться данной функцией (`chr(10)` и `chr(13)`) и представленные ею соответствующие символы будут вставлены в строку.

Практика Работа со строками

Создадим простую PHP-программу, демонстрирующую использование операторов в выражениях с переменными. Данная программа должна продемонстрировать работу со строками. В ней используется строковый оператор, точка (`.`) и несколько встроенных строковых PHP-функций. Ниже описана последовательность действий для создания данной программы.

1. Создать файл в любом текстовом редакторе и сохранить его как `working_with_strings.php`. Файл необходимо поместить в каталог, поддерживаемый Web-сервером (если Web-сервер работает на локальной машине) или загрузить в соответствующий каталог удаленного Web-сервера (и загружать его после каждых изменений).
2. Ввести в данный файл следующий код (фрагменты PHP-кода выделены серым цветом):

```
<html>
<head>
<title>PHP5 для начинающих</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body bgcolor="#FFFFFF">
<table width="100%" border="1">
  <tr>    <td width="49%"><font face="Arial, Helvetica, sans-serif"><b>Работа
со строками</b>
</font></td>
    <td width="51%"> </td>
  </tr>
  <tr>    <td width="49%"><font face="Arial, Helvetica, sans-serif"
size="-1">Использование конкатенации - оператор точка </font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

```
</font></td>
  </tr>
  <tr>   <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-1">
Определение
       длины строки - использование <b>strlen()</b></font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$string_length = strlen($whole_name);
echo "Длина строки <b>" . $string_length . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
     <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-
1">Получение
       подстроки - использование <b>strstr()</b></font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$part_after_space = strstr($whole_name, " ");
echo "Часть строки после пробела - <b>" . $part_after_space .
 "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
     <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-
1">Определение
       позиции начала подстроки – использование <b>strpos()</b></font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$letter_position = strpos($whole_name, "a");
echo "Позиция буквы &quot;a&quot;: <b>" . $letter_position .
"</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
     <td width="49%"><font face="Arial, Helvetica, sans-serif" size="-
1">Возвращение
       символа по его ASCII-значению – использование <b>chr()</b></font></td>
    <td width="51%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$ascii_character_returned = chr(224);
echo "Символ, соответствующий ASCII-коду 224: <b>"
. $ascii_character_returned . "</b>";
?>
```

```
</font></td>
  </tr>
</table>
</body>
</html>
```

3. Сохранить файл, при необходимости выгрузить его на сервер, а затем вызывать в браузере. Результат представлен на рис. 2.2.

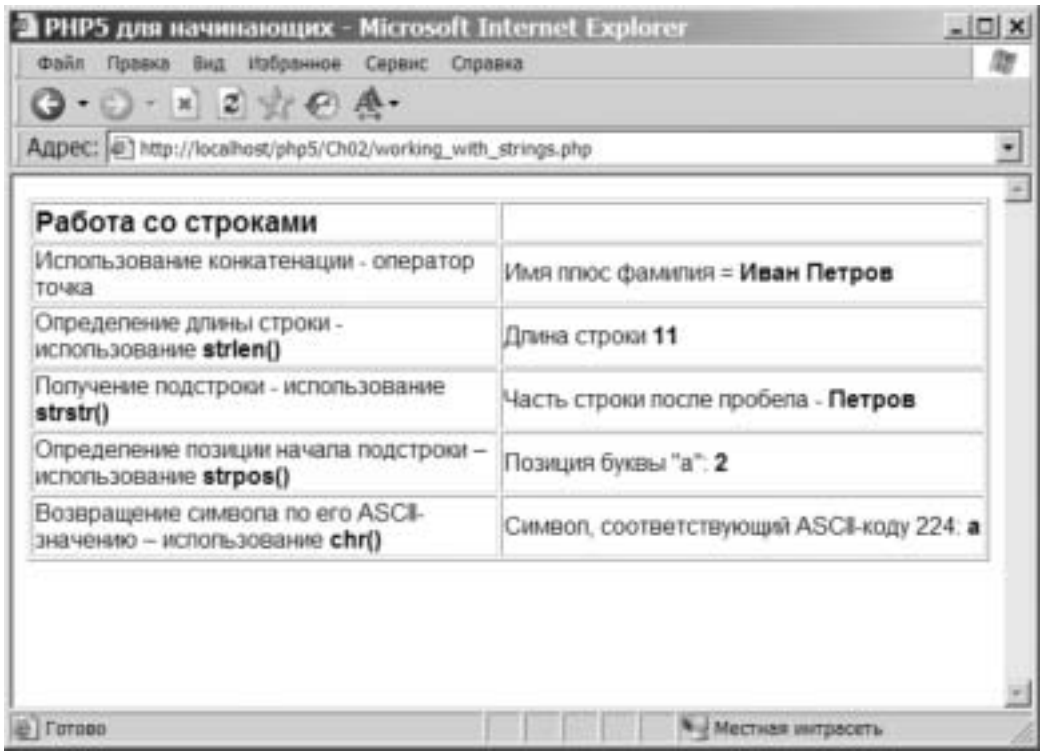

Рис. 2.2.

Как это работает

Только что созданная программа внедряется в завершенную Web-страницу. Внутри HTML-элемента `<body>` фрагменты программы содержатся в HTML-таблице исключительно с целью удобства чтения. Когда браузер запрашивает у Web-сервера файл данной страницы, PHP-код проходит синтаксический анализ и выполняется, а результат помещается в HTML-поток, возвращаемый браузеру.

Для объединения строковых значений (включая пробелы) используется оператор конкатенации (`.`):

```
<?php
$first_name = "Иван";
$last_name = "Петров";
$whole_name = $first_name . " " . $last_name;
echo "Имя плюс фамилия = <b>$whole_name</b>";
?>
```

Функция `strlen()` определяет и возвращает длину строки. В данной программе эта функция возвращает число, соответствующее количеству символов в строковом значении переменной `$whole_name`.

Функция `strstr()` определяет и возвращает любую часть строки, которая находится после первого вхождения определенного символа или подстроки в строке. В данной программе эта функция просматривает значение переменной `$whole_name` (`"Иван Петров"`) до тех пор, пока не найдет первое вхождение символа пробела, а затем возвращает оставшуюся после пробела часть строки.

Функция `strpos()` определяет и возвращает число, соответствующее позиции символа в строке. В данном случае поиск буквы `a` в значении переменной `$whole_name` возвращает позицию 2. (Напомним, что значения начинаются с `0`, поэтому третьей по счету позицией является позиция номер 2.)

Функция `chr()` возвращает строковый символ, соответствующий десятичному ASCII-значению, переданному функции в качестве аргумента. В данной программе возвращается строковый символ для ASCII-значения 224.

Арифметические операции в PHP

В PHP арифметические операторы ("плюс", "минус" и т.д.) работают так же, как от них следует ожидать, позволяя создавать выражения как простые уравнения. Например, в выражении `$c = $a + $b` складываются значения переменных `$a` и `$b`, а затем результат присваивается переменной `$c`. (Оператор присваивания `=` полностью отличается от операторов сравнения `==` и `===`, которые рассматриваются в главе 4.)

Кроме того, как и в обычных уравнениях, имеет значение приоритет операторов; повлиять на него можно с помощью круглых скобок, см. пример ниже:

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
<b>$total</b><br>";
echo "Двадцать умножить на тридцать и разделить на (три плюс два) равно
<b>$total2</b>";
?>
```

Если запустить программу, то различие, вызванное использованием скобок, будет очевидным, поскольку первый оператор `echo` выведет значение 202, а второй — 120.

Специальные операторы присваивания

Знак равенства можно комбинировать с другими операторами. Это позволяет создавать специальные операторы присваивания, которые упрощают написание некоторых выражений. Специальные операторы присваивания (такие как `+=`, `-=` и др.) позволяют использовать стенографический метод для выполнения обычных арифметических операторов; при этом не требуется несколько раз писать имя одной и той же переменной. Например, можно использовать следующий код:

```
$first_number += $second_number;
```

вместо такого кода:

```
$first_number = $first_number + $second_number;
```

Такой подход также применим для других видов операторов. Например, оператор конкатенации можно комбинировать со знаком равенства (`.=`), так чтобы к текущему значению левой стороны выражения присоединялось значение правой стороны, см. пример ниже:

```
$a = "Начало и ";
$b = "конец предложения.";
$a .= $b //в результате значением $a является строка "Начало и конец предложения."
```

Основные арифметические операторы, строковый и битовые операторы поддерживают такое комбинирование. Более подробная информация о комбинируемых операторах приведена на официальном Web-сайте PHP.

Использование инкрементных и декрементных операторов

Очень часто возникает необходимость многократно прибавлять к числу или вычитать из него одно и то же число. Для решения подобных задач существуют специальные операторы: инкрементные и декрементные. Они записываются соответственно как два знака "плюс" или два знака "минус" перед или после имени переменной, например:

```
$a = ++$a; //прибавляет единицу к $a, а затем возвращает результат
$a = $a++; //возвращает $a, а затем прибавляет к $a единицу
$b = --$b; //вычитает единицу из $b, а затем возвращает результат
$b = $b--; //возвращает $b, а затем вычитает из $b единицу.
```

Расположение данных операторов имеет большое значение. Оператор, предшествующий имени переменной, приводит к тому, что операция (сложение или вычитание единицы) выполняется перед тем, как возвращается значение переменной; оператор, следующий после имени переменной, возвращает текущее значение переменной, а после этого выполняет операцию.

> *Следует отметить, что инкрементные и декрементные операторы можно (ограниченно) использовать и для символов. Например, можно "прибавить" единицу к символу B и возвращаемым значением будет символ C. Однако вычитать единицу (декрементировать) из символьных значений нельзя.*

Использование математических PHP-функций

В PHP встроены многие распространенные математические функции. Некоторые из них требуют указания аргументов, другие не принимают аргументы, а для третьих аргументы являются необязательными. Например, можно использовать функцию `floor()` для округления числа в меньшую сторону независимо от величины его дробной части. Однако данной функции необходимо передать аргумент. В противном случае, что она будет округлять? Аргумент представляет собой первоначальное значение, которое необходимо округлить. Например, чтобы округлить число 100.0, необходимо использовать следующий код:

```
$a = 100.01;
$floor_a = floor($a);
```

С другой стороны такие функции, как `pi()` и `rand()`, не требуют аргументов. Функция `pi()` возвращает число π до 14 знаков после запятой (по умолчанию 14 знаков, но фактическая точность зависит от параметра директивы `precision` в файле `php.ini`). Функция `rand` генерирует (псевдо) случайное число в диапазоне от `1` до `RAND_MAX` (максимальное число, разное для разных операционных систем), если ей не переданы аргументы, ограничивающие диапазон, из которого функция может выбрать случайное число.

Практика Работа с числами

Следующая программа демонстрирует работу с числами. В ней используются как уже знакомые читателю, так и новые операции, а также некоторые доступные в PHP операторы и встроенные функции. В данном случае PHP-код также внедряется в HTML-код с тем, чтобы можно было вывести результаты на Web-странице.

1. Откройте HTML-редактор и введите в него следующий код. (Хотя ввод всего кода будет хорошим упражнением, вместо этого можно загрузить файл `working_with_numbers.php` с Web-сайта данной книги).

```
<html>
<head>
<title>PHP5 для начинающих</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>

<body bgcolor="#FFFFFF">
<table width="100%" border="1">
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif"><b>Работа с
      числами</b></font></td>
    <td width="43%"> </td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Использование оператора сложения (+)</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$second_number = 30;
$total = $first_number + $second_number;
echo " Двадцать плюс тридцать равно <b>$total</b>";
?>
```

```
 </font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Использование инкрементного оператора (++)</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$first_number = ++$first_number;
echo " Двадцать, инкрементированное на единицу, равно <b>$first_number</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Использование операторов умножения и деления (* и /)</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
       <b>$total</b><br>";
echo "Двадцать умножить на тридцать, разделить на (три плюс два) равно
       <b>$total2</b>";
?>
```

```
</font></td>
  </tr>
```

```
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Специальные операторы присваивания — использование += и *=
      </font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
$second_number = 30;
$total = $first_number += $second_number;
$total2 = $first_number *= $second_number;
echo " Двадцать += тридцать равно <b>$total</b><br>";
echo " Предыдущий результат *= тридцать равно <b>$total2</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Получение абсолютного значения числа — использование функции abs()
      </font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = -2.7;
echo " Абсолютное значение -2,7 равно <b>" . abs($first_number) . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Преобразование двоичного числа в десятичное — использование функции
      bindec()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$binary_number = 10101111;
$decimal_number = bindec($binary_number);
echo "Десятичным эквивалентом двоичного числа 10101111 является число
      <b>$decimal_number</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Округление чисел в большую и меньшую сторону — использование функций
      ceil() и floor()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 2.4;
echo "Число 2,4, округленное в большую сторону, равно <b>"
      . ceil($first_number)
      . "</b>, в меньшую сторону — <b>" . floor($first_number)
      . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Определение максимального или минимального значения — использование
      функций max() и min()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$max_value = max(2,3,4);
$min_value = min(2,3,4);
echo "Максимальным числом из 2,3,4 является <b>"
    . $max_value . "</b>,
    а минимальным — <b>" . $min_value . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Получение числа пи — использование функции pi()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
echo " Число пи равно <b>" . pi() . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Получение случайного числа — использование функции rand()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
echo " Случайное число: <b>" . rand() . "</b>";
?>
```

```
</font></td>
  </tr>
  <tr>
    <td width="57%"><font face="Arial, Helvetica, sans-serif" size="-1">
      Извлечение квадратного корня — использование функции sqrt()</font></td>
    <td width="43%"><font face="Arial, Helvetica, sans-serif" size="-1">
```

```
<?php
$first_number = 20;
echo " Квадратный корень из двадцати равен <b>"
       . sqrt($first_number) . "</b>";
?>
```

```
</font></td>
  </tr>
</table>
</body>
</html>
```

2. Сохраните файл как `working_with_numbers.php`, в случае необходимости выгрузите его на сервер и отобразите в браузере. Результат показан на рис. 2.3.

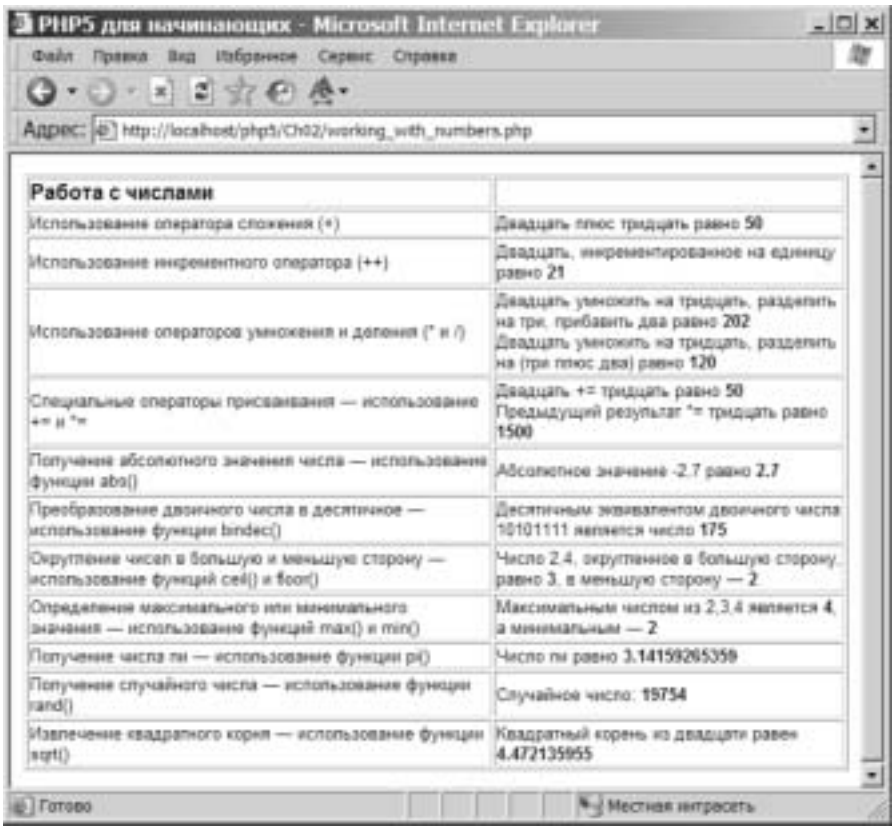

Рис. 2.3.

Как это работает

В данной программе используется тот же формат отображения, что и в предыдущем примере. В программе показаны некоторые простые вычисления с помощью операторов, простейшее использование встроенных функций, а также результаты работы нескольких встроенных функций, не принимающих аргументов (например, `pi()`).

В коде задавались некоторые значения, а затем производилась их обработка при различном расположении круглых скобок. Это иллюстрирует приоритетность операторов.

```
<?php
$first_number = 20;
$second_number = 30;
$third_number = 3;
$fourth_number = 2;
$total = $first_number * $second_number / $third_number + $fourth_number;
$total2 = $first_number * $second_number / ($third_number + $fourth_number);
echo "Двадцать умножить на тридцать, разделить на три, прибавить два равно
      <b>$total</b><br>";
echo "Двадцать умножить на тридцать, разделить на (три плюс два) равно
      <b>$total2</b>";
?>
```

Затем использовались инкрементный и декрементный операторы для прибавления и вычитания единицы из численного значения, содержащегося в переменных `$a` и `$b`.

```
$a = ++$a; //прибавляет единицу к $a, а затем возвращает результат
$a = $a++; //возвращает $a, а затем прибавляет к $a единицу
$b = --$b; //вычитает единицу из $b, а затем возвращает результат
$b = $b--; //возвращает $b, а затем вычитает из $b единицу
```

Пример использования функций `pi()` и `rand()` показывает, как заставить PHP генерировать значение для числа пи или случайные значения. Все что для этого требуется — просто написать имя функции без параметров в скобках (и, конечно, использовать знак равенства для присвоения значения переменной).

Массивы

Массивы представляют собой переменные типа `array` (`array` — в данном случае тип данных), однако это весьма специфические и эффективные переменные, которые заслуживают отдельного раздела в книге.

Строго говоря, массивы представляют собой списки, состоящие из *ключей (keys)* (индексов) и *значений (values)*, которые содержатся в каждом *элементе (element)* массива. Элементами являются контейнеры значений в массиве. Элементы можно представить как отдельные переменные, состоящие из пар имя/значение.

И хотя в литературе часто отмечается, что массивы могут быть весьма сложными и трудными для понимания, в них достаточно просто разобраться: массивами являются переменные с множеством контейнеров значений и несколькими способами доступа к определенному значению. Они в некотором смысле похожи на реляционные мини-базы данных, которые являются динамическими, поскольку существуют в памяти до тех пор, пока выполняется текущая программа (если между запросами страницы они не сохраняются в сеансах или в реальных базах данных). Имена массивов подобны именам таблиц в базе данных, а элемент массива, содержащий другой массив, подобен связанной таблице в базе данных. Подобная аналогия не претендует на абсолютную точность, но может оказаться весьма полезной, раскрывающей механизм создания или доступа к массивам и их значениям.

Индексы массивов

Функция (а фактически конструкция языка) `array()` используется для создания массивов и принимает в качестве аргументов значения, которые необходимо поместить в создаваемый массив. Доступ к элементу массива можно получить по его *индексному номеру (или индексу)*. Индексный номер подобен небольшому адресу, по которому можно получить доступ к определенной переменной внутри массива. Так как имена всех переменных в массиве начинаются с имени массива, каждая такая переменная должна иметь собственный уникальный номер. Индексы массивов начинаются с нуля (0). Например, в следующей строке кода в переменную `$my_array` записывается массив, содержащий четыре элемента, пронумерованные 0, 1, 2 и 3:

```
$my_array = array ("кошка", "собака", "лошадь", "золотая рыбка");
```

Переменной `$my_array` присваивается результат выполнения функции `array()`. Если теперь вызвать для данной переменной функцию `is_array()`, то результат будет истинным (функция вернет `true`), указывая на то, что переменная `$my_var` действительно структурирована как массив.

Чтобы получить доступ к значениям только что созданного массива, можно использовать следующий код:

```
$zero_element = $my_array[0];
$one_element = $my_array[1];
$two_element = $my_array[2];
$three_element = $my_array[3];
```

Использование строк в качестве индексов массивов

Новый элемент массива получает следующий номер (начиная с нуля) в последовательности. Однако массивы можно использовать в разных ситуациях, поэтому часто полезно давать элементам имена, а не последовательные числа. Например, в следующем фрагменте кода создается массив, в котором каждый элемент имеет в качестве имени строку, а затем нескольким переменным присваиваются значения именованных строк:

```
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик", "hamster" =>
"Пушок");
$my_dog = $my_named_array["dog"];
$my_cat = $my_named_array["cat"];
$my_hamster = $my_named_array["hamster"];
echo "Мою собаку зовут $my_dog, кота - $my_cat,
 а хомячка $my_hamster";
```

Возможность доступа к любому значению по его имени важна, потому что не требуется знать последовательность значений или фактический индексный номер — необходимо знать только имя, заданное элементу. При использовании строк в качестве индексов массивов лучше всего использовать кавычки вокруг имен элементов. Несмотря на то, что опускать кавычки просто и удобно, официальная документация предостерегает от подобной практики, предвосхищая то время, когда кавычки станут обязательными, а их пропуск будет нарушать работу кода.

При необходимости можно использовать индексные номера вместо имен, так как PHP-массивы всегда содержат индексы наряду с любыми присвоенными именами, поэтому следующий код будет работать точно так же, как и в предыдущем примере:

```
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик", "hamster" =>
"Пушок");
$my_dog = $my_named_array[0];
$my_cat = $my_named_array[1];
$my_hamster = $my_named_array[2];
echo "Мою собаку зовут $my_dog, кота - $my_cat,
 а хомячка $my_hamster";
```

Инициализация массивов

Инициализировать (т.е. создавать первоначальные значения) массивы можно по-разному. Например, можно использовать функцию `array()` или завершать имя переменной квадратными скобками (`[]`). Написание имени переменной с пустыми квадратными скобками указывает PHP на то, что необходимо создать массив и начать инкрементировать индекс с нуля, если данный элемент является в массиве первым, см. код ниже.

```
$my_array[] = "первый элемент";
```

Если требуется назначить имя новому элементу, то следует вставить это имя в квадратные скобки:

```
$my_array["first"] = "первый элемент";
```

Если после этого снова присвоить элементу `$my_array["first"]` какое-либо значение, то PHP не будет создавать новых элементов, а просто перепишет исходное значение. Однако если снова использовать запись `$my_array[]`, то PHP создаст

в массиве новый элемент и назначит ему индексный номер (следующий номер в последовательности).

Интересно отметить: если программа того требует, то в одном массиве могут содержаться различные значения, каждое из которых может иметь свой тип данных (индексами для всех элементов могут быть исключительно строки или целые числа). Это означает, что массивы можно использовать для хранения данных почти так же, как записи в таблице базы данных.

Работа с массивами

Иногда после создания и инициализации массива (особенно со значениями, взятыми из записи в таблице базы данных) могут возникать некоторые трудности при определении того, какими могут быть эти значения, а, следовательно, в таком случае будет труднее отлаживать код. Эту задачу решает функция `print_r()`, которая позволяет распечатывать все значения элементов массива наряду с их именами и индексами. Продолжая данный пример, создадим простую HTML-страницу, содержащую следующий PHP-код:

```
<?php
$my_named_array = array("dog" => "Пират", "cat" => "Мурзик",
 "hamster" => "Пушок");
print_r($my_named_array);
?>
```

Результат работы этого сценария показан на рис. 2.4.

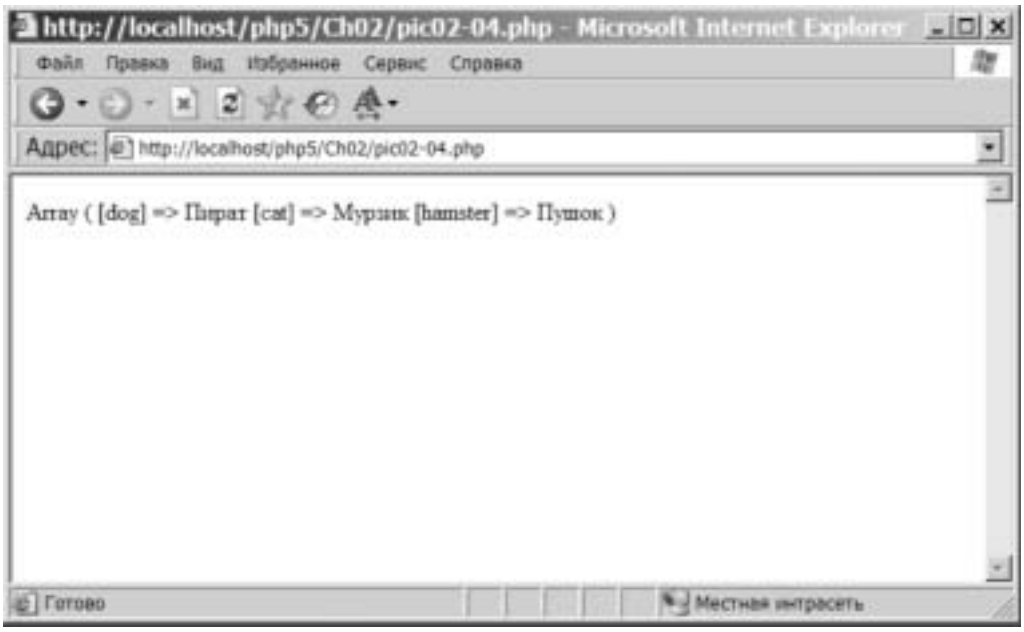

Рис. 2.4.

Использование функции `print_r` весьма полезно, когда требуется просмотреть все содержимое массива. (В главе 4 рассказывается о специальных циклах, которые также значительно облегчают доступ ко всем значениям массива.)

Массивы в PHP, как и во многих других языках программирования, можно смело назвать "рабочими лошадками"; в PHP имеется множество встроенных функций, специально предназначенных для работы с массивами. Далее рассматриваются некоторые из наиболее широко используемых функций. (Более серьезная работа с данными функциями описана в главах 3 и 4.) Многие из функций для работы с массивами аналогичны функциям, которые можно применять к базам данных.

Часто возникают ситуации, когда требуется выяснить, сколько элементов имеется в массиве. Для подсчета количества элементов можно использовать функцию `count()` (при необходимости ее можно также использовать для подсчета элементов в любой переменной):

```
$number_of_elements = count($my_array);
```

Функция `array_count_values()` решает другую задачу — она возвращает (в виде массива) частоту появления значений в массиве, который передан ей в качестве аргумента. Имена элементов в массиве `$returned_array` — значения в массиве `$argument_array`, а значения элементов `$returned_array` — числа, показывающие, сколько раз данные значения встречаются в `$argument_array`. Работу функции проще понять, если рассмотреть следующий код:

```
$argument_array = array("dog", "dog", "cat", "cat", "hamster");
$returned_array = array_count_values($argument_array);
print_r($returned_array);
```

В результате выполнения этого фрагмента выводится следующая информация:

```
Array
(
    [dog] => 2
    [cat] => 2
    [hamster] => 1
)
```

Функция `array_flip()` оказывается полезной, когда требуется поменять местами значения и имена ключей. Например, когда имеется список имен людей в качестве имен элементов, а значением каждого элемента является SSN (Social Security number — номер социального страхования), и необходимо "перевернуть" массив так, чтобы доступ к имени человека можно было получить посредством его SSN-номера (так как SSN должен быть уникален, а имена могут дублироваться). Добиться этого можно с помощью следующего кода:

```
$my_people_array = array("John" => "555-66-7777", John => "444-55-3333");
$my_ssn_array = array_flip($my_people_array);
```

Затем массив `$my_ssn_array` можно использовать для поиска людей, даже если имя John повторяется дважды.

Сортировка массивов с помощью функций sort() и asort()

Часто возникает необходимость отсортировать элементы массива (например, требуется создать список имен в алфавитном порядке). Это можно сделать с помощью функции `sort()`. Для того чтобы отсортировать элементы массива и сохранить при этом связь значений и индексов, используется функция `asort()`. Функция `sort()` сортирует значения и последовательно присваивает им индексы, а функция `asort()` сохраняет связь индексов и значений элементов. Код может выглядеть так:

```
$my_unsorted_array = array("Jim", "Bob", "Mary");
$my_sorted_array = sort($my_unsorted_array);
$my_sorted_array_with_unchanged_indexes = asort($my_unsorted_array);
```

Резюме

В данной главе рассматривались основные этапы написания простой PHP-программы, внедрение PHP-кода в HTML (с использованием корректного расширения имен файлов), запуск программы с Web-сервера, а также принципы написания корректного PHP-кода (использование точки с запятой, разделителей, обратной косой черты и т.д.).

В данной главе описано, как PHP-программы интерпретируются процессором Zend Engine, и как Web-сервер "узнает", когда отправлять код данному процессору. PHP-код выполняется на сервере, следовательно, он не передается конечному пользователю в формате исходного кода.

Как и во многих языках программирования, в PHP широко применяются переменные, но необходимость объявлять переменные перед использованием отсутствует. И хотя PHP-переменные всегда имеют какой-либо тип данных, явно объявлять этот тип не требуется, PHP автоматически конвертирует типы данных в зависимости от контекста, в котором используются данные.

В главе описывались некоторые встроенные PHP-функции для работы со строками и числами, а также с массивами; на примерах страниц было показано, как код работает на практике. В данной главе рассматривается лишь создание начальных, простейших PHP-программ. Глава 3 проливает свет на то, как сконструированы "настоящие" PHP-программы. Кроме того, в следующей главе подробно рассматривается работа протокола HTTP и HTML-форм.

Упражнения

1. Создайте PHP-программу, которая трансформирует первое предложение во второе, и выводит и результат. Оба предложения представлены ниже:

   А. Теперь пора всем хорошим людям прийти на помощь стране;

   Б. Пора теперь стране прийти на помощь всем хорошим людям.

2. Напишите PHP-программу, которая создает два массива чисел и прибавляет значения одного массива к соответствующим (по индексу) значениям другого массива. Массивы должны содержать следующие значения:

   А. 2, 4, 6, 8, 10;

   Б. 3, 5, 7, 9, 11.