O'REILLY®

Изучаем PHP 7

РУКОВОДСТВО ПО СОЗДАНИЮ ИНТЕРАКТИВНЫХ ВЕБ-САЙТОВ

Дэвид Скляр

Изучаем PHP 7

РУКОВОДСТВО ПО СОЗДАНИЮ
ИНТЕРАКТИВНЫХ ВЕБ-САЙТОВ

Learning PHP 7

A GENTLE INTRODUCTION TO
THE WEB'S MOST POPULAR LANGUAGE

David Sklar

Beijing • Boston • Farnham • Sebastopol • Tokyo

Изучаем PHP 7

РУКОВОДСТВО ПО СОЗДАНИЮ ИНТЕРАКТИВНЫХ ВЕБ-САЙТОВ

Дэвид Скляр

Москва • Санкт-Петербург • Киев
2017

ББК 32.973.26-018.2.75
С43
УДК 681.3.07

Компьютерное издательство “Диалектика”
Зав. редакцией *С. Н. Тригуб*
Перевод с английского и редакция *И.В. Берштейна*

По общим вопросам обращайтесь в издательство “Диалектика” по адресу:
info@dialektika.com, http://www.dialektika.com

Скляр, Дэвид.
С43 Изучаем PHP 7: руководство по созданию интерактивных веб-сайтов. : Пер. с англ. — СпБ. : ООО “Альфа-книга”, 2017. — 464 с. : ил. — Парал. тит. англ.

ISBN 978-5-9908462-3-4 (рус.)

ББК 32.973.26-018.2.75

Книга отпечатана согласно договору с ООО "ПРИМСНАБ".

Научно-популярное издание
Дэвид Скляр

Изучаем PHP 7
Руководство по созданию интерактивных веб-сайтов

Литературный редактор *И.А. Попова*
Верстка *О.В. Мишутина*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 30.11.2016 Формат 70×100/16.
Гарнитура Times.
Усл. печ. л. 29,0. Уч.-изд. л. 22,3.
Тираж 400 экз. Заказ № 9149.

Отпечатано в АО «Первая Образцовая типография»
Филиал «Чеховский Печатный Двор»
142300, Московская область, г. Чехов, ул. Полиграфистов, д.1

ООО “И.Д. Вильямс” 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-9908462-3-4 (рус.)

ISBN 978-1-491-93357-2 (англ.)

Оглавление

Содержание

Предисловие

Статические веб-сайты скучны. Намного интереснее *динамические* веб-сайты, поскольку их содержимое изменяется. Громадная статическая HTML-страница, где перечисляются наименования, изображения, описания и цены всей обширной продукции, выставляемой компанией на продажу, неудобна в употреблении и бесконечно долго загружается. А динамическая веб-страница с каталогом товаров, где можно искать и отбирать товары по цене и категории, оказывается более удобной, оперативной и скорее приводящей к успешному завершению сделки по продаже.

Язык программирования PHP упрощает создание динамических веб-сайтов. Он позволяет решать самые разные задачи создания интерактивного содержимого, будь то составление каталога товаров, фотоальбома, календаря событий и даже организация блога. Прочитав эту книгу, и вы будете способны справиться с задачей построения динамического веб-сайта.

Кому адресована эта книга

Она будет полезной разным категориям читателей, включая следующие.

- Любители, которым требуется создать интерактивный веб-сайт для себя, своей семьи или общественной организации.
- Разработчики или проектировщики, которым требуется подключаемый модуль или расширение для распространенного программного обеспечения, написанного на PHP (например, Drupal, WordPress или MediaWiki).
- Дизайнеры веб-страниц, которым требуется более тесная связь с коллегами-разработчиками.
- Программирующие на JavaScript, которым требуется писать серверные программы, дополняющие их клиентский код.
- Программирующие на Perl, Python или Ruby, которым требуется быстро освоить PHP.
- Все, кому требуется простое и понятное введение в один из самых распространенных языков программирования, предназначенных для создания динамических веб-сайтов.

Постепенное изучение PHP и доступный синтаксис делает этот язык идеальным “преддверием” для создателей веб-сайтов без специальной технической подготовки. Эта книга адресована тем, кто проявляет интерес к веб-разработке, обладает достаточной сообразительностью, но не имеет необходимой технической подготовки, а также программирующим на других языках и стремящимся овладеть PHP.

Если программирование для вас совершенно внове и вы собираетесь построить свой первый интерактивный веб-сайт, значит, вы выбрали нужную книгу. В ее начальных главах дается постепенное

введение в синтаксис языка PHP и основные понятия программирования на компьютере применительно к PHP. Поэтому изучайте материал этой книги с самого начала, постепенно продвигаясь вперед.

Помимо элементарной компьютерной грамотности (т.е. умения обращаться с файлами и просматривать веб-содержимое в Интернете), от читателей требуется хотя бы беглое знакомство с HTML. Для этого совсем не обязательно быть знатоком HTML, но необходимо разбираться в таких дескрипторах HTML, размечающих элементарные веб-страницы, как, например, `<html>`, `<head>`, `<body>`, `<p>`, `<a>` и `<br>`. Если же вы не знакомы с HTML, рекомендуется прочитать книгу Эда Титтеля и Криса Минника *HTML5 и CSS3 для чайников* (ISBN 978-5-8459-2035-5, пер. с англ., изд-во "Диалектика", 2016) г.).

Содержание книги

Эта книга составлена таким образом, чтобы постепенно прорабатывать представленный в ней материал по порядку следования глав. Материал каждой главы основывается главным образом на материале предыдущих глав. Главы 2-13 завершаются упражнениями для закрепления приобретенных знаний и проверки правильности усвоения материала.

В главе 1 дается общее представление о языке PHP и поясняются особенности его взаимодействия с веб-браузером и веб-сервером. В ней также демонстрируются и разъясняются примеры программ, чтобы дать ясное представление о том, как они выглядят и что делают. Эту главу особенно полезно прочитать тем, кто только начинает осваивать программирование и построение динамических веб-сайтов.

В последующих пяти главах рассматриваются основы языка PHP. Прежде чем писать литературный шедевр, необходимо хотя бы изучить грамматику и приобрести некоторый словарный запас. Именно этому и посвящены главы 2-6. Прочитав их, вы в достаточной степени освоите грамматику PHP и накопите необходимый словарный запас для написания коротких программ, если не шедевров.

В главе 2 поясняется, как обрабатывать разные типы данных, в том числе фрагменты текста и числа. Это очень важно, поскольку веб-страницы, формируемые в программах на PHP, представляют собой крупные фрагменты текста.

В главе 3 описываются команды PHP, которыми можно пользоваться в программах для принятия решений. Такие решения составляют саму суть веб-сайта, который считается *динамическим*. Команды, рассматриваемые в главе 3, служат, например, для отображения только тех товаров в каталоге, которые относятся к диапазону цен, введенному пользователем в веб-форме.

В главе 4 представлены *массивы*, образующие совокупности отдельных чисел или фрагментов текста. Массивы применяются во многих операциях, часто выполняемых в программах на PHP (например, при интерпретации параметров передаваемой на обработку веб-формы или анализе информации, извлекаемой из базы данных).

При написании более сложных программ нередко приходится повторно решать одни и те же задачи. Повторно использовать фрагменты кода помогают *функции*, обсуждаемые в главе 5.

В главе 6 поясняется, каким образом данные и логика объединяются в *объекты*, которые образуют связки кода, помогающие структурировать программы. Кроме того, объекты позволяют интегрировать существующие дополнения PHP и библиотеки в прикладной код.

Последующие пять глав посвящены решению основных задач построения динамического веб-сайта. К их числу относится взаимодействие с пользователями, сохранение информации и взаимодействие с другими веб-сайтами.

В главе 7 подробно рассматриваются вопросы обработки веб-форм, которые представляют собой основное средство взаимодействие пользователей с веб-сайтом.

В главе 8 обсуждаются базы данных. В базе данных хранится информация, отображаемая на веб-сайте (например, каталог товаров или календарь событий). В этой главе поясняется, как организуется взаимодействие программ на PHP с базой данных. Овладев приемами, рассматриваемыми

в главе 8, можно реализовать на своем сайте выполнение таких операций, как, например, отображение секретных сведений только для привилегированных посетителей или уведомление о количестве новых посланий, созданных на доске сообщений с момента последнего входа на сайт.

Помимо обращения к базе данных, у вас может возникнуть потребность обрабатывать данные, хранящиеся в файлах. В главе 9 поясняется, как читать и записывать данные в файлы из программ на PHP.

В главе 10 подробно рассматриваются способы отслеживания пользователей. К их числу относится применение cookie-файлов для хранения не только временных данных, но и сведений о пользователях, зарегистрированных по учетным записям, а также данных отслеживания сеансов (например, сведений о корзине с закупленными товарами).

Последняя в данной части глава 11 посвящена вопросам взаимодействия программ на PHP с другими веб-сайтами и веб-службами. В частности, в программах на PHP можно организовать извлечение содержимого других веб-страниц или применение прикладных программных интерфейсов API веб-служб, а также обслуживание ответов этих интерфейсов на запросы других клиентов.

В трех последующих главах обсуждаются вопросы, помогающие совершенствоваться в программировании на PHP, вместо новых языковых средств, которые можно внедрить в свои программы.

В главе 12 поясняются особенности отладки программ на PHP. К их числу относится обнаружение и устранение ошибок в программах.

В главе 13 показывается, как писать тесты для проверки различных частей программы. Такие тесты позволяют убедиться, что программа делает именно то, что от нее требуется.

А в главе 14 речь пойдет от некоторых особенностях программотехники, не характерных для PHP. Тем не менее о них следует знать, чтобы работать над проектами с другими разработчиками.

Последняя часть данной книги посвящена краткому исследованию ряда типичных задач и вопросов веб-разработки. И хотя они не носят такой же фундаментальный характер, как материал, посвященный основной структуре PHP или сохранению информации, тем не менее, их придется так или иначе решать, приобретя некоторый опыт программирования на PHP. В главах заключительной части рассматриваются самые основные подходы к решению подобных задач и вопросов веб-разработки.

В главе 15 демонстрируются эффективные и обширные возможности языка PHP для обработки дат и времени. В главе 16 обсуждаются вопросы *управления пакетами*, предоставляющими необыкновенно простой способ внедрения в свой код полезных библиотек, написанных другими. В главе 17 поясняется, как посылать сообщения электронной почтой из программы на PHP. В главе 18 исследуются три распространенных каркаса веб-приложений на PHP, с которых можно начать свой проект, исключив немалую долю общего стереотипного кода. В главе 19 поясняется, как пользоваться средствами PHP из командной строки, а не из веб-сервера, что может быть удобно для написания утилит или коротких тестовых программ. И, наконец, в главе 20 рассматриваются некоторые методики для успешного написания программ на PHP, способных безошибочно обрабатывать текст на разных языках и в разных наборах символов.

В двух приложениях к данной книге представлен дополнительный материал. Чтобы выполнить программу на PHP, нужно иметь копию интерпретатора PHP, установленного на вашем компьютере (или учетную запись, предоставляемую поставщиком услуг веб-хостинга, поддерживающим PHP). Материал приложения А поможет вам установить интерпретатор PHP на платформе Windows, Mac OS X или Linux. А в приложении Б приведены ответы на вопросы всех упражнений, предлагаемых в данной книге. Только не заглядывайте в ответы, не попробовав выполнить эти упражнения самостоятельно!

На кого *не* рассчитана эта книга

Эта книга имеет ограниченный объем, и поэтому, к сожалению, не может охватить все, что известно о языке PHP. Ее основное назначение — дать введение в PHP и основы программирования на компьютере.

Если у вас уже имеется опыт программирования на PHP и вы хотели бы ознакомиться с нововведениями в версии PHP 7, рекомендуется приобрести отличную книгу *Upgrading to PHP 7* Дэйви Шафика (Davey Shafik; издательство O'Reilly). А на сайте Бруно Скворца (Bruno Skvorc) по адресу `https://www.sitepoint.com/learn-php-7-find-out-whats-new-and-m` можно найти подборку ссылок на ресурсы, посвященные нововведениям в версии PHP 7.

Другие ресурсы

Аннотированное руководство по PHP (`http://ua2.php.net/manual/ru/index.php`) служит отличным первоисточником для изучения обширной библиотеки функций PHP. В обильных комментариях, внесенных пользователями, дается немало полезных советов и образцов написания кода. Кроме того, в этом руководстве упоминаются многие списки рассылки, посвященные установке, программированию, расширению языка PHP и самым разным вопросам его применения. Узнать об этих списках рассылки и подписаться на них можно по адресу `http://www.php.net/mailing-lists.php`. Полезно также исследовать архив PHP Presentation System, доступный по адресу `http://talks.php.net`. Он содержит целый ряд презентаций PHP, сделанных на различных конференциях. Сайт PHP The Right Way (`http://www.phptherightway.com` или `http://getjump.me/ru-php-the-right-way/` для русскоязычных пользователей) также служит замечательным ресурсом для изучения PHP, особенно теми, у кого уже имеется опыт программирования на других языках.

Проработав материал данной книги, можете воспользоваться следующей литературой для дальнейшего изучения PHP.

- *Programming PHP*, Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre (издательство O'Reilly). Это более подробное, техническое руководство по написанию программ на PHP. В нем рассматриваются вопросы безопасности, XML-раз- метки документов и формирования графики.
- *PHP Cookbook*, David Sklar and Adam Trachtenberg (издательство O'Reilly; в русском переводе книга вышла под названием *PHP. Рецепты программирования в издательстве* "Питер"в 2015 г.). Это полное описание типичных задач программирования на PHP и их решений.
- *PHP: объекты, шаблоны и методики программирования, 4-е издание*, Мэтт Зандстра (ISBN 978-5-8459-1922-9, пер. с англ. ИД "Вильямс 2015). Эта книга посвящена не синтаксису PHP и конкретным задачам, решаемым на этом языке, а, напротив, помогает написанию программ на PHP в согласованном, высококачественном стиле и овладению нормами надлежащей практики программирования на PHP. В ней рассматриваются такие вопросы, как развертывание, тестирование и профилирование программ на PHP.

Перечисленная ниже литература будет полезной для изучения баз данных, в том числе SQL и MySQL.

- *Learning PHP, MySQL, JavaScript, and CSS*, Robin Nixon (издательство O'Reilly). В этой книге поясняется, как гармонично сочетать средства PHP, MySQL и JavaScript для построения надежного динамичного веб-сайта.
- *SQL: полное руководство, 3-е издание*, Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель (ISBN 978-5-8459-1654-9, пер. с англ., ИД "Вильямс 2014). В этой книге рассматриваются самые основы, которые требуется знать для составления запросов SQL, включая различные диалекты этого языка, применяемые в базах данных Microsoft SQL Server, MySQL, Oracle и PostgreSQL.
- *MySQL Cookbook*, Paul DuBois (издательство O'Reilly). Это полное собрание типичных задач, решаемых в базе данных MySQL.
- *MySQL Reference Manual*. Это справочное руководство служит основным источником информации по базе данных MySQL и диалекту языка SQL.

Условные обозначения, принятые в книге

Ниже перечислены условные обозначения, принятые в отношении исходного кода и текста данной книги.

Условные обозначения в исходном коде

Примеры исходного кода, приведенные в данной книге, были составлены по версии PHP 7.0.0. Они были протестированы в PHP 7.0.5 — самой последней версии PHP, имевшейся на момент издания данной книги. Если же в примерах используются языковые средства, внедренные в PHP после выпуска версии 5.4.0, то в тексте книги обычно указывается конкретная версия PHP, в которой было внедрено применяемое языковое средство.

Условные обозначения в тексте книги

В тексте данной книги приняты следующие условные обозначения.

- *Курсив*. Служит для выделения новых терминов, когда они впервые появляются в тексте.
- `Моноширинный шрифт`. Служит для выделения исходного кода примеров или фрагментов программ на PHP, имен классов, методов, переменных, файлов и URL.
- **`Моноширинный полужирный шрифт`**. Служит для выделения команд и прочих данных, вводимых из командной строки, а также имен классов, методов, переменных в исходном коде.
- ***`Моноширинный полужирный наклонный шрифт`***. Служит для выделения текста, который должен быть заменен значениями, подставляемыми пользователем или определяемыми из контекста.

Обозначает рекомендацию, предположение или общее примечание

Обозначает предупреждение или предостережение.

Пользование примерами кода

Набирать вручную исходный код примеров, приведенных в этой книге, полезно начинающим программировать. Но если вы устанете набирать исходный код всех этих примеров, то можете загрузить его по адресу `https://github.com/oreillymedia/Learning_PHP`.

Эта книга служит справочным пособием, помогающим читателю решать стоящие перед ним задачи разработки прикладных программ. В общем, примерами кода, приводимыми в книге, можно пользоваться в своих программах и документации. Для этого не нужно спрашивать разрешения у автора или издателя. Так, для употребления в прикладной программе нескольких фрагментов кода

из примеров из данной книги специальное разрешение не требуется. Но для продажи или распространения в иных целях на CD-ROM фрагментов кода из примеров к данной книге обязательно требуется разрешение издательства O'Reilly. Для цитирования текста и примеров кода из данной книги в ответах на вопросы специальное разрешение не требуется. Но для внедрения значительной части примеров кода из данной книги в документацию на собственную продукцию обязательно требуется разрешение издательства O'Reilly.

Ссылки на эту книгу как на первоисточник желательны, но не обязательны. В ссылке обычно указываются название книги, автор, издатель и ISBN. Например, "*Learning PHP* by David Sklar (O'Reilly). Copyright 2016 David Sklar, 978-149-193357-2".

Если читатель считает, что употребление им примеров кода из этой книги выходит за рамки правомерного использования или упомянутых выше разрешений, он может связаться с издательством O'Reilly по адресу `permissions@oreilly.com`.

Посвящение

Посвящается M и C с пожеланием никогда не переставать учиться.

Благодарности

Эта книга стала результатом нелегкого труда многих людей. В связи с этим особая благодарность выражается:

- Многим программистам, тестировщикам, составителям документации, правщикам программных ошибок и прочим лицам, которые посвятили свое время, талант и преданность делу создания из PHP первоклассной платформы для современной веб-разработки. Без них было бы не о чем вообще писать.
- Моим прилежным рецензентам: Томасу Дэвиду Бейкеру (Thomas David Baker) и Филу Маккласки (Phil McCluskey). Они выявили немало погрешностей в рукописи книги, прояснив сбивающие с толку описания, чтобы эта книга стала лучше, чем если бы она была без их помощи.
- Моему прилежному редактору Элли Макдональд (Ally MacDonald). Отвечая лишь за одну часть, из которых состоит работа над книгой, Элли позаботилась о своевременности всех остальных частей книги!

Благодарю скорее судьбу, чем мудрость, за то, что могу и далее пренебрегать синтаксическими ошибками, которые прилежно исправляет мой литературный редактор Сюзанна.

Об авторе

Дэвид Скляр работает штатным разработчиком программного обеспечения в компании Google. До этого он работал в компании Ning, занимаясь построением платформ, прикладных программных интерфейсов API, а также сред выполнения кода PHP в “песочницах”. Дэвид проживает в Нью-Йорке, где он предпочитает питаться на ходу. Подробнее о нем можно узнать из его личного блога по адресу `www.sklar.com/blog`.

Изображение на обложке

На обложке данной книги изображен орел. Орлы относятся к виду птиц-хищников, к которому принадлежат также соколы и ястребы. Имеются два вида птиц-хищников: хватающие и убивающие, а также хватающие и удерживающие. У первого вида имеется клюв такой формы, чтобы терзать и разрывать жертву, а также округлые, заостренные когти на коротких пальцах лап, чтобы хватать и душить жертву. И у второго вида имеется клюв такой формы, чтобы терзать и кусать жертву, а также длинные пальцы лап, чтобы крепко держать жертву. Орлы относятся к хватающим и убивающим птицам-хищникам. Морские орлы (или орланы) имеют пальцы лап, специально приспособленные для захвата гладкой жертвы вроде рыбы. Превосходное зрение всех орлов позволяет им обнаруживать жертву, паря высоко в воздухе или сидя высоко на насесте. Наметив жертву, орел устремляется вниз, хватает ее и поднимается вверх одним изящным движением. Нередко орлы поедают свои жертвы в полете, разрывая их на части и отбрасывая несъедобные части, чтобы облегчить свой груз. Как и большинство птиц-хищников, орлы питаются ослабевшими или раненными животными.

Насчитывается более 50 видов орлов, обитающих по всему миру, кроме Новой Зеландии и Антарктиды. Все виды орлов строят гнезда, называемые орлиными, высоко над землей, на деревьях или скалистых выступах. Пара орлов годами пользуется одним и тем же гнездом, обкладывая его зелеными листьями и травой, мехом, дерном и прочими мягкими материалами. Орлы расширяют свои гнезда каждый год. Самое крупное из обнаруженных орлиных гнезд оказалось размерами около 6×3 м. Охота, повышенное применение пестицидов и сокращение естественной среды обитания наряду с оскудением источников добывания пищи поставило под угрозу существование многих видов орлов.

Многие виды животных, изображенных на обложках книг издательства O'Reilly, находятся под угрозой исчезновения, хотя все они важны для нашего мира. Подробнее о том, как помочь спасению этих видов животных, можно узнать по адресу `animals.oreilly.com`.

Изображение орла для обложки данной книги взято с гравюры XIX века, хранящейся в Дуврском художественном архиве (Dover Pictorial Archive).

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 веб-сайт 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 электронные адреса:

E-mail: `info@williamspublishing.com`
WWW: `http://www.williamspublishing.com`

Наши почтовые адреса:

в России: 195027, Санкт-Петербург, ул. Магнитогорская, д. 30, ящик 116
в Украине: 03150, Киев, а/я 152

ГЛАВА 1

Краткое введение в PHP

Имеется немало веских оснований для написания компьютерных программ на языке PHP. Вам, вероятно, требуется изучить язык PHP, чтобы создать небольшой сайт с рядом интерактивных элементов. А может быть, PHP применяется там, где вы работаете, и вам нужно войти в курс дела. В этой главе рассматривается контекст, необходимый для того, чтобы вписать язык PHP в общую конструкцию веб-сайта, а также поясняются его возможности и предназначение. Кроме того, в ней дается первое представление о языке PHP, который демонстрируется в действии.

Место PHP в мире Интернета

Язык программирования PHP предназначен, главным образом, для построения веб-сайтов. Вместо того чтобы выполнять программу, написанную на PHP, в однопользовательском режиме на настольном компьютере, ее обычно запускают на вебсервере, чтобы сделать доступной для многих людей, пользующихся веб-браузерами на своих компьютерах. В этом разделе поясняется, каким образом язык PHP вписывается во взаимодействие веб-браузера и веб-сервера.

Сев за свой компьютер и открыв веб-страницу в окне браузера (например, Safari или Firefox), вы, по существу, вызываете диалог между компьютерами через Интернет. Этот диалог, приводящий к появлению веб-страницы на экране вашего компьютера, наглядно показан на рис. 1.1.

Отдельные этапы рассматриваемого здесь диалога без участия PHP пронумерованы на рис. 1.1 и поясняются ниже.

1. Вы вводите **`www.example.com/catalog.html`** в строке веб-адреса, находящейся в верхней части окна браузера.

2. Браузер посылает сообщение через Интернет на компьютер по адресу `www.example.com`, запрашивая страницу `/catalog.html`.

3. HTTP-сервер Apache, работающий на компьютере по адресу `www.example.com`, получает сообщение и читает файл `catalog.html` из своего накопителя на жестких дисках.

4. Веб-сервер посылает содержимое файла обратно на ваш компьютер через Интернет в качестве ответа на запрос браузера.

5. Браузер отображает страницу на экране вашего компьютера, следуя инструкциям, указанным в дескрипторах HTML-разметки данной страницы.

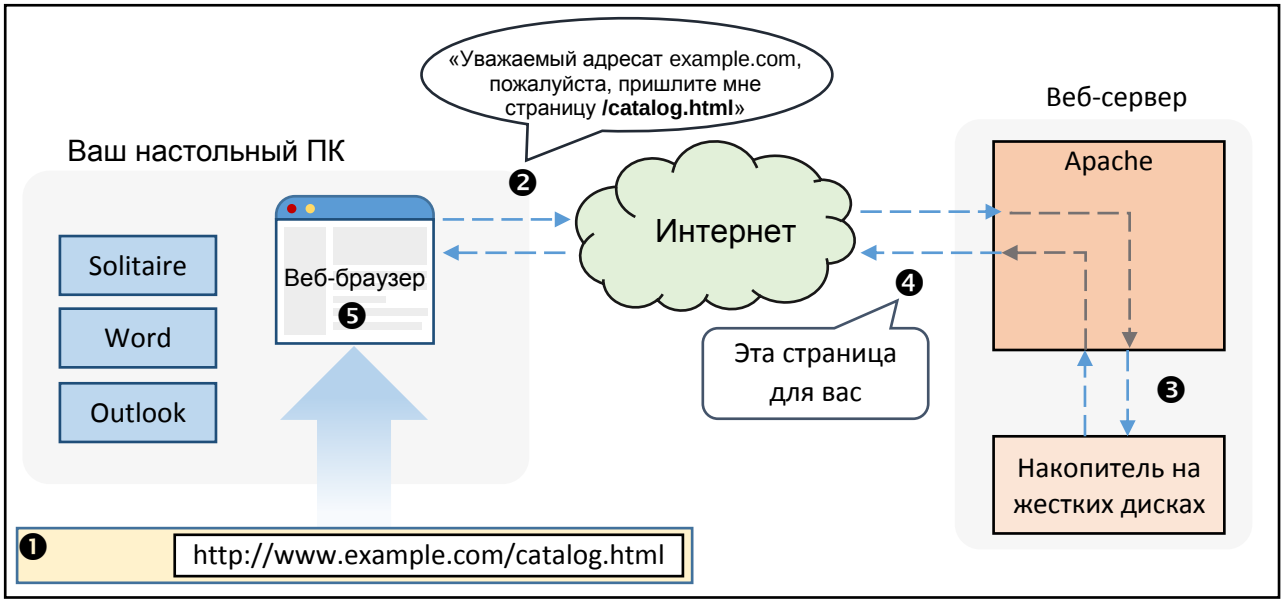


Рис. 1.1: Связь клиента с сервером без участия PHP

Всякий раз, когда браузер запрашивает страницу по адресу `http://www.example.com/catalog.html`, веб-сервер посылает обратно содержимое того же самого файла `catalog.html`. Ответ веб-сервера изменится лишь в том случае, если кто-нибудь отредактирует запрашиваемый файл на сервере.

Но если задействовать PHP, то сервер сможет сделать нечто большее со своей стороны диалога. На рис. 1.2 наглядно показано, что произойдет, когда веб-браузер запросит страницу, сгенерированную средствами PHP.

Отдельные этапы рассматриваемого здесь диалога с участием PHP пронумерованы на рис. 1.2 и поясняются ниже.

1. Вы вводите **`www.example.com/catalog/yak.php`** в строке веб-адреса, находящейся в верхней части окна браузера.

2. Браузер посылает сообщение через Интернет на компьютер по адресу `www.example.com`, запрашивая страницу `/catalog/yak.php`.

3. HTTP-сервер Apache, работающий на компьютере по адресу `www.example.com`, получает сообщение и обращается к интерпретатору PHP, также работающему на компьютере по адресу `www.example.com`, со следующим вопросом: “Как выглядит страница `/catalog/yak.php`?”

4. Интерпретатор PHP читает файл `yak.php` из накопителя на жестких дисках.

5. Интерпретатор PHP выполняет команды из файла `yak.php`, возможно, обмениваясь данными с системой управления базой данных, например MySQL.

6. Интерпретатор PHP принимает результат выполнения программы из файла `yak.php` и посылает его обратно на HTTP-сервер Apache в качестве ответа на вопрос “Как выглядит страница `/catalog/yak.php`?”

7. HTTP-сервер Apache посылает содержимое страницы, полученное обратно от интерпретатора PHP, на ваш компьютер через Интернет в ответ на запрос браузера.

8. Этот браузер отображает страницу на экране вашего компьютера, следуя инструкциям, указанным в дескрипторах HTML-разметки данной страницы.

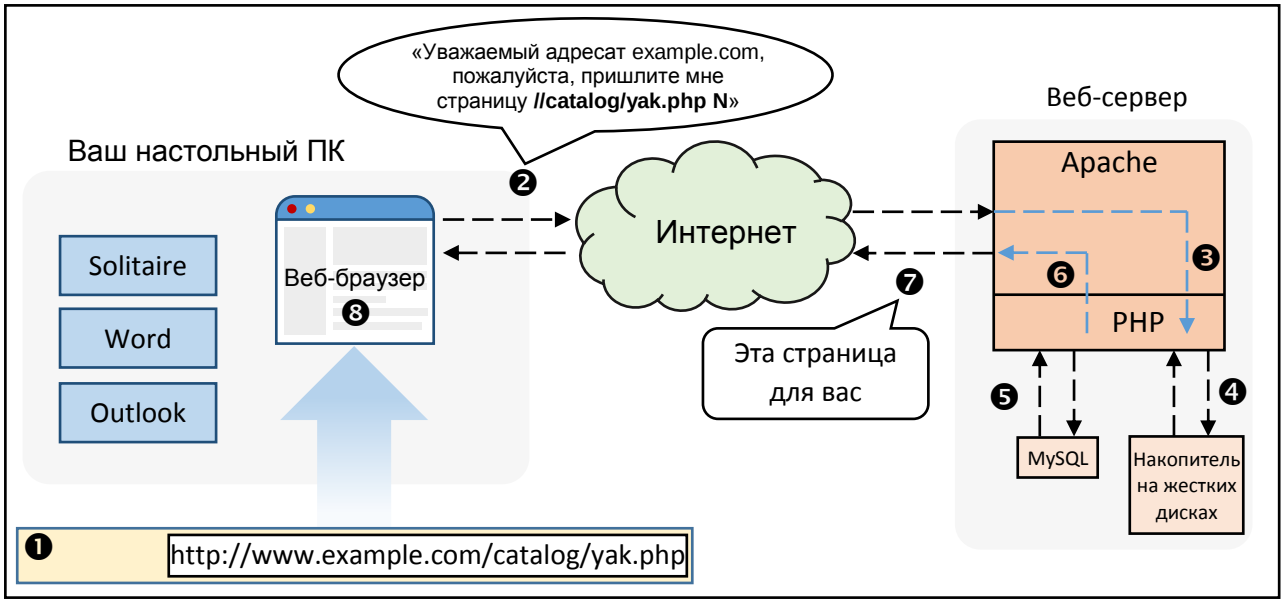


Рис. 1.2: Связь клиента с сервером при участии PHP

PHP — это язык программирования. Программы на PHP представляют собой написанные на этом языке инструкции, которые читаются на компьютере веб-сервера, и на их основании решается, что делать дальше. *Интерпретатор PHP* следует инструкциям. Программисты нередко подразумевают под термином PHP интерпретируемый язык программирования. А в данной книге под термином PHP подразумевается средство, выполняющее команды из написанных на PHP программ и формирующее веб-страницы.

Если язык программирования PHP можно сравнить с естественным английским языком, то интерпретатор PHP — с англоязычным человеком. В английском языке определены различные слова и их сочетания. Прочитав или услышав их, англоязычный человек переводит их в различные смысловые значения, которые побуждают его к соответствующим действиям, например, почувствовать стеснение, пойти в магазин и купить молоко или надеть штаны. А программы, написанные на языке программирования PHP, побуждают интерпретатор PHP выполнять такие действия, как обращение к базе данных, формирование персонализированной веб-страницы или показ изображения.

Данная книга посвящена особенностям написания подобных программ, т.е. тому, что происходит на этапе 5, приведенном на рис. 1.2. Хотя в приложении А подробно рассматривается конфигурирование и установка интерпретатора PHP на веб-сервере.

Язык PHP называется *серверным* потому, что программы на нем выполняются на веб-сервере, как показано на рис. 1.2. А языки вроде JavaScript называются *клиентскими* потому, что они встроены в веб-браузер, вынуждая его выполнять такие действия, как открытие нового окна, при выполнении на настольном ПК. Как только веб-сервер отправит сформированную веб-страницу клиенту (см. этап 7 на рис. 1.2), участие PHP на этом завершается. Если страница содержит какой-нибудь сценарий JavaScript, то он выполняется на стороне клиента, хотя он совершенно не связан с программой на PHP, сформировавшей страницу.

Простая HTML-страница похожа на форму письма “Просим извинить за таракан, обнаруженный в вашем супе”, которое вы можете получить после гневной жалобы на обслуживание в самолетах авиалинии, кишащих насекомыми. Когда ваше письмо поступит в штаб-квартиру авиалинии, перегруженный обязанностями секретарь из отдела обслуживания клиентов извлечет ответное письмо “по поводу таракана в супе” из канцелярского шкафа, сделает его копию и отравит его адресату по почте. На каждый такой запрос будет получен точно такой же ответ.

А динамическая страница, формируемая средствами PHP, напротив, похожа на письмо, отправленное по почте товарищу на другом конце света. На такой странице можно разместить все, что угодно: каракули, диаграммы, стихи, трогательные истории о том, как ваш необыкновенно смышленый младенец разбрызгивает морковное пюре по всей кухне. Содержимое письма привязано к

конкретному человеку, которому оно адресуется. Но как только вы заклеите письмо в конверт и отправите его по почте, вы больше не сможете его изменить. Получив письмо на другом конце света, ваш товарищ прочитает его в том виде, в каком вы уже не в состоянии его изменить.

А теперь представьте, что вы пишете письмо товарищу, занимающемуся прикладным искусством. Наряду с каракулями и трогательными историями вы приводите в письме инструкции вроде "Вырезать небольшое изображение лягушки вверху страницы и наложить его на мелкое изображение кролика внизу страницы" и "Прочитать последний абзац на странице прежде всех остальных абзацев". Читая письмо, ваш товарищ выполнит также действия по указанным в нем инструкциям. Эти действия аналогичны сценарию JavaScript на веб-странице. Они задаются при написании письма, после чего их уже нельзя изменить. Но когда получатель письма последует указанным в нем инструкциям, само письмо может измениться. Аналогично веб-браузер подчиняется любым командам JavaScript на странице, открывая окна, изменяя пункты в меню формы или обновляя страницу по новому URL.

Достоинства PHP

Язык PHP может привлечь вас тем, что он свободно доступен, прост в изучении, или же только потому, что ваше начальство поручило вам приступить к работе над проектом PHP на следующей неделе. А поскольку вы собираетесь воспользоваться PHP, то должны хотя бы немного знать о его достоинствах. И когда вас когда-нибудь спросят, что же такого особенного в PHP, воспользуйтесь материалом этого раздела в качестве основания для своего ответа.

Язык PHP свободно доступен (бесплатно)

Платить за пользование PHP не нужно никому. Запускаете ли вы интерпретатор PHP на изношенном ПК десятилетней давности в подвале своего дома или в помещении, заполненном дорогостоящими серверами масштаба предприятия, вам не придется платить за лицензию, техническую поддержку, сопровождение, обновление и любые другие виды поборов.

PHP уже входит в комплекты поставки Mac OS X и большинства версий Linux. Если PHP отсутствует в комплекте поставки вашей операционной системы или вы пользуетесь другой операционной системой (например, Windows), загрузите PHP по адресу `http://www.php.net`. Подробные инструкции по установке PHP приведены в приложении A.

Язык PHP свободно доступен (как речь)

Как проект с открытым исходным кодом, PHP дает возможность изучить свое внутреннее устройство всякому желающему. Если PHP не делает то, что вы от него ожидаете, или вам просто интересно разобраться в механизме действия какого- нибудь языкового средства, можете свободно заглянуть во внутреннее устройство интерпретатора PHP, написанного на языке программирования C. И даже если у вас нет для этого достаточной технической подготовки, вы можете обратиться за помощью к какому-нибудь специалисту, который проведет подобное исследование для вас. Ведь многие владельцы автомашин не умеют их ремонтировать и обращаются к автомеханикам, которые могут быстро устранить неполадку, заглянув под капот.

PHP является межплатформенным языком

PHP можно применять на компьютере веб-сервера, работающем под управлением ОС Windows, Mac OS X, Linux и многих версий Unix. И даже если сменить операционную систему на веб-сервере, то не придется менять ни одну из программ на PHP. Для этого достаточно скопировать их, например, из сервера под Windows на сервер под Unix, и при этом они будут по-прежнему работать исправно.

Несмотря на то что Apache считается самым распространенным веб-сервером, применяемым вместе с PHP, имеется также возможность воспользоваться nginx, Microsoft Internet Information Server (IIS) или любым другим веб-сервером, поддерживающим стандарт CGI. Кроме того, PHP нормально взаимодействует с огромным числом баз данных, включая MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite, Redis и MongoDB.

Если все упомянутые выше названия программных продуктов вам непонятны, не отчаивайтесь! Все они означают лишь одно: какой бы операционной системой вы ни пользовались, PHP вполне может работать в ней, нормально взаимодействуя с любой базой данных, которая уже применяется в ней.

PHP является широко употребляемым языком

PHP применяется на более чем 200 млн различных веб-сайтов: от бесконечного числа личных начальных страниц до гигантских порталов вроде Facebook, Wikipedia, Tumblr, Slack и Yahoo. Имеется немало литературы, периодических изданий и вебсайтов, посвященных обучению PHP. Короче говоря, если вы — пользователем PHP, то вы далеко не один.

Сложности PHP скрыты внутри

Средствами PHP можно создавать эффективные движки сайтов электронных магазинов, способные обслуживать миллионы клиентов, а также небольшие сайты, на которых автоматически поддерживаются ссылки на изменяющийся список статей или сообщений для печати. Если вы пользуетесь PHP для выполнения простых проектов, этому не помешают обязанности, имеющие отношение только к крупной системе. Любые дополнительные средства вроде кеширования, применения специальных библиотек или динамического формирования изображений доступны вам по мере надобности в них. А если они вам не нужны, то о них и не стоит особенно беспокоиться, уделив внимание только основам обработки и отображения данных, вводимых пользователем.

Язык PHP предназначен для веб-разработки

В отличие от других языков программирования, язык PHP с самого начала предназначался для формирования веб-страниц. Это означает, что типичные задачи веб-разработки вроде организации доступа к формам, передаваемым на обработку, и обращения к базе данных зачастую легче решать средствами PHP. Язык PHP обладает возможностями форматировать HTML-документы, манипулировать датами, временем и cookie-файлами, т.е. решать задачи, которые можно реализовать на других языках только с помощью дополнительных библиотек.

PHP в действии

Итак, готовы ли вы сделать свою первую попытку воспользоваться PHP? В этом разделе представлен ряд листингов программ на PHP и поясняется их назначение. Если вам непонятно, что происходит в каждом из листингов, не отчаивайтесь! Пояснению особенностей программирования на PHP посвящена остальная часть данной книги. Проанализируйте эти листинги, чтобы получить некоторое представление о программах на PHP и принципе их действия, не пытаясь вдаваться в подробности.

Когда программа запускается на выполнение, интерпретатор PHP уделяет внимание только тем частям программы, которые находятся между начальными и конечными дескрипторами PHP. А все, что находится за пределами этих дескрипторов, выводится на экран без всяких видоизменений. Благодаря этому упрощается встраивание небольших фрагментов кода PHP на страницах, которые содержат в основном HTML- разметку. Интерпретатор PHP выполняет команды, находящиеся между начальным `<?php` и конечным `?>` дескрипторами PHP. Как правило, PHP-страницы находятся в

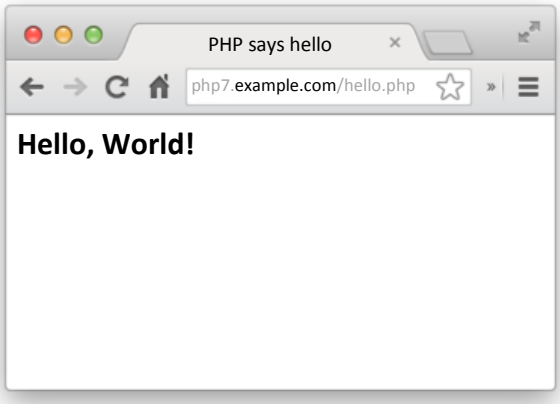


Рис. 1.3: Приветствие на PHP

файлах с расширением **.php**. В примере 1.1 демонстрируется разметка страницы с одной командой PHP.

Пример 1.1. Здравствуй, мир!

```
<html>
<head><title>PHP says hello</title></head>
<body>
<b>
<?php
print "Hello, World!";
?>
</b>
</body>
</html>
```

Результат выполнения кода из примера 1.1 приведен ниже. А в окне веб-браузера он выглядит так, как показано на рис. 1.3.

```
<html>
<head><title>PHP says hello</title></head>
<body>
<b>
Hello, World!</b>
</body>
</html>
```

Но вывод на экран сообщения, которое вообще не изменяется, не особенно впечатляет. Ведь сообщение `"Hello, World!"` (Здравствуй, мир!) можно было бы с тем же успехом включить в состав простой HTML-страницы. Больше пользы может принести вывод динамических данных, т.е. такой информации, которая изменяется. Одним из самых распространенных источников информации для программ на PHP служит сам пользователь. В частности, браузер отображает заполняемую форму, пользователь вводит информацию в этой форме и щелкает на кнопке для передачи формы на обработку, а браузер посылает ее на сервер, где она в конечном итоге передается интерпретатору PHP, доступному в программе обработки данной формы.

В примере 1.2 демонстрируется HTML-форма без кода PHP. Она состоит лишь из текстового поля user и кнопки Submit (Передать) и передается на обработку программе из файла `sayhello.php`,

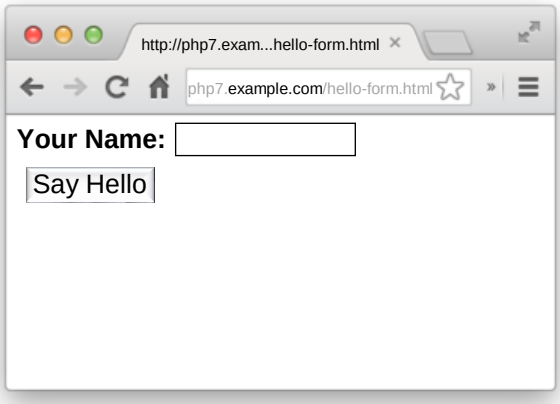


Рис. 1.4: Вывод формы на экран

указанного в атрибуте `action` дескриптора `<form>`.

Пример 1.2. HTML-форма для передачи введенных данных на обработку

```
<form method="POST" action="sayhello.php">
Your Name: <input type="text" name="user" />
<br/>
<button type="submit"">Say Hello</button>
</form>
```

HTML-форма из примера 1.2 воспроизводится в окне браузера так, как показано на рис. 1.4.

В примере 1.3 демонстрируется программа из файла `sayhello.php`, выводящая на экран приветствие всякому, кто введет свое имя в текстовом поле данной формы.

Пример 1.3. Динамические данные

```
<?php
print "Hello, ";
// вывести на экран значение параметра 'user'
// из переданной на обработку формы
print $_POST['user'];
print "!";
?>
```

Если вы введете **`Ellen`** в текстовом поле и передадите ее на обработку, то программа из примера 1.3 выведет на экран сообщение `"Hello, Ellen!"`. На рис. 1.5 показано, каким образом это сообщение отображается в окне веб-браузера.

Переменная `$_POST` содержит значения параметров формы, передаваемой на обработку. В терминологии программирования переменная называется так потому, что содержащееся в ней значение можно изменять. В частности, переменная типа *массива* содержит не одно, а целый ряд значений. В общем, массивы обсуждаются в главе 4, а более подробно — в главе 7.

В данном примере строка кода, начинающаяся со знаков `//`, называется *строкой комментария*. Строки комментариев присутствуют в исходном коде и предназначены для пояснения принципа действия исходного кода тем, кто его читает. Они игнорируются интерпретатором PHP. Комментарии удобны для аннотирования программ сведениями о том, как они работают. Более подробно комментарии обсуждаются далее, в разделе “Комментарии”.

Средствами PHP можно также вывести на экран HTML-форму, позволяющую передать значение параметра `user`, как демонстрируется в примере 1.4.

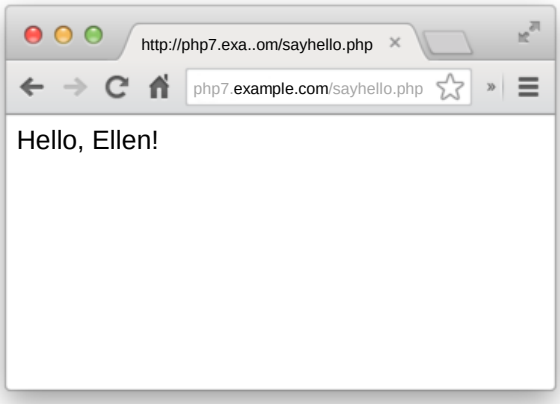


Рис. 1.5: Вывод параметра формы на экран

Пример 1.4. Вывод формы на экран

```
<?php
print <<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Your Name: <input type="text" name="user" />
<br/>
<button type="submit">Say Hello</button>
</form>
_HTML_;
?>
```

В примере 1.4 применяется строковый синтаксис, называемый *встраиваемым документом* (дословно "здесь документ" — *here document*). Все, что находится между идентификаторами `<<<_HTML_` и `_HTML_`, передается команде `print` для отображения на экране. Как и в примере 1.3, `$_SERVER[PHP_SELF]` обозначает переменную. Это специальная переменная, которая содержит URL текущей страницы (без указания сетевого протокола или имени хоста, т.е. веб-узла). Так, если URL страницы из примера 1.4 равен `http://www.example.com/users/enter.php`, то переменная `$_SERVER[PHP_SELF]` содержит значение `/users/enter.php`.

Указав переменную `$_SERVER[PHP_SELF]` в качестве обработчика формы, можно разместить на той же самой странице код для вывода формы на экран и выполнения какого-нибудь действия над данными, передаваемыми в этой форме на обработку. Пример 1.5 сочетает в себе примеры 1.3 и 1.4 для формирования одной страницы, где отображается форма и выводится приветствие, когда эта форма передается на обработку.

Пример 1.5. Вывод формы или приветствия на экран

```
<?php
// вывести на экран приветствие, если форма
// передана на обработку
if ($_POST['user']) {
    print "Hello,
    // вывести на экран значение параметра 'user' из
    // переданной на обработку формы
    print $_POST['user'];
    print "!";
} else {
    // иначе - вывести на экран саму форму
```

```
    print <<<_HTML_
      <form method="post" action="$_SERVER[PHP_SELF]">
      Your Name: <input type="text" name="user" />
      <br/>
      <button type="submit">Say Hello</button>
      </form>
      _HTML_;
}
?>
```

В примере 1.5 используется конструкция `if()`, чтобы проверить, отправил ли браузер значение параметра `user` из формы, переданной на обработку. Эта конструкция позволяет выбрать одно из следующих действий: вывести на экран приветствие или же саму форму. Более подробно конструкция `if()` обсуждается в главе 3, а применение переменной `$_SERVER[PHP_SELF]` и обработка форм — в главе 7.

В языке PHP имеется огромная библиотека внутренних функций, которые можно применять в своих программах. Эти функции помогают решать типичные задачи. К ним относится встроенная функция `number_format()`, предоставляющая отформатированный вариант числа. В примере 1.6 она применяется для вывода числа на экран.

Пример 1.6. Вывод отформатированного числа на экран

```
<?php print "The population of the US is about: ";
print number_format(320853904);
?>
```

Выполнение кода из примера 1.6 дает следующий результат:

```
The population of the US is about: 320,853,904
```

Функциям полностью посвящена глава 5. В ней поясняется написание собственных функций, а также синтаксис вызова функций и обработки результатов их выполнения. У многих функций, включая и упомянутую выше функцию `number_format()`, имеется *возвращаемое значение*. Оно получается в результате выполнения функции. Так, в примере 1.6 второму оператору `print` передается значение, возвращаемое функцией `number_format()`. В данном случае это численность населения США в формате с разделением запятой разрядов в целой части числа.

К числу наиболее распространенных типов программ, написанных на PHP, относится программа, отображающая веб-страницу, содержащую информацию, извлекаемую из базы данных. Если позволить параметрам передаваемой на обработку формы управлять процессом извлечения информации из базы данных, то тем самым можно добиться интерактивности веб-сайта. В примере 1.7 демонстрируется программа на PHP, подключающаяся к серверу базы данных, извлекающая список блюд и цены на них, исходя из значения параметра `meal` в форме, а затем выводящая на экран эти блюда и цены на них в HTML-таблице.

Пример 1.7. Отображение информации из базы данных

```
<?php
// использовать базу данных SQLite из файла 'dinner.db'
$db = new PDO('sqlite:dinner.db');
// определить, какие блюда имеются
$meals = array('breakfast','lunch','dinner');
// проверить, содержит ли параметр "meal" переданной
// на обработку формы одно из строковых значений
// "breakfast", "lunch" или "dinner"
if (in_array($_POST['meal'], $meals)) {
    // Если данный параметр содержит указанное значение,
    // получить все блюда для указанной трапезы
    $stmt = $db->prepare('SELECT dish,price FROM meals
                                    WHERE meal LIKE ?');
    $stmt->execute(array($_POST['meal']));
    $rows = $stmt->fetchAll();
    // Если блюда в базе данных не обнаружены, сообщить от этом
    if (count($rows) == 0) {
        print "No dishes available.";
    } else {
        // вывести на экран каждое блюдо и цену на него
        // в отдельной строке HTML-таблицы
        print '<table><tr><th>Dish</th><th>Price</th></tr>';
        foreach ($rows as $row) {
            print "<tr><td>$row[0]</td><td>$row[1]</td></tr>";
        }
        print "</table>";
    }
} else {
    // Это сообщение выводится на экран в том случае,
    // если параметр "meal" переданной на обработку формы
    // не содержит ни одно из строковых значений
    // "breakfast", "lunch" или "dinner"
    print "Unknown meal.";
}
?>
```

В примере 1.7 происходит немало интересного, но о простоте и эффективности PHP свидетельствует тот факт, что для формирования динамической веб-страницы с поддержкой базы данных потребовалось всего лишь 20 строк кода, не считая комментарии. Ниже поясняется, что же происходит в этих 20 строках кода.

Функция `new PDO()`, вызываемая в самом начале рассматриваемого здесь примера программы, устанавливает соединение с базой данных SQLite, находящейся в отдельном файле. Эта и другие функции обращения к базе данных из данного примера, в том числе `prepare()`, `execute()` и `fetchAll()`, подробнее поясняются в главе 8.

Элементы кода, начинающиеся в данном примере со знака **$**, например, `$db`, `$_POST`, `$stmt` или `$row`, являются переменными. В переменных хранятся значения, которые могут изменяться по ходу выполнения программы или задаваться и сохраняться в какой-то момент ее выполнения для последующего применения. Более подробно переменные обсуждаются в главе 2.

Следующая задача после подключения к базе данных состоит в том, чтобы выяснить, какую именно трапезу заказал пользователь. Массив `$meals` инициализируется для хранения дозволенных видов трапезы: `breakfast` (завтрак), `lunch` (ленч) и `dinner` (обед). В операторе

`in_array($POST['meal'], $meals)` проверяется, содержится ли значение (`$_POST['meal']`) параметра `meal` из переданной на обработку формы в массиве `$meals`. Если это значение в нем отсутствует, выполнение сразу же переходит в самый конец данного примера программы, где вслед за последним оператором else на экран выводится сообщение `"Unknown meal"` (Неизвестная трапеза).

Если была передана дозволенная трапеза, функции `prepare()` и `execute()` посылают запрос в базу данных. Так, если заказана трапеза `breakfast`, то посылается следующий запрос:

```
SELECT dish,price FROM meals WHERE meal LIKE 'breakfast'
```

Запросы базы данных SQLite и большинства других реляционных баз данных составляются на языке Structured Query Language (язык структурированных запросов — SQL), основы которого представлены в главе 8. Функция `prepare()` возвращает идентификатор, которым можно воспользоваться для получения дополнительной информации о запросе.

Упомянутый выше идентификатор применяется в функции `fetchAll()` для получения всех совпадающих трапез, обнаруженных в базе данных по запросу. Если же в ней отсутствуют подходящие трапезы, рассматриваемая здесь программа выводит на экран сообщение `"No dishes available"` (Блюда отсутствуют). В противном случае она отображает информацию о совпадающих трапезах.

Далее программа выводит на экран начало HTML-таблицы, а затем обрабатывает в конструкции `foreach` каждое блюдо, обнаруженное по запросу. Элементы массива, возвращаемого функцией `fetchAll()`, используются в операторе `print` для отображения блюд в отдельных строках HTML-таблицы.

Основные правила написания программ на PHP

В этом разделе закладывается основание под правила структурирования программ на PHP. Это более основательные правила, чем просто вывод информации на экран или сложение двух чисел. Их можно сравнить с рекомендацией читать страницы данной книги сверху вниз и слева направо или обращать больше внимания на выделенный черным текст, а не на крупные пробелы на странице.

Если у вас уже имеется хотя бы небольшой опыт программирования на PHP и вы предпочитаете поэкспериментировать с кнопками на своем новом проигрывателе типа Blu-Ray, прежде чем ознакомиться с их назначением в руководстве пользователя, можете сразу перейти к главе 2, вернувшись к этому разделу впоследствии. А если вы стремитесь безотлагательно писать программы на PHP, но они ведут себя неожиданно или содержат ошибки синтаксического анализа, о которых сообщает интерпретатор PHP при попытке выполнить программу, обратитесь к материалу этого раздела за справкой.

Начальные и конечные дескрипторы

В каждом из примеров программ, рассмотренных ранее в этой главе, применялись дескрипторы `<?php` и `?>` кода PHP как начальный и конечный соответственно. Все, что находится за пределами этих дескрипторов, игнорируется интерпретатором PHP. Текст перед начальным дескриптором или после конечного дескриптора выводится без вмешательства интерпретатора PHP. Конечный дескриптор можно оставить в конце файла PHP. Если интерпретатор PHP достигает конца файла и не обнаруживает конечный дескриптор PHP, он действует так, как будто это самый последний элемент в файле. Это очень удобно для гарантии того, что невидимый дополнительный материал (например, пустые строки) после конечного пробела не появится случайно в результате, выводимом из программы.

Программа на PHP может содержать несколько начальных и конечных пар дескрипторов, как демонстрируется в примере 1.8.

Пример 1.8. Несколько начальных и конечных пар дескрипторов

```
Five plus five is:
<?php print 5 + 5; ?>
<p>
Four plus four is:
<?php
  print 4 + 4;
?>
<p>
<img src="vacation.jpg" alt="My Vacation" />
```

Исходный код PHP, заключенный в каждую пару дескрипторов <?php ?>, обрабатывается интерпретатором PHP, а остальная часть страницы выводится в исходном виде. Так, в результате выполнения кода из примера 1.8 выводится следующий результат:

```
Five plus five is:
10<p>
Four plus four is:
8<p>
<img src="vacation.jpg" alt="My Vacation" />
```

В ряде устаревших программ на PHP в качестве начального употребляется дескриптор <?, а не <?php. В таком случае дескриптор <? называется *коротким открывающим дескриптором*, поскольку он короче, чем дескриптор <?php. Но лучше все же пользоваться обычным открывающим дескриптором <?php, поскольку этим гарантируется нормальная работа программы на любом сервере, где выполняется интерпретатор PHP. Поддержка короткого открывающего дескриптора может быть включена или отключена с помощью соответствующего параметра конфигурации PHP. В приложении А поясняется, как видоизменить конфигурацию PHP, чтобы определить, какие именно открывающие дескрипторы оказываются достоверными в программах на PHP.

Все примеры программ, приведенных в остальной части этой главы, начинаются с дескриптора <?php и оканчиваются дескриптором ?>. А в последующих главах не все примеры программ содержат начальные и конечные дескрипторы. Но не забывайте, что они необходимы в программах для правильного распознавания написанного вами кода интерпретатором PHP.

Пробелы и учет регистра букв

Как и все программы на PHP, примеры, представленные в этом разделе, состоят из последовательного ряда операторов, каждый из которых завершается точкой с запятой. В одной строке кода можно разместить несколько операторов PHP, при условии, что они разделены точкой с запятой. Между операторами допускается любое количество пробелов, которые игнорируются интерпретатором PHP. Точка с запятой указывает интерпретатору PHP на окончание одного оператора и начало другого. Отсутствие или большое количество пробелов между операторами не оказывает никакого влияния на ход выполнения программы. Пробелами в программировании называют внешне пустые знаки пробела, табуляции и новой строки.

На практике рекомендуется вводить по одному оператору в каждой строке исходного кода, размещая пустые строки между операторами только в том случае, если это повышает удобочитаемость исходного кода. Разрядка в примерах 1.9 и 1.10 произведена неудачно. Вместо этого исходный код следует форматировать так, как показано в примере. 1.11.

Пример 1.9. Этот код PHP чрезмерно сжат

```
<?php print "Hello"; print " World!"; ?>
```

Пример 1.10. А этот код PHP чрезмерно растянут

```
<?php

print "Hello";

print " World!";

?>
```

Пример 1.11. Этот код PHP отформатирован верно

```
<?php
print "Hello";
print " World!";
?>
```

Помимо пробелов между строками кода, интерпретатор PHP игнорирует пробелы между ключевыми словами данного языка и значениями. Так, между ключевым словом `print` и символьной строкой `"Hello, World!"` может быть один, сто или вообще ни одного пробела, как, впрочем, и между этой строкой и точкой с запятой в конце строки кода.

В качестве хорошего стиля программирования рекомендуется разделять оператор `print` и выводимое значение одним пробелом, а после этого значения сразу же указывать точку с запятой. В примере 1.12 демонстрируются три строки кода с разной разрядкой: одна — чрезмерно растянута, другая — чрезмерно сжата, а третья — отформатирована верно.

Пример 1.12. Разрядка

```
<?php
print                  "Too many spaces" ;
print"Too few spaces";
print "Just the right amount of spaces";
?>
```

Ключевые слова языка PHP (например, `print`) и имена функций (например, `number_format`) указываются без учета регистра букв. Интерпретатор PHP не различает прописные и строчные буквы в ключевых словах и именах функций, указываемых в программах. Так, операторы, приведенные в примере 1.13, одинаковы с точки зрения интерпретатора PHP.

Пример 1.13. Ключевые слова языка и имена функций указываются без учета регистра букв

```
<?php
// Во всех приведенных ниже строках кода выполняется
// одно и то же действие
print number_format(320853904);
PRINT Number_Format(320853904);
Print number_format(320853904);
pRiNt NUMBER_FORMAT(320853904);
?>
```

Комментарии

Как демонстрировалось в ряде примеров программ ранее в этой главе, комментарии служат для пояснения другим людям особенностей работы программы. Комментарии в исходном коде составляют важную часть любой программы. Вводимый вами исходный код может казаться совершенно ясным, когда вы программируете. Но через несколько месяцев, когда вам потребуется вернуться к написанной программе, чтобы внести в нее изменения, логика ее выполнения может показаться не столь ясной и очевидной. И здесь на помощь приходят комментарии. Объясняя простым языком принцип действия программы, комментарии делают ее намного более понятной.

Комментарии еще более важны для того, кто не является первоначальным автором программы, но должен внести в нее изменения. Сделайте одолжение себе и всем, кому придется читать исходный код вашей программы, снабдив ее обильными комментариями.

В языке PHP предусмотрены самые разные способы снабжения программ комментариями в силу их особой важности. Так, в приведенных ранее примерах строки комментариев начинались со знаков **//**. Эти знаки указывают интерпретатору PHP считать всю остальную строку комментарием. А по окончании строки комментария исходный код интерпретируется, как обычно. Подобный стиль комментариев применяется и в других языках программирования, в том числе C++, JavaScript и Java. Знаки **//** можно указывать в строке кода и после оператора, и тогда остальная часть строки интерпретируется как комментарий. В языке PHP поддерживаются также однострочные комментарии в стиле языка Perl и командного процессора. Такие комментарии начинаются со знака **#**. Комментарии можно начинать со знака **#** там же, где они начинаются со знаков **//**, но в современном стиле программирования предпочтение все же отдается знакам **//**. Некоторые образцы однострочных комментариев приведены в примере 1.14.

Пример 1.14. Однострочные комментарии, обозначаемые знаками // и #

```
<?php
// Это строка комментария
print "Smoked Fish Soup ";
print 'costs $3.25.';

# ввести еще одно блюдо в меню
print 'Duck with Pea Shoots ';
print 'costs $9.50.';
// Знаки // или # можно употреблять в однострочных
// комментариях. Комментарии начинаются со знаков
// // или # в любом другом месте строки кода
print 'Shark Fin Soup'; // Надеюсь, что это вкусно!
print 'costs $25.00!'; # А это дороговато!
# Комментарии не начинаются со знаков // или #,
# указываемых в символьной строке
print 'http://www.example.com';
print 'http://www.example.com/menu.php#dinner';
?>
```

Многострочные комментарии начинаются со знаков **/*** и оканчиваются знаками ***/**. А все, что находится между знаками **/*** и ***/**, воспринимается интерпретатором PHP как комментарий. Многострочные комментарии удобны для временного отключения небольшого блока кода. Некоторые образцы многострочных комментариев приведены в примере 1.15.

Пример 1.15. Многострочные комментарии

```
<?php
/* Мы собираемся ввести в меню следующие блюда:
   - Суп из копченой рыбы
   - Утка с гороховыми побегами
   - Суп из акульих плавников
*/
print 'Smoked Fish Soup, Duck with Pea Shoots, Shark Fin Soup ';
print 'Cost: 3.25 + 9.50 + 25.00';

/* А это прежнее меню:
Приведенные ниже строки кода заключены в данный комментарий,
и поэтому они не выполняются.
print 'Hamburger, French Fries, Cola ';
print 'Cost: 0.99 + 1.25 + 1.50';
*/
?>
```

В языке PHP не существует строгих правил в отношении наилучшего стиля комментариев. Многострочные комментарии зачастую наиболее удобны в употреблении, особенно если требуется закомментировать блок кода или описать функцию в нескольких строках. Но если требуется присоединить краткое пояснение в конце строки кода, то для этой цели вполне подойдет комментарий в стиле **//**. Пользуйтесь тем стилем комментариев, который вам больше всего подходит.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Применение языка PHP на веб-сервере для составления ответа или документа, посылаемого обратно браузеру.
- Применение языка PHP на стороне сервера для выполнения программ на веб-сервере, в отличие от таких языков, как JavaScript, программы на которых выполняются на стороне клиента, т.е. в веб-браузере.
- Принимая решение, применять ли язык PHP, следует иметь в виду, что он свободно доступен (бесплатно и как речь), не зависит от конкретной платформы, широко распространен и предназначен для веб-разработки.
- Организация вывода информации, обработки форм и взаимодействия с базой данных в программах на PHP.
- Основы структурирования программ на PHP, включая начальные и конечные дескрипторы PHP (`?php` и `?>`), пробелы, учет регистра букв и комментарии.

ГЛАВА 2

Обработка числовых и текстовых данных

В языке PHP можно обрабатывать самые разные типы данных. Из этой вы главы узнаете о таких отдельных типах данных, как числа и фрагменты текста, о том, как вводить в программы тест и числа, о некоторых ограничениях, накладываемых интерпретатором PHP на эти типы данных, а также ознакомитесь с самыми распространенными приемами их обработки.

В большинстве программ на PHP немало времени уделяется обработке текста, поскольку им приходится формировать HTML-разметку и обрабатывать информацию, извлекаемую из базы данных. HTML-разметка представляет собой отформатированный особым способом текст, а информация в базе данных (например, имя пользователя, описание продукции или адрес) — фрагмент текста. Чем легче обрабатывать текст, тем легче формировать динамические веб-страницы.

Действие переменных было продемонстрировано в главе 1, а в этой главе они рассматриваются более подробно. Переменная является именованным контейнером, где хранится значение, которое может изменяться по ходу выполнения программы. Переменные используются для доступа к данным, передаваемым в форме или обмениваемым с базой данных. На практике обращение к переменной можно сравнить с проверкой остатков на банковском счете. Величина остатков на банковском счете колеблется во времени, а в программе на PHP переменная может содержать значение параметра формы, переданной на обработку. Всякий раз, когда выполняется программа, значение этого параметра может оказаться иным. Но каким бы ни было это значение, к нему можно обратиться по одному и тому же имени. В этой главе также поясняется, как создавать переменные и выполнять над ними различные операции, в том числе изменять и выводить их значения на экран.

Текст

Фрагменты текста, применяемые в компьютерных программах, называются *символьными строками*, потому что они состоят из отдельных символов, составляющих строку. Символьные строки могут состоять из букв, чисел, знаков препинания, пробела, табуляции и любых других символов. К некоторым примерам относятся символьные строки `I would like 1 bowl of soup, "Is it too hot?"he asked` и `There's no spoon!`. Символьная строка может даже содержать двоичный файл, например, изображения или фонограммы. Единственным ограничением на длину символьной строки в программе на PHP служит объем оперативной памяти компьютера.

Символьные строки в PHP являются последовательностями *байтов*, а не символов. Если вы имеете дело только с текстом на английском языке, то никак не ощущаете это отличие. А если вы работаете с текстом на другом языке и вам нужно обеспечить правильность интерпретации символов из других алфавитов, непременно прочитайте главу 20, где поясняется, как обращаться с различными наборами символов.

Определение символьных строк текста

Обозначить символьную строку в программе на PHP можно несколькими способами. Это проще всего сделать, заключив символьную строку в одиночные кавычки, как показано ниже.

```
print 'I would like a bowl of soup.';
print 'chicken';
print '06520';
print '"I am eating dinner," he growled.';
```

Символьная строка состоит из всего, что заключено в одиночные кавычки. Поэтому при выполнении приведенных выше строк кода на экран выводится следующий результат:

```
I would like a bowl of soup.chicken06520"I am eating dinner," he growled.
```

Обратите внимание на то, что результат выполнения всех четырех упомянутых выше строк кода появляется в одной строке. Команда `print` не вводит никаких дополнительных пробелов[1].

Одиночные кавычки не являются частью символьной строки, а служат лишь в качестве *ограничителей*, указывающих интерпретатору PHP, где начинается и оканчивается символьная строка. Если же требуется включить одиночную кавычку в символьную строку, ее следует предварить знаком обратной косой черты (\) в самой строке, как показано ниже.

```
print 'We\'ll each have a bowl of soup.';
```

Последовательность символов \' преобразуется в одиночную кавычку в символьной строке. Поэтому при выполнении приведенной выше строки кода на экран выводится следующий результат:

```
We'll each have a bowl of soup.
```

Знак обратной косой черты указывает интерпретатору PHP на то, что следующий далее символ следует интерпретировать буквально как одиночную кавычку, а не знак, обозначающий конец символьной строки. Такая операция называется *экранированием*, а знак обратной косой черты — *экранирующим* или *управляющим символом*. Управляющий символ указывает системе сделать нечто особенное со следующим после него символом. Одиночная кавычка в символьной строке, заключенной в одиночные кавычки, обычно обозначает конец символьной строки. Но назначение этого знака изменяется на буквальное, т.е. на одиночную кавычку, если он предваряется знаком обратной косой черты.

[1] В некоторых программах на PHP можно также обнаружить, что для вывода текста применяется команда `echo`. Она действует таким же образом, как и команда `print`.

Закругленные кавычки и текстовые редакторы

В текстовых процессорах и редакторах прямые кавычки ' и " нередко преобразуются в закругленные кавычки ‘, ’, “ и ”. Интерпретатор PHP распознает только прямые кавычки в качестве ограничителей символьных строк. Если вы пишете программы на PHP в текстовом процессоре или редакторе, автоматически вводящем закругленные кавычки в исходный текст программ, то укажите текстовому процессору или редактору одно из двух: перестать это делать или использовать другие кавычки. В таких текстовых редакторах, как Emacs, Vi, Sublime Text или Windows Notepad, введенные кавычки оставляются в прежнем виде.

Сам управляющий символ может быть также экранирован. Чтобы включить буквальный знак обратной косой черты в символьную строку, его следует предварить еще одним знаком обратной косой черты:

```
print 'Use a \\ to escape in a string';
```

При выполнении приведенной выше строки кода на экран выводится следующий результат:

```
Use a \ to escape in a string
```

Первый знак обратной косой черты является управляющим символом, указывая интерпретатору PHP на иное значение последующего символа. В итоге второй знак обратной косой черты интерпретируется буквально как входящий в состав символьной строки, а не как управляющий символ.

Следует, однако, иметь в виду, что знаки обратной косой черты воспроизводятся по диагонали от левого верхнего к правому нижнему углу, тогда как знаки прямой косой черты — от левого нижнего угла к правому верхнему. Напомним, что два знака косой черты (//) обозначают комментарии в программе на PHP.

В символьные строки, заключаемые в одиночные кавычки, можно вводить пробелы, в том числе знаки перевода строки, как показано ниже.

```
print '<ul>
<li>Beef Chow-Fun</li>
<li>Sauteed Pea Shoots</li>
<li>Soy Sauce Noodles</li>
</ul>';
```

В результате выполнения приведенных выше строк кода на экран выводится несколько следующих строк, размеченных в формате HTML:

```
<ul>
<li>Beef Chow-Fun</li>
<li>Sauteed Pea Shoots</li>
<li>Soy Sauce Noodles</li>
</ul>
```

Перевод строки в конце выводимой символьной строки отсутствует, поскольку одиночная кавычка, обозначающая конец символьной строки, следует сразу же за дескриптором .

Специальную интерпретацию в символьной строке, заключаемой в одиночные кавычки, получают только знаки обратной косой черты и одиночной кавычки. Все остальные знаки интерпретируются буквально.

Символьные строки можно заключать и в двойные кавычки. Такие символьные строки аналогичны символьным строкам в одиночных кавычках, но они могут содержать больше специальных символов, которые перечислены в табл. 2.1.

Таблица 2.1. Специальные символы в строках, заключаемых в двойные кавычки

Символ	Назначение
`\n`	Новая строка (значение `10` в коде ASCII)
`\r`	Возврат каретки (значение `13` в коде ASCII)
`\t`	Табуляция (значение `9` в коде ASCII)
`\\`	Обратная косая черта (`\`)
`\$`	Денежная единица (`$`)
`\"`	Двойная кавычка (»)
`\0 .. \777`	Восьмеричное число (по основанию `8`)
`\x0 .. \xFF`	Шестнадцатеричное число (по основанию `16`)

Символьные строки в одиночных и двойных кавычках отличаются, главным образом, следующим: если имя переменной указывается в символьной строке, заключаемой в двойные кавычки, то значение этой переменной подставляется вместо ее имени в символьную строку, чего не происходит с символьными строками в одиночных кавычках. Так, если переменная `$user` содержит значение `Bill`, символьная строка `'Hi $user'` интерпретируется буквально следующим образом: `Hi $user`, тогда как символьная строка `"Hi $user"` — таким образом: `Hi Bill`. Подробнее об этом речь пойдет далее, в разделе “Переменные”.

Как упоминалось в разделе “PHP в действии” главы 1, символьные строки можно также определять с помощью синтаксиса *встраиваемого документа*. Встраиваемый документ начинается с идентификатора `<<<` и ограничивающего слова, а оканчивается тем же самым словом в начале строки кода. Синтаксис встроенного документа приведен в примере 2.1.

Пример 2.1. Встраиваемый документ

```
<<<HTMLBLOCK
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles </ul>
</body>
</html>
HTMLBLOCK
```

В примере 2.1 ограничительным является слово `HTMLBLOCK`. Ограничители встраиваемого документа могут состоять из букв, чисел и знаков подчеркивания. Первым символом ограничителя должна быть буква или знак подчеркивания. Ограничители встраиваемого документа рекомендуется указывать прописными буквами, чтобы визуально выделить этот документ. Ограничитель, обозначающий конец встраиваемого документа, должен быть указан в отдельной строке кода. Ограничитель нельзя указывать с отступом, пробелами, комментариями или другими символами после него. Единственным исключением из этого правила является точка с запятой, которую допускается указывать сразу же после ограничителя для завершения оператора. Этому правилу следует исходный код, приведенный в примере 2.2, для вывода встраиваемого документа на экран.

Пример 2.2. Вывод встраиваемого документа на экран

```
print <<<HTMLBLOCK <html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles
</ul>
</body>
</html>
HTMLBLOCK;
```

Встраиваемые документы подчиняются тем же правилам экранирования символов и подстановки значений переменных, что и символьные строки в двойных кавычках. Благодаря этому они особенно удобны для определения или вывода на экран символьной строки, содержащей немало текста или HTML-разметки в сочетании с переменными, как демонстрируется далее, в примере 2.2.

Для соединения двух символьных строк служит знак точки (`.`), обозначающий операцию сцепления строк. Ниже приведен ряд примеров сцепления символьных строк.

```
print 'bread' . 'fruit';
print "It's a beautiful day " . 'in the neighborhood.';
print "The price is; " . '$3.95';
print 'Inky' . 'Pinky' . 'Blinky' . 'Clyde';
```

Соединенные символьные строки выводятся на экран следующим образом:

```
breadfruit
It's a beautiful day in the neighborhood.
The price is: $3.95
InkyPinkyBlinkyClyde
```

Манипулирование текстом

В языке PHP имеется целый ряд полезных функций для обработки символьных строк. В этом разделе представлены те функции, которые оказываются наиболее полезными для решения типичных задач проверки достоверности и форматирования. В разделе “Строки” руководства по PHP, оперативно доступном по адресу `http://www.php.net/strings`, можно найти немало сведений о встроенных в PHP функций обработки символьных строк.

Проверка достоверности символьных строк

Проверка достоверности — это процесс проверки данных, поступающих из внешнего источника, на соответствие предполагаемому формату или назначению. В ходе данного процесса, например, проверяется, действительно ли пользователь ввел почтовый индекс и правильный адрес электронной почты в соответствующих полях заполняемой формы. Все вопросы обработки форм подробно обсуждаются в главе 7, но поскольку данные заполненной формы передаются на обработку в программе на PHP в виде символьных строк, то в этом разделе поясняется, каким образом проверяется достоверность подобных строк.

Функция `trim()` удаляет пробел в начале и в конце символьной строки. Этой функцией можно воспользоваться в сочетании с функцией `strlen()`, возвращающей длину заданной строки, для

определения длины переданного значения, игнорируя начальные и конечные пробелы, как показано в примере 2.3. А условный оператор `if()`, применяемый в примере 2.3, более подробно обсуждается в главе 3.

Пример 2.3. Проверка длины усеченной символьной строки

```
// Переменная $_POST['zipcode'] содержит значение параметра
// "zipcode" из переданной на обработку формы
$zipcode = trim($_POST['zipcode']);
// А теперь это значение с удаленными начальными и конечными
// пробелами содержит переменная $zipcode
$zip_length = strlen($zipcode);
// выдать предупреждение, если указан почтовый индекс
// длиной меньше 5 символов
if ($zip_length != 5) {
  print "Please enter a zip code that is 5 characters long.";
}
```

Благодаря применению функции `trim()` никто не сможет ввести почтовый индекс, состоящий, например, из цифр **732** и двух пробелов. Иногда лишние пробелы вводятся случайно, а порой — злонамеренно. Но как бы там ни было, их следует отбросить, когда это уместно, чтобы получить требуемую длину символьной строки.

Вызовы функций `trim()` и `strlen()` можно связать в цепочку, чтобы сделать исходный код более кратким. Так, в примере 2.4 делается то же самое, что и в примере 2.3.

Пример 2.4. Более краткая форма проверки длины усеченной символьной строки

```
if (strlen(trim($_POST['zipcode'])) != 5) {
  print "Please enter a zip code that is 5 characters long.";
}
```

В первой строке кода из примера 2.4 происходит следующее. Во-первых, значение переменной `$_POST['zipcode']` передается функции `trim()`. Во-вторых, значение, возвращаемое этой функцией, т.е. усеченное значение переменной `$_POST['zipcode']` без начальных и конечных пробелов, передается функции `strlen()`, которая, в свою очередь, возвращает длину усеченной символьной строки. В-третьих, длина этой строки сравнивается с числовым значением **5**. И, наконец, если длина символьной строки не равна **5**, то выполняется оператор `print` в блоке условного оператора `if()`.

Чтобы сравнить две символьные строки, следует воспользоваться операцией равенства (**==**), как показано в примере 2.5.

Пример 2.5. Сравнение символьных строк с помощью операции равенства

```
if ($_POST['email'] == 'president@whitehouse.gov') {
  print "Welcome, US President.";
}
```

Оператор `print` выполняется в коде из примера 2.5 лишь в том случае, если параметр `email` передаваемой на обработку формы содержит строковое значение `president@whitehouse.gov`, набранное только строчными буквами. Дело в том, что при сравнении символьных строк с помощью операции **==** учитывается регистр букв. В частности, строковые значения `president@whitehouse.gov`, `president@whitehouse.GOV` и `president@whitehouse.gov` не одинаковы.

Чтобы сравнить символьные строки без учета регистра букв, следует воспользоваться функцией `strcasecmp()`. Эта функция сравнивает две символьные строки, игнорируя отличия в регистре

букв. Если обе символьные строки, предоставляемые функции `strcasecmp()`, окажутся одинаковыми независимо от отличий в прописных и строчных буквах, она возвратит нулевое значение. В примере 2.6 показано, как пользоваться функцией `strcasecmp()`.

Пример 2.6. Сравнение символьных строк без учета регистра

```
if (strcasecmp($_POST['email'], 'president@whitehouse.gov') == 0) {
  print "Welcome back, OS President.";
}
```

Оператор `print` выполняется в коде из примера 2.6 лишь в том случае, если параметр `email` передаваемой на обработку формы содержит строковое значение `President@Whitehouse.Gov`, `PRESIDENT@WHITEHOUSE.GOV`, `presIDENT@whiteHOUSE.GoV` или любой другой вариант выделения прописными буквами строкового значения `president@whitehouse.gov`.

Форматирование текста

Функция `printf()` предоставляет более полный контроль над внешним видом выводимого результата по сравнению с оператором `print`. Функции `printf()` можно передать строку форматирования и целый ряд выводимых на экран элементов. Каждое правило в строке форматирования заменяется одним выводимым элементом. Применение функции `printf()` демонстрируется в примере 2.7.

Пример 2.7. Форматированный вывод цены с помощью функции `printf()`

```
$price = 5; $tax = 0.075;
printf('The dish costs $%.2f', $price * (1 + $tax));
```

При выполнении приведенных выше строк кода на экран выводится следующий результат:

```
The dish costs \$5.38
```

В примере 2.7 правило форматирования **`%.2f`** заменяется значением выражения `$price * (1 + $tax)`. А выводимый результат форматируется таким образом, чтобы числовое значение цены на блюдо содержало два разряда после десятичной точки.

Правила в строке форматирования начинаются со знака **`%`**, после чего указываются перечисленные ниже дополнительные, хотя и необязательные модификаторы, которые определяют, каким должно быть правило форматирования.

- **Заполняющий символ**. Если символьная строка, заменяющая правило форматирования, слишком короткая, то данный символ дополняет ее до нужной длины. В качестве заполняющего символа обычно употребляется пробел или нуль.
- **Знак**. Для чисел знак “плюс” (**`+`**) при вызове функции `printf()` означает, что этот знак ставится перед положительными числами (как правило, они выводятся без знака). А для символьных строк знак “минус” (**`-`**) при вызове функции `printf()` означает, что символьная строка выравнивается по правому краю (хотя, как правило, она выравнивается по левому краю).
- **Минимальная ширина**. Определяет минимальное пространство, которое отводится под значение, заменяющее правило форматирования. Если оно оказывается более коротким, то дополняется заполняющим символом.
- **Точка и числовое выражение точности**. Для чисел с плавающей точкой этот модификатор определяет, сколько цифр следует после десятичной точки. В примере 2.7 это единственный указанный модификатор. Так, если указан модификатор **`.2`**, значение выражения `$price * (1 + $tax)` будет отформатировано с двумя разрядами после десятичной точки.

После модификаторов следует обязательный символ, обозначающий тип выводимого значения. В частности, символ **d** обозначает десятичное число, символ **s** — символьную строку, а символ **f** — число с плавающей точкой.

Если эта загадочная смесь, состоящая из знака процента и модификаторов, не вкладывается в вашу голову, не отчаивайтесь! Чаще всего функция `printf()` применяется для вывода цен, отформатированных по правилу **%.2f**, приведенному в примере 2.7. Если вы не в состоянии пока что воспринимать остальные правила форматирования, просто запомните, что эту функцию следует вызывать, когда требуется отформатировать выводимое на экран десятичное числовое значение.

Но при более тщательном рассмотрении данной функции в ней можно обнаружить немало других полезных возможностей. Например, с помощью заполняющего символа **0** и минимальной ширины можно надлежащим образом отформатировать дату или почтовый индекс с начальными нулями, как показано в примере 2.8.

Пример 2.8. Заполнение нулями значения, выводимого на экран с помощью функции `printf()`

```
$zip = '6520';
$month = 2;
$day = 6;
$year = 2007;

printf("ZIP is %05d and the date is %02d/%02d/%d",
        $zip, $month, $day, $year);
```

При выполнении кода из примера 2.8 на экран выводится следующий результат:

```
ZIP is 06520 and the date is 02/06/2007
```

Модификатор знака удобен для явного указания положительных и отрицательных величин. В примере 2.9 он употребляется для отображения ряда температур.

Пример 2.9. Отображение чисел со знаками с помощью функции `printf()`

```
$min = -40;
$max = 40;
printf("The computer can operate between %+d and %+d degrees Celsius.",
        $min, $max);
```

При выполнении кода из примера 2.9 на экран выводится следующий результат:

```
The computer can operate between -40 and +40 degrees Celsius.
```

Более подробно ознакомиться с возможностями функции `printf()` можно на странице документации по PHP, оперативно доступной по адресу `http://www.php.net/printf`.

Еще одним способом форматирования текста является манипулирование регистром букв в символьных строках. В частности, функция `strtolower()` делает все буквы строчными, а функция `strtoupper()` — прописными в символьной строке. Применение функции `strtolower()` демонстрируется в примере 2.10.

Пример 2.10. Смена регистра букв

```
print strtolower('Beef, CHICKEN, Pork, duCK');
print strtoupper('Beef, CHICKEN, Pork, duCK');
```

При выполнении кода из примера 2.10 на экран выводится следующий результат:

```
beef, chicken, pork, duck
BEEF, CHICKEN, PORK, DUCK
```

Функция `ucwords()` делает прописной первую букву каждого слова в символьной строке. Ею удобно пользоваться в сочетании с функцией `strtolower()`, чтобы выделять заглавными буквами имена, если они исходно набраны только прописными буквами. В примере 2.11 демонстрируется, как сочетать функции `strtolower()` и `ucwords()`.

Пример 2.11. Выделение имен заглавными буквами с помощью функции ucwords()

```
print ucwords(strtolower('JOHN FRANKENHEIMER'));
```

При выполнении кода из примера 2.11 на экран выводится следующий результат:

```
John Frankenheimer
```

С помощью функции `substr()` можно извлечь лишь часть символьной строки, например, для того, чтобы отображать начало сообщений на итоговой странице. В примере 2.12 демонстрируется применение функции `substr()` для усечения строкового значения параметра `comments` передаваемой на обработку формы.

Пример 2.12. Усечение символьной строки с помощью функции substr()

```
// Извлечь первые 30 байтов из строкового значения
// переменной $_POST['comments']
print substr($_POST['comments'], 0, 30);
// добавить многоточие
print '...';
```

Если упомянутый выше параметр передаваемой на обработку формы содержит следующий комментарий:

```
The Fresh Fish with Rice Noodle was delicious,
but I didn't like the Beef Tripe.
```

то при выполнении кода из примера 2.12 на экран выводится такой результат:

```
The Fresh Fish with Rice Noodle...
```

В качестве трех аргументов функция `substr()` принимает обрабатываемую символьную строку, начальное положение извлекаемой подстроки и количество извлекаемых байтов. Начало символьной строки находится на позиции **0**, а не **1**, и поэтому вызов функции substr(`$_POST['comments'], 0, 30`) означает следующее: извлечь 30 байтов из строкового значения переменной `$_POST['comments']`, начиная с самого его начала.

Если в качестве начального положения при вызове функции `substr()` указывается отрицательное число, она производит отсчет от конца исходной символьной строки, чтобы определить начало извлекаемой подстроки. Так, начальное положение **-4** означает следующее: начать извлечение подстроки, отступив четыре байта от конца исходной строки. В примере 2.13 отрицательное начальное положение используется для отображения только последних четырех цифр номера кредитной карточки.

Пример 2.13. Извлечение конца символьной строки с помощью функции `substr()`

```
print 'Card: XX';
print substr($_POST['card'], -4,4);
```

Если параметр `card` передаваемой на обработку формы содержит номер кредитной карточки **4000-1234-5678-9101**, то при выполнении кода из примера 2.13 на экран выводится следующий результат:

```
Card: XX9101
```

Вместо вызова функции `substr($_POST['card'], -4,4)` проще вызвать функцию `substr ($_POST['card'], -4)`. Ведь если оставить последний аргумент в вызове данной функции, она возвратит подстроку, начиная с заданного начального положения (отрицательного или положительного) и до конца исходной символьной строки.

Функция `str_replace()` заменяет отдельные части исходной символьной строки вместо того, чтобы извлекать из нее подстроку. С этой целью она находит подстроку и заменяет ее новой строкой. Это удобно для простой настройки HTML-разметки по шаблону. В примере 2.14 функция `str_replace()` применяется для установки атрибута `class` в дескрипторах `<span>`.

Пример 2.14. Применение функции `str_replace()`

```
$html = '<span class="class">Fried Bean Curd<span>
<span class="class">Oil-Soaked Fish</span>';

print str_replace('class', $my_class, $html);
```

Если в переменной `$my_class` установлено значение `lunch`, то при выполнении кода из примера 2.14 на экран выводится следующий результат:

```
<span class="lunch">Fried Bean Curd<span>
<span class="lunch">Oil-Soaked Fish</span>
```

Каждый экземпляр шаблона `class`, указываемого в качестве первого аргумента функции `str_replace()`, заменяется значением `lunch` переменной `$my_class` в символьной строке, задаваемой в качестве третьего аргумента данной функции.

Числа

Числа выражаются в PHP с помощью знакомого обозначения, хотя для разделения тысяч нельзя воспользоваться запятыми или другими символами. Чтобы воспользоваться десятичной дробью, не нужно делать ничего особенного по сравнению с целым числом. В примере 2.15 на экран выводится ряд чисел, достоверных в PHP.

Пример 2.15. Числа

```
print 56;
print 56.3;
print 56.30;
print 0.774422;
print 16777.216;
print 0;
print -213;
print 1298317;
print -9912111;
print -12.52222;
print 0.00;
```

Применение разных типов чисел

В интерпретаторе PHP проводится внутреннее различие между числами с дробной частью и без таковой. К первой категории относятся *числа с плавающей точкой*, а ко второй — *целые числа*. Числа с плавающей точкой называются так потому, что десятичная точка может "плавать", представляя числа с разной степенью точности.

В интерпретаторе PHP применяется математический аппарат базовой операционной системы для представления чисел. Поэтому самые крупные и самые мелкие числа с плавающей точкой, которыми можно воспользоваться, а также количество десятичных разрядов в них варьируются в разных системах.

Внутреннее представление чисел с плавающей точкой и целых чисел в интерпретаторе PHP отличается, в частности, точностью их хранения. Так, целое число **47** сохраняется в точности как **47**. А число с плавающей точкой **46,3** может быть сохранено как **46,2999999**. Это оказывает непосредственное влияние на точность способа сравнения чисел. В разделе "Принятие сложных решений" главы 3 поясняются операции сравнения и показывается, как правильно сравнивать числа с плавающей точкой.

Арифметические операции

Арифметические операции в PHP выполняются таким же образом, как и на уроках в начальной школе, только намного быстрее. Некоторые арифметические операции приведены в примере 2.16.

Пример 2.16. Арифметические операции в PHP

```
print 2 + 2;
print 17 - 3.5;
print 10 / 3;
print 6 * 9;
```

При выполнении кода из примера 2.16 на экран выводится следующий результат:

```
4
13.5
3.3333333333333
54
```

Помимо знака "плюс" (**+**) для обозначения операции сложения, знака "минус" (**–**) для операции вычитания, знака косой черты (**/**) для операции деления и знака "звездочка" (*****) для операции умножения, в PHP поддерживается обозначение операции возведения в степень двумя знаками "звездочка" (******), а операции деления по модулю, в результате которой возвращается остаток от деления, — знаком процента (**%**): `print 17 % 3;`

При выполнении приведенной выше строки кода на экран выводится следующий результат:

```
2
```

Если разделить число **17** на **3**, то получится целая часть **5** и остаток **2**, поэтому деление по модулю **17 % 3** равно **2**. Операция деления по модулю удобна для вывода строк, имена классов CSS которых чередуются в HTML-таблице, как будет показано в примере 4.13.

Операция возведения в степень была внедрена в версии PHP 5.6. Если вы пользуетесь более ранней версией PHP, вызывайте функцию `pow()` для возведения чисел в степень.

Арифметические, а также другие операции в PHP, рассматриваемые далее в данной книге, подчиняются правилу строгого старшинства операций. Именно таким образом в интерпретаторе PHP определяется порядок выполнения вычислений, если они составлены неоднозначно. Например, выражение **3 + 4 * 2** может, с одной стороны, означать сначала сложение **3** и **4**, а затем умножение полученной суммы на **2**, что в конечном итоге дает **14**. А с другой стороны, это выражение может означать сложение **3** с произведением **4** на **2**, что в конечном итоге дает **11**. В языке PHP в частности и в математике вообще операция умножения имеет более высокое старшинство, чем операция сложения, и поэтому правильной является вторая интерпретация упомянутого выше выражения. Таким образом, в интерпретаторе PHP сначала производится умножение **4** на **2**, а затем полученное произведение складывается с **3**.

С таблицей старшинства всех операций можно ознакомиться на странице руководства по PHP, оперативно доступной по адресу `http://www.php.net/language.operators.precedence`. Чтобы не запоминать эту таблицу и не обращаться к ней неоднократно за справкой, рекомендуется благоразумно пользоваться круглыми скобками. Группирование операций в круглых скобках однозначно определяет те операции, которые интерпретатор PHP должен в первую очередь производить в самих скобках. Так, выражение **(3 + 4) * 2** означает сначала сложение **3** и **4**, а затем умножение полученной суммы на **2**. А выражение **3 + (4 * 2)** означает сначала умножение **4** на **2**, а затем сложение полученного произведения с **3**.

Как и в других современных языках программирования, в PHP не нужно делать ничего особенного, чтобы результат вычислений был верно представлен целыми числами или числами с плавающей точкой. Так, в результате деления одного целого числа на другое получается число с плавающей точкой, если эти числа не делятся нацело. А если в результате какой-нибудь операции целое число выходит за максимальный или минимальный пределы представления целых чисел, то интерпретатор PHP преобразует полученный результат в число с плавающей точкой, чтобы результат вычислений оказался в конечном итоге правильным.

Переменные

В переменных хранятся данные, которыми программа манипулирует по ходу своего выполнения. Это может быть информация, извлекаемая из базы данных, или данные, введенные в заполняемой HTML-форме. В языке PHP переменные обозначаются знаком **$**, вслед за которым указывается имя переменной. Для присваивания переменной значения служит знак равенства (**=**), обозначающий операцию присваивания.

Ниже приведены некоторые примеры применения данной операции.

```
$plates = 5;
$dinner = 'Beef Chow-Fun';
$cost_of_dinner = 8.95;
$cost_of_lunch = $cost_of_dinner;
```

Присваивание распространяется и на встроенные документы, как показано ниже.

```
$page_header = <<<HTML_HEADER <html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
HTML_HEADER;

$page_footer = <<<HTML_FOOTER
</body>
</html>
HTML_FOOTER;
```

В именах переменных можно указывать лишь следующее.

- Прописные или строчные буквы основного латинского алфавита (**A–Z** и **a–z**).
- Цифры (**0–9**).
- Знак подчеркивания (**_**).
- Любой символ, не относящийся к основному латинскому алфавиту, если в исполняемом файле используется такая кодировка, как UTF-8.

Кроме того, первым символом в имени переменной нельзя указывать цифру. В табл. 2.2 перечислены некоторые примеры допустимых имен переменных.

Таблица 2.2. Допустимые имена переменных

`$size`
`$drinkSize`
`$SUPER_BIG_DRINK`
`$_d_r_i_n_k_y`
`$drink4you2`
`$напиток`
$🄳🅁🄸🄽🄺
$☺

Следует, однако, иметь в виду, что в именах переменных, применяемых в большей части кода PHP, чаще всего употребляются цифры, знаки подчеркивания и буквы основного латинского алфавита. В табл. 2.3 приведены некоторые примеры недопустимых имен переменных.

Таблица 2.3. Недопустимые имена переменных

Имя переменной	Изъян
$2hot4u	Начинается с цифры
$drink-size	Недопустимый символ: **-**
$drinkmaster@example.com	Недопустимые символы: **@** и **.**
$drink!lots	Недопустимый символ: **!**
$drink+dinner	Недопустимый символ: **+**

В именах переменных учитывается регистр. Это означает, что переменные `$dinner`, `$Dinner` и `$DINNER` оказываются совершенно разными. У них столько же общего, сколько и у переменных `$breakfast`, `$lunch` и `$supper`. На практике следует избегать употребления переменных, имена которых отличаются только регистром букв, поскольку они затрудняют чтение исходного кода и отладку программ.

Выполнение операций над переменными

Арифметические и строковые операции над переменными, содержащими числа и символьные строки, выполняются таким же образом, как и над числами и символьными строками. В примере 2.17 демонстрируется ряд арифметических и строковых операций над переменными.

Пример 2.17. Операции над переменными

```
$price = 3.95;
$tax_rate = 0.08;
$tax_amount = $price * $tax_rate;
$total_cost = $price + $tax_amount;

$username = 'james';
$domain = '@example.com';
$email_address = $username . $domain;


print 'The tax is ' . $tax_amount;
print "\n"; // вывести разрыв строки
print 'The total cost is ' .$total_cost;
print "\n"; // вывести разрыв строки
print $email_address;
```

При выполнении кода из примера 2.17 на экран выводится следующий результат:

```
The tax is 0.316
The total cost is 4.266
james@example.com
```

Операцию присваивания можно сочетать с арифметическими и строковыми операциями, чтобы сократить порядок модификации переменной. Знак равенства, указываемый после операции, означает присваивание результата этой операции заданной переменной. В примере 2.18 показаны два способа сложить одно и то же число 3 со значением переменной `$price`.

Пример 2.18. Сочетание операции присваивания с арифметической операцией

```
// прибавить число 3 обычным способом
$price = $price + 3;
// прибавить число 3 с помощью составной операции
$price += 3;
```

Благодаря сочетанию операции присваивания с операцией сцепления символьных строк заданное значение присоединяется к символьной строке. В примере 2.19 демонстрируются два способа присоединения суффикса к символьной строке. Преимущество составных операций состоит в их краткости.

Пример 2.19. Сочетание операций присваивания и сцепления символьных строк

```
$username = 'james';
$domain = '@example.com';
// присоединить символьную строку из переменной $domain в
// конце строки из переменной $username обычным способом
$username = $username . $domain;
// сцепить символьные строки с помощью составной операции
$username .= $domain;
```

Инкрементирование (т.е. приращение) и декрементирование (т.е. отрицательное приращение) переменных на **1** настолько распространено, что для этой цели предназначены отдельные операции. В частности, операция инкрементирования **++** прибавляет **1** к значению переменной, а операция

декрементирования `--` вычитает **1** из значения переменной. Эти операции обычно применяются в циклах `for()`, подробнее рассматриваемых в главе 3. Но их можно выполнять над любой переменной, хранящей число (пример 2.20).

Пример 2.20. Инкрементирование и декрементирование

```
// прибавить 1 к значению переменной $birthday
$birthday = $birthday + 1;
// прибавить еще одну 1 к значению переменной $birthday
++$birthday;
// вычесть 1 из значения переменной $years_left
$years_left = $years_left - 1;
// вычесть еще одну 1 из значения переменной $years_left
--$years_left;
```

Вставка переменных в символьные строки

Нередко значения переменных выводятся на экран вместе с другим текстом, например, при отображении HTML-таблицы с вычисленными значениями в ее ячейках или страницы профиля пользователя, где конкретные сведения о пользователе представлены по стандартному шаблону HTML-разметки. Такой вывод упрощается благодаря возможности *вставлять* переменные в символьные строки, заключенные в двойные кавычки, а также во встраиваемые документы. Так, в примере 2.21 значение переменной `$email` вставляется в символьную строку, выводимую на экран.

Пример 2.21. Вставка переменной

```
$email = 'jacob@example.com';
print "Send replies to: $email";
```

При выполнении кода из примера 2.21 на экран выводится следующий результат:

```
Send replies to: jacob@example.com
```

Встраиваемые документы особенно удобны для вставки многих переменных в длинный блок HTML-разметки, как показано в примере 2.22.

```
$page_title = 'Menu';
$meat = 'pork';
$vegetable = 'bean sprout';
print <<<MENU
<html>
<head><title>$page_title</title></head>
<body>
<ul>
<li> Barbecued $meat
<li> Sliced $meat
<li> Braised $meat with $vegetable </ul>
</body>
</html>
MENU;
```

При выполнении кода из примера 2.22 на экран выводится следующий результат:

```
<html>
<head><title>Menu</title></head>
<body>
<ul>
<li> Barbecued pork
<li> Sliced pork
<li> Braised pork with bean sprout </ul>
</body>
</html>
```

Встраиваемые и актуальные документы

В версии PHP 5.3 в дополнение к встраиваемому документу появился так называемый актуальный документ. Если заключить начальный ограничитель в одиночные кавычки, то вместо встраиваемого документа получится *актуальный документ*. В отличие от встраиваемого документа, в актуальный нельзя вставлять переменные. Если встраиваемый документ можно представить, как многострочный текст в двойных кавычках, то актуальный документ — как многострочный текст в одиночных кавычках.

Когда переменная вставляется в таком месте символьной строки, где интерпретатор PHP может испытывать трудности, в определении имени переменной, такую переменную следует заключить в фигурные скобки, чтобы устранить недоразумение. Так, в коде из примера 2.23 фигурные скобки необходимы для правильной интерпретации переменной `$preparation`, вставляемой в выводимую на экран символьную строку.

Пример 2.23. Вставка переменной в фигурных скобках

```
$preparation = 'Braise';
$meat = 'Beef';
print "$preparationd $meat with Vegetables";
```

При выполнении кода из примера 2.23 на экран выводится следующий результат:

```
Braised Beef with Vegetables
```

В отсутствие фигурных скобок последняя строка кода из примера 2.23 выглядела бы следующим образом:

```
print "$preparationd $meat with Vegetables";
```

Похоже, что в данном случае в символьную строку вставлена переменная `$preparationd`. Поэтому фигурные скобки требуются для того, чтобы явно указать, где оканчивается имя переменной и начинается символьная строка. Синтаксис фигурных скобок также удобен для вставки более сложных выражений и значений элементов массивов, как поясняется в главе 4.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Определение символьных строк в программах на PHP тремя способами: в одиночных, в двойных кавычках и в виде встраиваемого документа.

- Назначение экранирования и символы, которые следует экранировать в отдельных видах символьных строк.
- Проверка достоверности символьной строки путем определения ее длины, удаления начальных и конечных пробелов и сравнения с другой символьной строкой.
- Форматирование символьной строки с помощью функции `printf()`.
- Манипулирование регистром букв в символьной строке с помощью функций `strtolower()`, `strtoupper()` и `ucwords()`.
- Извлечение подстроки с помощью функции `substr()`.
- Изменение части символьной строки с помощью функции `str_replace()`.
- Определение чисел в программах.
- Выполнение арифметических операций над числами.
- Сохранение значений в переменных.
- Надлежащее именование переменных.
- Выполнение составных операций над переменными.
- Выполнение операций инкрементирования и декрементирования над переменными.
- Вставка переменных в символьные строки.

Упражнения

1. Найдите ошибки в приведенном ниже исходном тексте программы на PHP.

```
<? php
print 'How are you?';
print 'I'm fine.';
??>
```

2. Напишите на PHP программу, вычисляющую общую стоимость трапезы в ресторане, состоящей из двух гамбургеров по 4,95 доллара каждый, одного молочно-шоколадного коктейля за 1,95 доллара и одной порции кока-колы за 0,85 доллара. Ставка налога на добавленную стоимость составляет 7,5%, а чаевые без вычета налогов — 16%.
3. Видоизмените программу из предыдущего упражнения, чтобы вывести счет в отформатированном виде. В частности, выведите сначала цену и количество каждого блюда вместе с общей стоимостью трапезы, а затем общую стоимость еды и напитков как без учета, так и с учетом налога на добавленную стоимость и чаевых. Непременно выровняйте цены в выводимом счете по вертикали.
4. Напишите на PHP программу, задающую имя в переменной `$first_name` и фамилию в переменной `$last_name`. Выведите символьную строку, содержащую имя и фамилию, разделив их пробелом. Кроме того, выведите длину этой символьной строки.
5. Напишите на PHP программу, в которой применяются составные операции инкрементирования (**++**) и умножения с присваиванием (***=**), для вывода чисел в пределах от **1** до **5**, а также степеней числа **2** в пределах от **2** (**2**^{1}) до **32** (**2**^{5}).

6. Снабдите комментариями программы, написанные на PHP в предыдущих упражнениях. Попробуйте ввести как одно-, так и многострочные комментарии. После ввода комментариев выполните программу, чтобы убедиться в правильности их функционирования и синтаксиса введенных комментариев.

ГЛАВА 3

Управляющая логика для принятия решений и повторения операций

В главе 2 были рассмотрены основы представления данных в программах на PHP. Но наполнить программу данными — это лишь полдела. Нужно еще умело воспользоваться данными, чтобы управлять ходом выполнения программы, предприняв, например, следующие действия.

- Вывести специальное меню, если в систему входит административный пользователь.
- Вывести другой заголовок страницы, если уже больше трех часов дня.
- Уведомить пользователя, если с момента его последнего входа в систему отправлены новые сообщения.

Для всех этих действий характерна следующая общая особенность: в них принимаются решения относительно истинности или ложности определенного логического условия, включающего данные. Так, в первом действии проверяется логическое условие входа административного пользователя в систему. Если это условие истинно (т.е. административный пользователь вошел в систему), то выводится специальное меню. То же самое происходит и в следующем примере. Так, если истинно условие, что уже перевалило за три часа дня, то выводится другой заголовок страницы. А если в последнем примере отправлены новые сообщения с момента последнего входа пользователя в систему, то он уведомляется об этом.

Когда в интерпретаторе PHP принимается решение, выражение с проверяемым условием сводится к логическому значению `true` (истинно) или `false` (ложно). О том, каким образом винтерпретаторе PHP принимается решение об истинности или ложности выражений и значений, речь пойдет далее, в разделе “Общее представление об истинности или ложности”.

Логические значения `true` и `false` применяются в таких языковых конструкциях, как `if()`, чтобы решить, следует ли выполнять определенные операторы в программе. Особенности конструкции `if()` рассматриваются далее, в разделе “Принятие решений”. Этой и аналогичными языковыми конструкциями следует пользоваться всякий раз, когда результат выполнения программы зависит от каких-нибудь изменяющихся условий.

Несмотря на то что логические значения `true` и `false` ставятся во главу угла при принятии решений, в программах на PHP приходится искать ответы на более сложные вопросы вроде следующих: насчитывает ли возраст пользователя хотя бы 21 год, или имеется ли у пользователя ежемесячная подписка на веб-сайт и достаточно ли средств на его счету, чтобы приобрести ежедневный пропуск? Ниже, в разделе “Принятие сложных решений”, поясняются доступные в PHP операции сравнения и логические операции. Эти операции помогают выразить решения, которые приходится принимать в программе, например, для того, чтобы выяснить, являются ли одни числа или символьные строки больше или меньше, чем другие. Из отдельных решений можно также составлять более сложные решения, зависящие от их составляющих.

Решения принимаются в программах и в том случае, если требуется повторно выполнить определенный ряд операторов, а также указать, когда это повторение следует прекратить. Зачастую такое условие определяется простым подсчетом, например, повторить операцию 10 раз. Это все равно, что спросить, не было ли действие повторено уже 10 раз. Если это именно так, то выполнение программы продолжается дальше, а иначе действие повторяется. Определить, когда следует остановить повторяющееся действие (например, задать учащемуся очередной вопрос по математике лишь после того, как он правильно ответит на шесть предыдущих вопросов), возможно, будет сложнее. Для реализации такого рода повторений в циклах в PHP имеются языковые конструкции `while()` и `for()`, рассматриваемые в разделе “Повторение операций”.

Общее представление об истинности или ложности

Всякое выражение в PHP имеет истинное (`true`) или ложное (`false`) логическое значение. В одних случаях это истинное значение оказывается важным потому, что оно используется в вычислении, а в других случаях оно просто игнорируется. Представление о том, как в выражениях вычисляются логические значения `true` и `false`, является важной составляющей понимания языка PHP.

Большинство скалярных значений являются истинными. Все числа с плавающей точкой и целые числа, кроме **0** и **0.0**, а также все символьные строки, кроме пустой или содержащей символ **0**, являются истинными. Специальные константы **false** и **null** вычисляются как ложные, а значения всех остальных констант — как истинные[1].

Переменная, равная одному из ложных значений, или функция, возвращающая одно из этих значений, также вычисляется как ложная. А любое другое выражение вычисляется как истинное.

Выяснение истинного значения вычисляемого выражения происходит в два этапа. Сначала выясняется конкретное значение выражения, а затем проверяется, является ли значение истинным (`true`) или ложным (`false`). Некоторые выражения имеют совершенно практическое значение. Значение математического выражения получается в результате его вычисления на бумаге с карандашом в руках. Например, результат вычисления выражения **7 * 6** равен **42**. А поскольку значение **42** является истинным, то истинным оказывается и выражение **7 * 6**. Результат вычисления выражения **5 - 6 + 1** равен **0**. А поскольку значение **0** является ложным, то ложным будет и выражение **5 - 6 + 1**.

Это же относится и к сцеплению символьных строк. Значением выражения, где сцепляются две символьные строки, является новая соединенная строка. Так, выражение `'jacob'.'@example.com'` имеет строковое значение `jacob@example.com`, которое является истинным.

Операция присваивания имеет присваиваемое значение. Так, результат вычисления выражения `$price = 5` равен **5**, поскольку именно это числовое значение присваивается переменной `$price`. Присваивание имеет вполне определенный результат, и поэтому операции присваивания можно связать в цепочку, чтобы присвоить одно и то же значение сразу нескольким переменным:

```
$price = $quantity = 5;
```

Приведенное выше выражение означает, что переменной `$price` присваивается результат присваивания числового значения **5** переменной `$quantity`. При вычислении этого выражения целочисленное значение **5** сначала присваивается переменной `$quantity`, а результат этого присваивания (т.е. число **5**) затем присваивается переменной `$price`. В итоге целочисленное значение **5** устанавливается в обеих переменных — `$price` и `$quantity`.

Принятие решений

С помощью языковой конструкции `if()` в программе можно выполнить ряд операторов только в том случае, если определенные условия оказываются истинными. Это дает возможность пред-

[1] Пустой массив также вычисляется как ложный. Подробнее об этом — в главе 4.

принимать разные действия в программе, исходя из конкретных обстоятельств. Например, можно проверить, ввел ли пользователь достоверную информацию в веб-форме, прежде чем показывать ему конфиденциальную информацию.

Блок кода выполняется в языковой конструкции `if()` лишь в том случае, если проверочное выражение оказывается истинным (`true`). Эта особенность данной конструкции наглядно демонстрируется в примере 3.1.

Пример 3.1. Принятие решения с помощью языковой конструкции `if()`

```
if ($logged_in) {
  print "Welcome aboard, trusted user.";
}
```

В языковой конструкции `if()` обнаруживается истинное значение проверочного выражения, указываемого в круглых скобках, следующим образом: (*проверочное выражение*). Если вычисляется истинное значение проверочного выражения, то выполнение программы продолжается с оператора, следующего после фигурных скобок. В данном случае проверочное выражение составляет единственная переменная `$logged_in`. Если значение переменной `$logged_in` равно `true` (или же она имеет значение, вычисляемое как истинное, например, **`5`**, **`-12.6`** или **`Grass Carp`**), то на экран выводится символьная строка `"Welcome aboard, trusted user."` (Добро пожаловать, доверенный пользователь.).

В блоке кода, заключаемом в фигурные скобки, можно разместить сколько угодно операторов. Но каждый из них должен быть завершен точкой с запятой. Это же правило распространяется и на код, расположенный вне условного оператора `if()`. Но указывать точку с запятой после фигурной скобки, закрывающей блок кода, совсем не обязательно, как, впрочем, и после открывающей фигурной скобки. В примере 3.2 демонстрируется языковая конструкция `if()`, в которой выполняется несколько операторов, если проверочное выражение оказывается истинным.

Пример 3.2. Выполнение ряда операторов в блоке кода языковой конструкции `if()`

```
print "This is always printed.";
if ($logged_in) {
    print "Welcome aboard, trusted user.";
    print 'This is only printed if $logged_in is true.';
}
print "This is also always printed.";
```

Чтобы выполнить другие операторы, если проверочное выражение в языковой конструкции `if()` оказывается ложным, оператор `if()` следует дополнить предложением `else`, как показано в примере 3.3.

Пример 3.3. Применение предложения `else` в условном операторе `if()`

```
if ($logged_in) {
    print "Welcome aboard, trusted user.";
} else {
    print "Howdy, stranger.";
}
```

В коде из примера 3.3 первый оператор `print` выполняется лишь в том случае, если проверочное выражение, состоящее из единственной переменной `$logged_in` в языковой конструкции `if()`, окажется истинным. А второй оператор `print` выполняется в предложении `else` только в том случае, если проверочное выражение оказывается ложным.

Языковые конструкции `if()` и `else` расширяются конструкцией `elseif()`. С языковой конструкцией `if()` можно сочетать одну или несколько конструкций `elseif()` для проверки нескольких условий в отдельности. Применение языковой конструкции `elseif()` демонстрируется в примере 3.4.

Пример 3.4. Применение языковой конструкции `elseif()`

```
if ($logged_in) {
    // Следующая строка кода выполняется, если проверочное
    // условие $logged_in истинно
    print "Welcome aboard, trusted user.";
} elseif ($new_messages) {
    // Следующая строка кода выполняется, если проверочное
    // условие $logged_in ложно, но проверочное условие
    // $new_messages истинно
    print "Dear stranger, there are new messages.";
} elseif ($emergency) {
    // Следующая строка кода выполняется, если оба
    // проверочных условия, $logged_in и $new_messages, ложны,
    // но проверочное условие $emergency истинно
    print "Stranger, there are no new messages,
           but there is an emergency.";
}
```

Если проверочное выражение в условном операторе `if()` истинно, интерпретатор PHP выполняет операторы в блоке кода, следующем после условного оператора `if()`, игнорируя условные операторы `elseif()` и их блоки кода. Если проверочное выражение в условном операторе `if()` ложно, то интерпретатор PHP переходит к первому условному оператору `elseif()`, применяя ту же самую логику. Если же проверочное выражение в первом условном операторе `elseif()` истинно, то выполняется блок кода, следующий после этого оператора. А если данное выражение ложно, то интерпретатор PHP переходит к следующему условному оператору `elseif()`.

Для заданного ряда условных операторов `if()` и `elseif()` выполняется хотя бы один блок кода, т.е. блок кода первого условного оператора, проверочное условие которого истинно. Если же проверочное выражение условного оператора `if()` оказывается истинным, то блок кода ни одного из условных операторов `elseif()` вообще не выполняется, даже если их проверочное условие истинно. Как только проверочное выражение условного оператора `if()` или `elseif()` окажется истинным, остальные условные операторы игнорируются. Если же ни одно из проверочных условий в условных операторах `if()` и `elseif()` не оказывается истинным, то ни один из блоков кода не выполняется.

Условный оператор `else` можно сочетать с условным оператором `elseif()`, чтобы выполнить блок кода в том случае, если проверочные выражения ни одного из условных операторов `if()` или `elseif()` не оказываются истинными. В примере 3.5 условный оператор `else` вводится в код из примера 3.4.

Пример 3.5. Сочетание условных операторов else и elseif()

```
if ($logged_in) {
    // Следующая строка кода выполняется, если проверочное
    // условие $logged_in истинно
    print "Welcome aboard, trusted user.";
} elseif ($new_messages) {
    // Следующая строка кода выполняется, если проверочное
    // условие $logged_in ложно, но проверочное условие
```

```
    // $new_messages истинно
    print "Dear stranger, there are new messages.";
} elseif ($emergency) {
    // Следующая строка кода выполняется, если оба
    // проверочных условия, $logged_in и $new_messages, ложны,
    // но проверочное условие $emergency истинно
    print "Stranger, there are no new messages,
           but there is an emergency.";
} else {
    // Следующая строка кода выполняется, если все проверочные
    // условия, $logged_in, $new_messages и $emergency, ложны
    print "I don't know you, you have no messages,
           and there's no emergency.";
}
```

Все упоминавшиеся ранее блоки кода были заключены в фигурные скобки. Строго говоря, заключать блок кода в фигурные скобки совсем не обязательно, если он состоит из единственного оператора. Но даже если заключить такой блок кода в фигурные скобки, то он все равно будет выполнен правильно. Заключение блока кода любой величины в фигурные скобки способствует повышению удобочитаемости исходного кода, поэтому это следует делать всегда. Если интерпретатору PHP все равно, то читающий исходный код по достоинству оценит наличие фигурных скобок, которые обеспечивают ясность кода.

Принятие сложных решений

Операции сравнения и логические операции в PHP помогают составлять более сложные проверочные выражения для принятия решений в языковой конструкции `if()`. Эти операции позволяют сравнивать и отрицать значения, связывать в цепочку несколько выражений в одном условном операторе `if()`.

Операция сравнения на равенство обозначается двумя знаками равенства (**==**). В результате выполнения этой операции возвращается логическое значение `true`, если два сравниваемых значения равны. Сравниваемыми значениями могут быть переменные или литералы. Некоторые образцы применения операции сравнения на равенство приведены в примере 3.6.

Пример 3.6. Операция сравнения на равенство

```
if ($new_messages == 10) {
    print "You have ten new messages.";
}
if ($new_messages == $max_messages) {
    print "You have the maximum number of messages.";
}
if ($dinner == 'Braised Scallops') {
    print "Yum! I love seafood.";
}
```

Противоположной по отношению к операции сравнения на равенство является операция сравнения на неравенство, обозначаемая знаками **!=**. В результате выполнения этой операции возвращается логическое значение `true`, если сравниваемые значения не равны, как показано в примере 3.7.

Пример 3.7. Операция сравнения на неравенство

```
if ($new_messages != 10) {
    print "You don't have ten new messages.";
}
if ($dinner != 'Braised Scallops') {
    print "I guess we're out of scallops.";
}
```

С помощью операций сравнения на "меньше" (**<**) и "больше" (**>**) можно сравнивать разные величины. Аналогичное назначение имеют операции сравнения на "меньше или равно" (**<=**) и "больше или равно" (**>=**). Образцы применения этих операций сравнения демонстрируются в примере 3.8.

Пример 3.8. Операции сравнения на "меньше или равно" и "больше или равно"

```
if ($age > 17) {
    print "You are old enough to download the movie.";
}
if ($age >= 65) {
    print "You are old enough for a discount.";
}
if ($celsius_temp <= 0) {
    print "Uh-oh, your pipes may freeze.";
}
if ($kelvin_temp < 20.3) {
    print "Your hydrogen is a liquid or a solid now.";
}
```

Как уже упоминалось в разделе "Числа" главы 2, числа с плавающей точкой хранятся таким образом, что их внутреннее представление может несколько отличаться от их присваиваемых значений. Например, внутреннее представление сохраняемого числа **50.0** может быть равно **50.00000002**. Чтобы проверить на равенство два числа с плавающей точкой, следует сначала проверить, отличаются ли оба числа меньше, чем на допустимую пороговую величину, вместо того, чтобы выполнять операцию сравнения. Так, если сравниваются денежные суммы, то допустимой для них может быть пороговая величина **0.00001**. В примере 3.9 показано, каким образом сравниваются два числа с плавающей точкой.

Пример 3.9. Сравнение чисел с плавающей точкой

```
if(abs($price_1 - $price_2) < 0.00001) {
    print '$price_1 and $price_2 are equal.';
} else {
    print '$price_1 and $price_2 are not equal.';
}
```

Функция `abs()`, применяемая в примере 3.9, возвращает абсолютное значение своего аргумента. Благодаря функции `abs()` сравнение выполняется надлежащим образом, если значение переменной `$price_1` больше значения переменной `$price_2` или, наоборот, значение переменной `$price_2` больше значения переменной `$price_1`.

Операции сравнения на "меньше" и "больше", а также их аналоги с дополнительной проверкой на равенство можно употреблять для сравнения чисел и символьных строк. Как правило, символьные строки сравниваются так, как будто они находятся в словаре. Символьная строка, обнаруживаемая раньше в словаре, оказывается меньше, чем символьная, обнаруживаемая позже в словаре. Некоторые образцы сравнения символьных строк приведены в примере 3.10.

Присваивание и сравнение

Будьте внимательны, пользуясь знаком **=**, обозначающим операцию присваивания, на самом деле имея в виду операцию сравнения на равенство, обозначаемую знаком **==**. Два знака равенства обозначают проверку на равенство и возврат логического значения **true**, если сравниваемые значения равны. Если же отбросить второй знак равенства, то проверочное выражение в условном операторе **if()** всегда будет истинным, как показано ниже.

```
if ($new_messages =12) {
    print "It seems you now have twelve new messages.";
}
```

Вместо проверки переменной **$new_messages** на равенство значению **12** в приведенном выше фрагменте кода этой переменной присваивается значение **12**. В результате такого присваивания возвращается значение **12**. Таким образом, проверочное выражение всегда будет истинным независимо от значения переменной **$new_messages**. Кроме того, значение переменной **$new_messages** перезаписывается. Чтобы избежать случайного употребления операции **=** вместо операции **==**, рекомендуется разместить переменную в правой части операции сравнения, а литерал — в левой ее части:

```
if (12 == $new_messages) {
    print "It seems you now have twelve new messages.";
}
```

Приведенное выше проверочное выражение выглядит не совсем обычно, что гарантирует в какой-то степени от случайного употребления операции **=** вместо **==**. Если в этом выражении указан один знак равенства, то переменной **$new_messages** присваивается значение **12**. Но в этом нет никакого смысла, поскольку изменить значение **12** нельзя. Если выражение **12 = $new_messages** будет обнаружено в программе интерпретатором PHP, последний выдаст сообщение о синтаксической ошибке, а, следовательно, программа не сможет быть выполнена. Синтаксическая ошибка предупреждает об отсутствии еще одного знака **=** в проверочном выражении. А если бы литерал был указан в правой части проверочного выражения, то синтаксический анализ программы прошел бы успешно и сообщение об ошибке не появилось бы.

Пример 3.10. Сравнение символьных строк

```
if ($word < 'baa') {
    print "Your word isn't cookie.";
}
if ($word >= 'zoo') {
    print "Your word could be zoo or zymurgy, but not zone.";
}
```

Но сравнение символьных строк может привести к неожиданным результатам, если символьные строки содержат только числа или начинаются с чисел. Когда интерпретатор PHP обнаружит подобные символьные строки, он преобразует их в числа для сравнения. Такое автоматическое преобразование демонстрируется в примере 3.11.

Пример 3.11. Сравнение чисел и символьных строк

```
// Следующие значения сравниваются в лексикографическом порядке
if ("x54321"> "x5678") {
    print 'The string "x54321" is greater than
           the string "x5678".';
```

```
} else {
    print 'The string "x54321" is not greater than
           the string "x5678".';
}
// Следующие значения сравниваются в числовом порядке
if ("54321" > "5678") {
    print 'The string "54321" is greater than
           the string "5678".';
} else {
    print 'The string "54321" is not greater than
           the string "5678".';
}
// Следующие значения сравниваются в лексикографическом порядке
if ('6 pack' < '55 card stud') {
    print 'The string "6 pack" is less than
           the string "55 card stud".';
} else {
    print 'The string "6 pack" is not less than
           the string "55 card stud".';
}
// Следующие значения сравниваются в числовом порядке
if ('6 pack' < 55) {
    print 'The string "6 pack" is less than the number 55.';
} else {
    print 'The string "6 pack" is not less than the number 55.';
}
```

При выполнении кода из примера 3.11 на экран выводится следующий результат:

```
The string "x54321" is not greater than the string "x5678".
The string "54321" is greater than the string "5678".
The string "6 pack" is not less than the string "55 card stud".
The string "6 pack" is less than the number 55.
```

При первой проверке обе символьные строки интерпретируются как обычные строки, поскольку они начинаются с буквы, а следовательно, сравниваются в лексикографическом порядке. Их первые символы (**x5**) одинаковы, но третий символ первого слова (**4**) меньше третьего символа второго слова (**6**),[2] и поэтому в результате операции сравнения на “больше” возвращается логическое значение `false`. При второй проверке каждая символьная строка состоит только из цифр, и поэтому строки сравниваются как числа. В частности, число **54321** больше числа **5678**, поэтому в результате операции сравнения на “больше” возвращается логическое значение `true`. А при третьей проверке сравниваемые символьные строки интерпретируются и сравниваются в лексикографическом порядке, поскольку обе строки состоят из числовых и прочих символов. Цифра **6** следует после цифры **5** в словаре интерпретатора PHP, и поэтому в результате операции сравнения на “меньше” возвращается логическое значение `false`. И, наконец, при последней проверке интерпретатор PHP преобразует сначала символьную строку **"6 pack"** в число **6**, а затем сравнивает его с числом **55** в числовом порядке. А поскольку число **6** меньше числа **55**, в результате операции сравнения на “меньше” возвращается логическое значение `true`.

Если требуется, чтобы интерпретатор PHP сравнивал символьные строки в лексикографическом порядке, не преобразуя числа внутренним образом, воспользуйтесь функцией `strcmp()`. Она всегда сравнивается свои аргументы в лексикографическом порядке.

[2] “Словарь”, употребляемый интерпретатором PHP для сравнения символьных строк, содержит символы в коде ASCII, где числа расположены перед буквами в пределах от **0** до **9**, а прописные буквы — перед строчными.

Сравнение символьных строк не в коде ASCII

Напомним, что символьные строки в PHP представлены последовательностями байтов. При сравнении строк, символы которых отсутствуют в обычном словаре английского языка, обычные операции и функции сравнения могут дать результаты, отличающиеся от предполагаемых. В разделе "Сортировка и сравнение" главы 20 рассматривается класс **`Collator`**, позволяющий сравнивать и сортировать текст в разных наборах символов.

Функция `strcmp()` принимает в качестве аргументов две символьные строки. Она возвращает положительное число, если первая строка больше второй, или отрицательное число, если первая строка меньше второй. Условие "больше" или "меньше" определяется для функции `strcmp()` в лексикографическом порядке. Если обе символьные строки равны, эта функция возвращает нулевое значение. Те же самые операции сравнения, что и в примере 3.11, выполняются в примере 3.12 с помощью функции `strcmp()`.

Пример 3.12. Сравнение символьных строк с помощью функции `strcmp()`

```
$x = strcmp("x54321","x5678");
if ($x > 0) {
    print 'The string "x54321" is greater than the string "x5678".';
} elseif ($x < 0) {
    print 'The string "x54321" is less than the string "x5678".';
}

$x = strcmp("54321","5678");
if ($x > 0) {
    print 'The string "54321" is greater than the string "5678".';
} elseif ($x < 0) {
    print 'The string "54321" is less than the string "5678".';
}

$x = strcmp('6 pack','55 card stud');
if ($x > 0) {
    print 'The string "6 pack" is greater than
           the string "55 card stud".';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than
           the string "55 card stud".';
}

$x = strcmp('6 pack',55);
if ($x > 0) {
    print 'The string "6 pack" is greater than the number 55.';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than the number 55.';
}
```

При выполнении кода из примера 3.12 на экран выводится следующий результат:

```
The string "x54321" is less than the string "x5678".
The string "54321" is less than the string "5678".
The string "6 pack" is greater than the string "55 card stud".
The string "6 pack" is greater than the number 55.
```

Применение функции `strcmp()` и лексикографического порядка дает во второй и четвертой операциях сравнения иные результаты, чем в примере 3.11. Так, во второй операции сравнения функция `strcmp()` определяет, что символьная строка **"54321"** меньше, чем строка **"5678"**, поскольку вторые символы в этих строках отличаются, а цифра **4** предшествует цифре **6**. Для функции `strcmp()` не имеет значения, что символьная строка **"5678"** короче строки **"54321"** или что она численно меньше. В лексикографическом порядке символьная строка **"54321"** предшествует строке **"5678"**. А результат четвертой операции сравнения оказывается иным потому, что функция `strcmp()` не преобразует символьную строку **"6 pack"** в число. Вместо этого она сравнивает символьные строки **"6 pack"** и **"55"** и определяет, что первая из них больше другой, поскольку ее первый символ (**6**) следует в словаре после первого символа (5) во второй строке.

Составная операция типа "космический корабль" (**<=>**) выполняет сравнение аналогично функции `strcmp()`, но она пригодна для сравнения данных любого типа. Если левый операнд меньше, чем правый, то в результате данной операции получается отрицательное число. А если левый операнд больше, чем правый, то в результате данной операции получается положительное число. Если же оба операнда равны, то возвращается нулевое значение. Действие составной операции сравнения типа "космический корабль" показано в примере 3.13.

Пример 3.13. Сравнение разных типов данных с помощью составной операции типа "космический корабль"

```
// Переменной $a присваивается отрицательное число,
// поскольку 1 меньше 12.7
$a = 1 <=> 12.7;

// Переменной $b присваивается положительное число,
// поскольку символ "c" следует после символа "b"
$b = "charlie" <=> "bob";

// Сравнение числовых символьных строк осуществляется аналогично
// операциям сравнения < и >, но не функции strcmp()
$x = '6 pack' <=> '55 card stud';
if ($x > 0) {
    print 'The string "6 pack" is greater than
           the string "55 card stud".';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than
           the string "55 card stud".';
}

// Сравнение числовых символьных строк осуществляется аналогично
// операциям сравнения < и >, но не функции strcmp()
$x ='6 pack' <=> 55;
if ($x > 0) {
    print 'The string "6 pack" is greater than the number 55.';
} elseif ($x < 0) {
    print 'The string "6 pack" is less than the number 55.';
}
```

При выполнении кода из примера 3.13 на экран выводится следующий результат:

```
The string "6 pack" is greater than the string "55 card stud".
The string "6 pack" is less than the number 55.
```

Составная операция сравнения типа “космический корабль” была внедрена в версии PHP 7. Если вы пользуетесь более ранней версией PHP, придерживайтесь других операций сравнения.

Составная операция сравнения типа космический корабль следует тем же правилам преобразования символьных строк и чисел, что и другие операции сравнения. Она преобразует “числовые” символьные строки в числа аналогично операциям сравнения **==**, **<** и пр.

Для отрицания истинного значения служит операция **!**. Указание операции **!** перед выражением равнозначно проверке выражения на равенство логическому значению `false`. Так, условные операторы `if()` в примере 3.14 равнозначны.

Пример 3.14. Применение операции отрицания

```
// Все проверочное выражение ($finished == false)
// истинно, если значение переменной $finished равно false
if ($finished == false) {
    print 'Not done yet!';
}

// Все проверочное выражение (! $finished) истинно,
// если значение переменной $finished равно false
if (! $finished) {
    print 'Not done yet!';
}
```

Операцию отрицания можно выполнять над любым значением. Если значение истинно, то результат применения к нему операции отрицания будет ложным. А если значение ложно, то результат применения к нему операции отрицания будет истинным. Применение операции отрицания при вызове функции `strcasecmp()` показано в примере 3.15.

Пример 3.15. Операция отрицания

```
if (! streaseemp($first_name, $last_name)) {
    print '$first_name and $last_name are equal.';
}
```

В примере 3.15 оператор из блока кода в условном операторе `if()` выполняется только в том случае, если все проверочное выражение истинно. Если две символьные строки, предоставляемые функции `strcasecmp()`, равны (без учета регистра букв), то функция `strcasecmp()` возвращает нулевое значение, которое является ложным. В проверочном выражении операция отрицания применяется к этому ложному значению. В результате отрицания ложного значения получается истинное значение. Поэтому все проверочное выражение оказывается истинным, когда две равные символьные строки предоставляются функции `strcasecmp()`.

Логические операции позволяют объединять несколько выражений в одном условном операторе `if()`. В логической операции И (`&&`) проверяется, истинны ли оба объединяемых выражения. А в логической операции ИЛИ (`||`) проверяется, является ли истинным хотя бы одно из объединяемых выражений. Применение этих логических операций демонстрируется в примере 3.16.

Пример 3.16. Логические операции

```
if (($age >= 13) && ($age < 65)) {
    print "You are too old for a kid's discount and too
           young for the senior's discount.";
}
if (($meal == 'breakfast') || ($dessert == 'souffle')) {
    print "Time to eat some eggs.";
}
```

Первое выражение в примере 3.16 будет истинным, если истинны оба подвыражения, т.е. значение переменной `$age` равно хотя бы **13**, но не больше **65**. А второе выражение в том же примере оказывается истинным, если истинно хотя бы одно из подвыражений, т.е. значение переменной `$meal` равно `breakfast` или значение переменной `$dessert` равно `souffle`.

Рекомендации относительно старшинства операций и употребления круглых скобок, приведенные в главе 2, распространяются и на логические операции в проверочных выражениях. Во избежание неоднозначности заключайте в круглые скобки подвыражения, составляющие более крупное проверочное выражение.

Повторение операций

Повторное выполнение операций в компьютерной программе называется *организацией циклов*. Такое часто происходит, например, в том случае, если требуется извлечь ряд строк из таблицы базы данных, вывести строки из HTML-таблицы или элементы из списка, размечаемого дескриптором `<select>` в HTML-форме. В этом разделе рассматриваются две конструкции, `while()` и `for()`, предназначенные для организации циклов. У них имеются свои отличия, но каждая требует указания двух важных свойств любого цикла; повторно исполняемого кода и момента прекращения его исполнения. В качестве исполняемого кода служит блок кода, аналогичный указываемому в фигурных скобках после языковой конструкции `if()`. Условием прекращения цикла служит логическое выражение, аналогичное проверочному выражению в языковой конструкции `if()`.

Конструкция цикла `while()` аналогична повторению условного оператора `if()`. Как и в конструкции `if()`, в конструкции цикла `while()` предоставляется проверочное выражение. Если это выражение истинно, то блок кода выполняется. Но, в отличие от конструкции `if()`, выражение в конструкции цикла `while()` проверяется снова после выполнения блока кода. Если проверочное выражение по-прежнему истинно, блок кода выполняется снова, и так продолжается до тех пор, пока данное выражение остается истинным. Но как только проверочное выражение окажется ложным, выполнение программы будет продолжено со строк кода, следующих после блока кода, образующего цикл. Нетрудно догадаться, что в блоке кода должно происходить нечто, оказывающее влияние на проверочное выражение, чтобы цикл не продолжался бесконечно.

В примере 3.17 конструкция цикла `while()` применяется для вывода на экран списка, состоящего из 10 пунктов и размечаемого дескриптором `<select>` в HTML-форме.

Пример 3.17. Вывод на экран списка, размечаемого дескриптором `<select>`, в цикле, организуемом с помощью конструкции `while()`

```
$i = 1;
print '<select name="people">';
while ($i <= 10) {
    print "<option>\$i</option>\n";
    $i++;
}
print '</select>';
```

При выполнении кода из примера 3.17 на экран выводится следующий результат:

```
<select name="people"><option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>6</option>
<option>7</option>
<option>8</option>
<option>9</option>
<option>10</option>
</select>
```

Перед выполнением цикла `while()` в исходном коде из примера 3.17 в переменной `$i` устанавливается значение **1** и выводится открывающий дескриптор `<select>`. В проверочном выражении цикла `while()` значение переменной `$i` сравнивается со значением **10**. Если значение переменной `$i` меньше или равно значению `10`, выполняются два оператора из блока кода. Сначала в данном цикле из меню, размечаемого дескриптором `<select>`, выводится дескриптор `<option>`, а затем инкрементируется значение переменной `$i`. Если не инкрементировать значение переменной `$i` в цикле `while()`, дескрипторы `<option>1</option>` будут выводиться бесконечно.

После вывода в блоке кода дескрипторов `<option>10</option>` значение переменной `$i` становится равным **11** в строке кода `$i++`, а затем вычисляется проверочное выражение `($i <= 10)`. Но поскольку оно неистинно (т.е. значение **11** не больше или не равно значению **10**), то выполнение программы продолжается после блока кода из цикла `while()` и выводится закрывающий дескриптор `</select>`.

Конструкция цикла `for()` также предоставляет возможность многократно выполнять одни и те же операторы. Так, в примере 3.18 конструкция цикла `for()` применяется для вывода на экран того же списка, размечаемого дескриптором `<select>` в HTML-форме, что и в примере 3.17.

Пример 3.18. Вывод на экран списка, размечаемого дескриптором `<select>`, в цикле, организуемом с помощью конструкции `for()`

```
print '<select name="people">';
for ($i = 1; $i <= 10; $i++) {
    print "<option>\$i</option>\n";
}
print '</select>';
```

Применять конструкцию цикла `for()` немного сложнее, чем конструкцию цикла `while()`. Вместо одного проверочного выражения в круглых скобках приходится указывать три выражения, разделяемые точками с запятой: инициализирующее, проверочное и итерационное. Но привыкнув пользоваться конструкцией цикла `for()`, можно обнаружить, что она позволяет более кратко организовать цикл с помощью просто выражаемых условий инициализации и итерации.

Первое выражение, `$i = 1`, в исходном коде из примера 3.18 называется *инициализирующим*. Оно вычисляется один раз в самом начале цикла. Именно здесь размещается код инициализации переменных или другой установочный код. Второе выражение, `$i <= 10`, в исходном коде из примера 3.18 называется *проверочным*. Оно вычисляется на каждом шаге цикла перед выполнением операторов в теле цикла. Если это выражение истинно, то выполняется тело цикла (оператор `print "<option>\$i</option>\n";` в примере 3.18). И третье выражение, `$i++`, в исходном коде из примера 3.18 называется *итерационным*. Оно выполняется после каждого шага цикла. Последовательность выполнения операторов в цикле из примера 3.18 описана ниже.

1. Инициализирующее выражение: `$i = 1;`.

2. Проверочное выражение: `$i <= 10` (истинно, значение переменной `$i` равно **1**).
3. Блок кода: `print "<option>$i</option>\n";`.
4. Итерационное выражение: `$i++;`.
5. Проверочное выражение: `$i <= 10` (истинно, значение переменной `$i` равно **2**).
6. Блок кода: `print "<option>$i</option>\n";`.
7. Итерационное выражение: `$i++;`.
8. (Цикл продолжается по мере того, как инкрементируется значение переменной `$i`).
9. Проверочное выражение: `$i <= 10` (истинно, значение переменной `$i` равно **9**).
10. Блок кода: `print "<option>$i</option>\n";`.
11. Итерационное выражение: `$i++;`.
12. Проверочное выражение: `$i <= 10` (истинно, значение переменной `$i` равно **10**).
13. Блок кода: `print "<option>$i</option>\n";`.
14. Итерационное выражение: `$i++;`.
15. Проверочное выражение: `$i <= 10` (истинно, значение переменной `$i` равно **11**).

В инициализирующем и итерационном выражениях цикла `for()` можно сочетать несколько выражений, разделяя их запятой. Как правило, это делается в том случае, если требуется изменить несколько переменных по ходу выполнения цикла. В примере 3.19 это делается по отношению к переменным `$min` и `$max`.

Пример 3.19. Применение нескольких выражений в конструкции цикла `for()`

```
print '<select name="doughnuts">';
for ($min = 1, $max = 10; $min < 50; $min += 10, $max +=10) {
    print "<option>\$min - \$max</option>\n";
}
print '</select>';
```

На каждом шаге приведенного выше цикла переменные `$min` и `$max` инкрементируются на **10**. Ниже приведен результат выполнения кода из примера 3.19.

```
<select name="doughnuts"><option>1 - 10</option>
<option>11 - 20</option>
<option>21 - 30</option>
<option>31 - 40</option>
<option>41 - 50</option>
</select>
```

Резюме

В этой главе были рассмотрены следующие вопросы.

- Вычисление выражений и определение их истинности (`true`) или ложности (`false`).
- Принятие решений с помощью условного оператора `if()`.
- Расширение условного оператора `if()` оператором `else`.
- Расширение условного оператора `if()` оператором `elseif()`.
- Размещение нескольких операторов в блоке кода языковой конструкции `if()`, `elseif()` или `else`.
- Применение операций сравнения на равенство (**==**) и неравенство (**!=**) в проверочных выражениях.
- Различение операций присваивания (**=**) и сравнения на равенство (**==**).
- Применение операций сравнения на "больше" (**>**), "меньше" (**<**), "больше или равно" (**>=**) и "меньше или равно" (**<=**) в проверочных выражениях.
- Сравнение двух чисел с плавающей точкой с помощью функции `abs()`.
- Сравнение двух символьных строк с помощью соответствующих операций.
- Сравнение двух символьных строк с помощью функции `strcmp()` или `strcasecmp()`.
- Сравнение двух значений с помощью составной операции сравнения типа "космический корабль" (**<=>**).
- Применение операции отрицания (**!**) в проверочных выражениях.
- Применение логических операций (`&&` и `||`) для составления сложных проверочных выражений.
- Повторное выполнение блока кода с помощью конструкции цикла `while()`.
- Повторное выполнение блока кода с помощью конструкции цикла `for()`.

Упражнения

1. Определите истинность или ложность приведенных ниже выражений, не прибегая к помощи интерпретатора PHP.

```
a. 100.00 - 100
b. "zero"
c. "false"
d. 0 + "true"
e. 0.000
f. "0.0"
g. strcmp("false","False")
h. 0 <=> "0"
```

2. Выясните результат, выводимый приведенной ниже программой, не прибегая к помощи интерпретатора PHP.

```
$age = 12;
$shoe_size = 13;
if ($age > $shoe_size)
    print "Message 1.";
 elseif (($shoe_size++) && ($age > 20))
    print "Message 2.";
 else
    print "Message 3.";

print "Age: $age. Shoe Size: $shoe_size";
```

3. Воспользуйтесь конструкцией цикла `while()`, чтобы вывести на экран величины температур в пределах от **–50** до **50** градусов по Фаренгейту и эквивалентные им величины температур в градусах Цельсия. По температурной шкале Фаренгейта вода замерзает при температуре **23** градуса и закипает при **212** градусах. А по температурной шкале Цельсия вода замерзает при температуре **0** градусов и закипает при **100** градусах. Таким образом, для преобразования температуры по Фаренгейту в температуру по Цельсию следует вычесть из ее величины **32**, умножить полученную разность на **5** и разделить на **9**. А для преобразования температуры по Цельсию в температуру по Фаренгейту следует умножить ее величину на **9**, разделить полученный результат на **5** и прибавить **32**.

4. Видоизмените выполнение задания в упражнении 3, воспользовавшись конструкцией цикла `for()` вместо конструкции цикла `while()`.

ГЛАВА 4

Группирование и обработка данных в массивах

Массивы являются коллекциями связанных вместе значений, например, таких данных, передаваемых на обработку из заполняемой формы, как имена учащихся в классе или численность населения в перечне городов. Как пояснялось в главе 2, переменная является именованным контейнером, содержащим значение. А массив является контейнером, содержащим многие значения.

В этой главе поясняется, каким образом следует оперировать массивами. В следующем далее разделе "Основы организации массивов", с которого начинается данная глава, представлены такие основные положения о массивах, как их создание и манипулирование их элементами. Зачастую над каждым элементом массива требуется выполнить какое-нибудь действие, например, вывести его значение на экран или проанализировать его для проверки определенных условий. Ниже, в разделе "Перебор массивов" поясняется, как обрабатывать массивы с помощью языковых конструкций `foreach()` и `for()`. А в разделе "Модификация массивов" представлены функции `implode()` и `explode()`, преобразующие массивы в символьные строки, и наоборот. Еще одним способом модификации массивов является сортировка, обсуждаемая в разделе "Сортировка массивов". И, наконец, в разделе "Применение многомерных массивов" исследуются массивы, содержащие другие массивы.

Манипулирование массивами весьма характерно для программирования на PHP. В главе 7 будет показано, каким образом организуется обработка данных из заполняемой формы, которые интерпретатор PHP автоматически размещает в массиве. При извлечении информации из базы данных, как поясняется в главе 8, полученные в итоге данные зачастую упаковываются в массив. Умение обращаться с массивами упрощает манипулирование этими совокупностями данных.

Основы организации массивов

Массив состоит из *элементов*. У каждого элемента массива имеются свои *ключ* и *значение*. Например, в массиве, хранящем информацию об окраске овощей, имеются наименования овощей для ключей и цвета для значений (рис. 4.1).

Массив может иметь только один элемент с заданным ключом. Так, в массиве окраски овощей не должно быть еще одного элемента с ключом `corn` (кукуруза), даже если его значение равно `blue` (голубая окраска). Тем не менее одно и то же значение может повторяться в массиве неоднократно. В рассматриваемом здесь массиве могут быть представлены разные овощи одинаковой зеленой окраски, например, перец, огурцы, сельдерей.

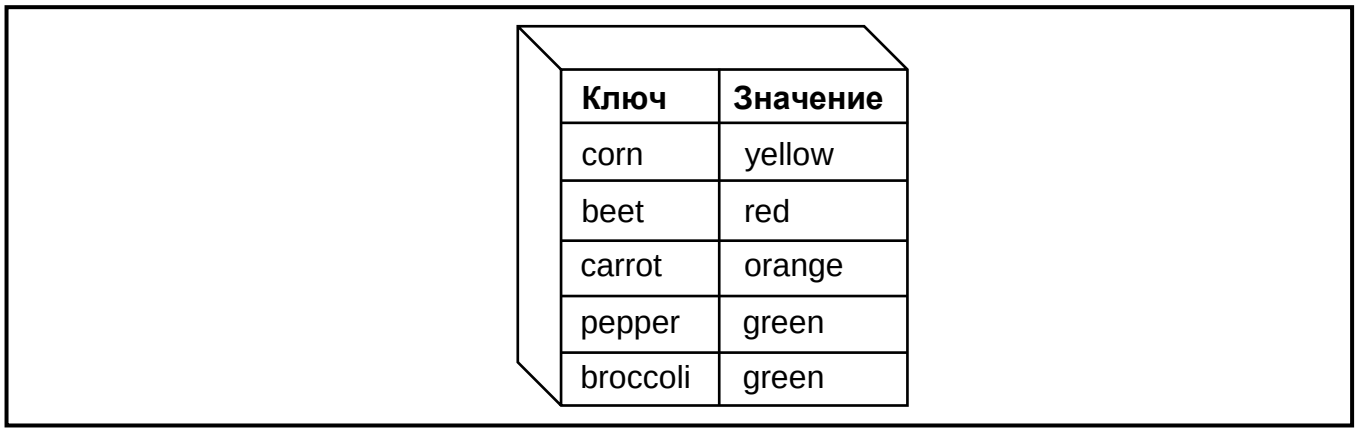


Рис. 4.1: Ключи и значения в массиве, хранящем информацию об окраске овощей

Любое строковое или числовое значение, например **corn**, **4**, **-36** или **Salt Baked Squid**, может служить ключом к элементу массива. Массивы и другие нескалярные величины[1] не могут служить ключами, но они могут быть значениями элементов массива. В качестве элемента массива можно указать символьную строку, число, логическое значение `true` или `false` и такой нескалярный тип данных, как другой массив.

Создание массива

Для создания массива следует воспользоваться языковой конструкцией array(), указав через запятую список пар "ключ-значение", где ключ и значение разделяются знаками =>. Такой порядок создания массивов демонстрируется в примере 4.1.

Пример 4.1. Создание массивов

```
$vegetables = array('corn' => 'yellow',
                    'beet' => 'red',
                    'carrot' => 'orange');

$dinner = array(0 => 'Sweet Corn and Asparagus',
                1 => 'Lemon Chicken',
                2 => 'Braised Bamboo Fungus');

$computers = array('trs-80' => 'Radio Shack',
                   2600 => 'Atari',
                   'Adam' => 'Coleco');
```

Ключи и значения в массиве из примера 4.1 представлены символьными строками (например, **corn**, **Braised Bamboo Fungus** и **Coleco**) и числами (в частности, **0**, **1** и **2600**). Они обозначаются как и любые другие символьные строки и числа в программах на PHP, где в кавычки заключаются символьные строки, но не числа.

Языковая конструкция `array()` сокращенно обозначается парой квадратных скобок, называемой *сокращенным синтаксисом массивов*. В примере 4.2 создаются те же самые массивы, что и в примере 4.1, но теперь это делается с помощью сокращенного синтаксиса массивов.

[1] *Скалярными* называются данные, имеющие единственное значение, например, число, фрагмент текста, логическое значение `true` или `false`. А такие сложные типы данных, как массивы, содержащие многие значения, называются *нескалярными*.

Пример 4.2. Применение сокращенного синтаксиса массивов

```
$vegetables = ['corn' => 'yellow', 'beet' => 'red',
               'carrot' => 'orange'];

$dinner = [0 => 'Sweet Corn and Asparagus',
           1 => 'Lemon Chicken',
           2 => 'Braised Bamboo Fungus'];

$computers = ['trs-80' => 'Radio Shack', 2600 => 'Atari',
              'Adam' => 'Coleco'];
```

Сокращенный синтаксис массивов был внедрен в версии PHP 5.4. Если вы пользуетесь более ранней версией PHP, придерживайтесь языковой конструкции `array()`.

Элементы можно также вводить в массив по очереди, присваивая значение конкретному ключу в массиве. Так, в примере 4.3 создаются те же самые массивы, что и в двух предыдущих примерах, но теперь это делается поэлементно.

Пример 4.3. Создание массива поэлементно

```
// Массив $vegetables со строковыми ключами
$vegetables['corn'] = 'yellow';
$vegetables['beet'] = 'red';
$vegetables['carrot') = 'orange';

// Массив $dinner с числовыми ключами
$dinner[0] = 'Sweet Corn and Asparagus';
$dinner[1] = 'Lemon Chicken';
$dinner[2] = 'Braised Bamboo Fungus';

// Массив $computers с числовыми и строковыми ключами
$computers['trs-80'] = 'Radio Shack';
$computers[2600] = 'Atari';
$computers['Adam'] = 'Coleco';
```

В примере 4.3 квадратные скобки после имени переменной обозначают ссылку на конкретный ключ в массиве. Присвоив значение этому ключу, можно создать элемент в массиве.

Выбор подходящего имени для массива

Имена переменных, хранящих массивы, следуют тем же правилам, что и имена любых других переменных. В частности, имена массивов и скалярных переменных выбираются из числа возможных имен, а следовательно, нельзя одновременно иметь массив и скалярную переменную под одним и тем же именем `$vegetables`. Если присвоить скалярное значение элементу массива, или наоборот, то прежнее значение будет негласно стерто, и переменная получит новое значение. Так, в примере 4.4 массив `$vegetables` превращается в скалярную переменную, а скалярная переменная

`$fruits` — в массив.

Пример 4.4. Взаимное превращение скалярных и нескалярных величин

```
// Создание массива $vegetables
$vegetables['corn'] = 'yellow';

// Бесследное удаление строк "corn" и "yellow" и
// создание скалярной переменной $vegetables
$vegetables = 'delicious';

// Создание скалярной переменной $fruits
$fruits = 283;

// Не пройдет! Значение 283 по-прежнему остается в
// переменной $fruits, а интерпретатор PHP выдает предупреждение
$fruits['potassium'] = 'banana';

// А в данном случае содержимое переменной $fruits
// перезаписывается, и она становится массивом
$fruits = array('potassium' => 'banana');
```

В каждом из массивов `$vegetables` и `$computers` из примера 4.1 хранится список взаимосвязей. Так, в массиве `$vegetables` овощи связаны со своей окраской, а в массиве `$computers` — наименования компьютеров с их производителями. Но в массиве `$dinner` важны лишь названия блюд, которые являются значениями элементов массива, тогда как ключи просто обозначены числами, чтобы отличать элементы массива друг от друга.

Создание числовых массивов

В языке PHP обеспечиваются сокращенные способы организации массивов, имеющих только числа в качестве ключей. Так, если создать массив с помощью сокращенного синтаксиса `[]` или конструкции `array()`, указав только список значений вместо пар "ключ-значение", интерпретатор PHP автоматически присвоит числовые ключи каждому значению в массиве. Ключи в таком массиве начинаются с нуля, возрастая по порядку с каждым новым элементом массива. Такой прием демонстрируется в примере 4.5, где создается числовой массив `$dinner`.

Пример 4.5. Создание числовых массивов с помощью языковой конструкции `array()`

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
print "I want $dinner[0] and $dinner[1]
```

При выполнении кода из примера 4.5 на экран выводится следующий результат;

```
I want Sweet Corn and Asparagus and Lemon Chicken.
```

В самом интерпретаторе PHP массивы с числовыми и строковыми ключами, а также со смешанными ключами, состоящими из чисел и строк, интерпретируются одинаково. По аналогии с другими языками программирования массивы только с числовыми ключами обычно называются числовыми, индексированными или упорядоченными, а массивы со строковыми ключами — ассоциативными.

Иными словами, *ассоциативный массив* относится к тем видам массивов, ключи в которых обозначают нечто иное, чем позиции значений в массиве. В данном случае каждый ключ *ассоциируется* со своим значением.

Ключи в числовых массивах автоматически инкрементируются при создании самого массива или вводе в него новых элементов с помощью сокращенного синтаксиса, как показано в примере 4.6.

Пример 4.6. Ввод в массив новых элементов с помощью сокращенного синтаксиса

```
// Создать массив $lunch, состоящий из двух элементов.
// В следующей строке кода задается первый элемент
// массива $lunch[0]
$lunch[] = 'Dried Mushrooms in Brown Sauce';
// В следующей строке кода задается второй элемент
// массива $lunch[1]
$lunch[] = 'Pineapple and Yu Fungus';

// Создать массив $dinner, состоящий из трех элементов
$dinner = array('Sweet Corn and Asparagus', 'Lemon Chicken',
                'Braised Bamboo Fungus');
// Ввести новый элемент в конце массива $dinner.
//В следующей строке кода задается четвертый элемент
// массива $dinner[3]
$dinner[] = 'Flank Skin with Spiced Flavor';
```

При указании пустых квадратных скобок в массив вводится новый элемент, который получает числовой ключ, на единицу больший самого большого числового ключа в массиве. Если же массив еще не существует, то при указании пустых квадратных скобок вводится первый элемент нового массива с нулевым ключом.

Присваивание первому элементу массива нулевого, а не единичного ключа кажется не совсем логичным для простых смертных, но не для программистов. Поэтому стоит еще раз подчеркнуть, что первый элемент числового массива имеет нулевой (`0`), а не единичный (`1`) ключ.

Определение размера массива

Функция `count()` сообщает о количестве элементов в массиве. Применение функции `count()` демонстрируется в примере 4.7.

Пример 4.7. Определение размера массива

```
dinner = array('Sweet Corn and Asparagus',
               'Lemon Chicken',
               'Braised Bamboo Fungus');

$dishes = count($dinner);

print "There are $dishes things for dinner.";
```

При выполнении кода из примера 4.7 на экран выводится следующий результат:

```
There are 3 things for dinner.
```

Если передать функции `count()` пустой массив, т.е. такой массив, в котором отсутствуют элементы, она возвратит нулевое значение. В проверочном выражении условного оператора `if()` пустой массив определяется как ложный (`false`).

Перебор массивов

К числу наиболее распространенных операций с массивами относится обращение к каждому элементу массива в отдельности и обработка каким-то способом его значения. Это может быть внедрение элемента массива в строку HTML-таблицы или сложение его значения с промежуточной суммой.

Обойти каждый элемент массива проще всего с помощью языковой конструкции `foreach()`, которая позволяет выполнять блок кода для каждого элемента в массиве. Так, в примере 4.8 конструкция `foreach()` применяется для вывода на экран HTML-таблицы, содержащей каждый элемент массива.

Пример 4.8. Перебор массива с помощью языковой конструкции foreach()

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');
print "<table>\n";
foreach ($meal as $key => $value) {
    print "<tr><td>$key</td><td>$value</td></tr>\n";
}
print '</table>';
```

При выполнении кода из примера 4.8 на экран выводится следующий результат:

```
<table>
<tr><td>breakfast</td><td>Walnut Bun</td></tr>
<tr><td>lunch</td><td>Cashew Nuts and White Mushrooms</td></tr>
<tr><td>snack</td><td>Dried Mulberries</td></tr>
<tr><td>dinner</td><td>Eggplant with Chili Sauce</td></tr>
</table>
```

Сначала в цикле, организуемом с помощью языковой конструкцией `foreach()`, ключ для каждого элемента массива `$meal` копируется в переменную `$key`, а значение — в переменную `$value`. Затем выполняется код в фигурных скобках. В примере 4.8 этот код осуществляет вывод на экран значений переменных `$key` и `$value` вместе с некоторой HTML-разметкой для формирования строки таблицы. Для обозначения ключа и значения в блоке кода допускается выбирать какие угодно имена переменных. Но если эти переменные употреблялись до языковой конструкции `foreach()`, то они перезаписываются значениями из массива.

Когда вывод данных из HTML-таблицы организуется с помощью языковой конструкции `foreach()`, зачастую к каждой строке таблицы требуется применить чередующиеся классы CSS. Это нетрудно сделать, сохранив сначала имена чередующихся классов в отдельном массиве, а затем заменяя значение переменной с **0** на **1**, и наоборот, на каждом шаге цикла в языковой конструкции `foreach()`, чтобы вывести имя соответствующего класса. Так, в примере 4.9 имена двух классов чередуются в массиве `$row_styles`.

Пример 4.9. Вывод HTML-таблицы с чередованием классов CSS

```
$row_styles = array('even','odd');
$style_index = 0;
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');
print "<table>\n";
foreach ($meal as $key => $value) {
    print '<tr class=" . $row_styles[$style_index] . ">';
    print "<td>$key</td><td>$value</td></tr>\n";
    // Смена значения переменной $style_index с 0 на 1, и обратно
    $style_index = 1 - $style_index;
}
print '</table>';
```

При выполнении кода из примера 4.9 на экран выводится следующий результат:

```
<table>
<tr class="even"><td>breakfast</td><td>Walnut Bun</td></tr>
<tr class="odd"><td>lunch</td>
    <td>Cashew Nuts and White Mushrooms</td></tr>
<tr class="even"><td>snack</td><td>Dried Mulberries</td></tr>
<tr class="odd"><td>dinner</td>
    <td>Eggplant with Chili Sauce</td></tr>
</table>
```

Изменение значений таких переменных цикла, как `$key` и `$value`, в блоке кода из языковой конструкции `foreach()` не оказывает влияния на элементы в конкретном массиве. Если же требуется изменить значения элементов массива, следует воспользоваться переменной `$key` в качестве индекса. Именно такой прием применяется в примере 4.10 для удвоения значения каждого элемента массива.

Пример 4.10. Модификация массива с помощью языковой конструкции `foreach()`

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50);
foreach ($meals as $dish => $price) {
    // выражение $price = $price * 2 НЕ пройдет!
    $meals[$dish] = $meals[$dish] * 2;
}
// перебрать массив снова и вывести измененные
// значения его элементов
foreach ($meals as $dish => $price) {
    printf("The new price of %s is \$%.2f.\n",$dish,$price);
}
```

При выполнении кода из примера 4.10 на экран выводится следующий результат:

```
The new price of Walnut Bun is $2.00.
The new price of Cashew Nuts and White Mushrooms is $9.90.
The new price of Dried Mulberries is $6.00.
The new price of Eggplant with Chili Sauce is $13.00.
```

Имеется более краткая форма языковой конструкции `foreach()` для перебора числовых массивов отсутствует, как показано в примере 4.11.

Пример 4.11. Применение языковой конструкции `foreach()` для перебора числовых массивов

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
foreach ($dinner as $dish) {
    print "You can eat: $dish\n";
}
```

При выполнении кода из примера 4.11 на экран выводится следующий результат:

```
You can eat: Sweet Corn and Asparagus
You can eat: Lemon Chicken
You can eat: Braised Bamboo Fungus
```

Применяя такую форму языковой конструкции `foreach()`, достаточно указать имя одной переменной после предложения `as`, и тогда значение каждого элемента будет скопировано в эту переменную в блоке кода. Хотя ключи элементов недоступны в блоке кода.

Чтобы отслеживать текущее положение в массиве с помощью языковой конструкции `foreach()`, придется воспользоваться отдельной переменной, инкрементируемой всякий раз, когда в цикле, организуемом с помощью данной конструкции, выполняется блок кода. Текущее положение в массиве можно получить явным образом в переменной цикла с помощью языковой конструкции `for()`. Если в цикле, организуемом с помощью языковой конструкции `foreach()`, можно получить значение каждого элемента массива, то в цикле, организуемом с помощью языковой конструкции `for()`, — позицию каждого элемента в массиве. Организовать такой цикл, где можно было бы сразу получить и то и другое, в PHP нельзя.

Так, если требуется выяснить текущее положение при переборе числового массива, следует воспользоваться языковой конструкцией `for()` вместо языковой конструкции `foreach()`. Цикл, организуемый с помощью языковой конструкции `for()`, должен зависеть от переменной цикла, значение которой начинается с нуля и наращивается вплоть до величины, на единицу меньшей количества элементов в массиве (пример 4.12).

Пример 4.12. Перебор числового массива с помощью языковой конструкции for()

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
for ($i = 0, $num_dishes = count($dinner); $i < $num_dishes; $i++) {
    print "Dish number $i is $dinner[$i]\n";
}
```

При выполнении кода из примера 4.12 на экран выводится следующий результат:

```
Dish number 0 is Sweet Corn and Asparagus
Dish number 1 is Lemon Chicken
Dish number 2 is Braised Bamboo Fungus
```

При переборе массива в цикле, организуемом с помощью языковой конструкции `for()`, доступен счетчик, обозначающий текущее положение в массиве. Над этим счетчиком можно выполнить операцию взятия модуля (`%`), чтобы чередовать классы CSS в строках HTML-таблицы, как показано в примере 4.13.

Пример 4.13. Чередование классов CSS в строках HTML-таблицы с помощью языковой конструкции `for()`

```
$row_styles = array('even', 'odd');
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
print "<table>\n";
for ($i = 0, $num_dishes = count($dinner);
             $i < $num_dishes; $i++) {
    print '<tr class="' . $row_styles[$i % 2] . '">';
    print "<td>Element $i</td><td>$dinner[$i]</td></tr>\n";
}
print '</table>';
```

В примере 4.13 класс CSS, подходящий для стилевого оформления строки HTML-таблицы, определяется в выражении `$i% 2`, значение которого чередуется между **0** и **1**, в зависимости от того, является ли четным или нечетным значение переменной `$i`. Чтобы хранить имя класса CSS, подходящего для стилевого оформления строки HTML-таблицы, не требуется отдельная переменная вроде `$style_index` из примера 4.9.

При выполнении кода из примера 4.13 на экран выводится следующий результат:

```
<table>
<tr class="even"><td>Element 0</td><td>Sweet Corn and Asparagus</td></tr>
<tr class="odd"><td>Element 1</td><td>Lemon Chicken</td></tr>
<tr class="even"><td>Element 2</td><td>Braised Bamboo Fungus</td></tr>
</table>
```

При переборе массива с помощью языковой конструкции `foreach()` элементы массива будут доступны в том порядке, в каком они были введены в массив: от первого и до последнего включительно. Если же имеется числовой массив, элементы которого введены не в том порядке, в каком обычно следуют их ключи, то в конечном итоге могут быть получены неожиданные результаты. Так, в примере 4.14 элементы массива выводятся на экран не в числовом и не в алфавитном порядке.

Пример 4.14. Порядок вывода элементов массива в цикле, организуемом с помощью языковой конструкции `foreach()`

```
$letters[0] = 'A';
$letters[1] = 'B';
$letters[3] = 'D';
$letters[2] = 'C';

foreach ($letters as $letter) {
    print $letter;
}
```

При выполнении кода из примера 4.14 на экран выводится следующий результат:

```
ABDC
```

Чтобы гарантировать доступ к элементам массива в числовом порядке следования их ключей, достаточно перебрать массив в цикле, организуемом с помощью языковой конструкции `for()`, следующим образом:

```
for ($i = 0, $num_letters = count($letters);
    $i < $num_letters; $i++) {
    print $letters[$i];
}
```

При выполнении приведенных выше строк кода на экран выводится следующий результат:

```
ABCD
```

Если требуется найти конкретный элемент в массиве, то для такого поиска придется перебрать целый массив. Имеются более эффективные способы обнаружения конкретного элемента. Чтобы проверить элемент с помощью определенного ключа, следует воспользоваться функцией `array_key_exists()`, как показано в примере 4.15. Эта функция возвращает логическое значение `true`, если элемент с предоставляемым ключом существует в искомом массиве.

Пример 4.15. Проверка наличия элемента в массиве по конкретному ключу

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50,
               'Shrimp Puffs' => 0); // Shrimp Puffs are free!
$books = array("The Eater's Guide to Chinese Characters",
               'How to Cook and Eat in Chinese');

// Следующая проверка дает истинное значение
if (array_key_exists('Shrimp Puffs',$meals)) {
    print "Yes, we have Shrimp Puffs";
}

// Следующая проверка дает ложное значение
if (array_key_exists('Steak Sandwich',$meals)) {
    print "We have a Steak Sandwich";
}

// Следующая проверка дает истинное значение
if (array_key_exists(1, $books)) {
    print "Element 1 is How to Cook and Eat in Chinese";
}
```

Чтобы проверить элемент с конкретным значением, следует воспользоваться функцией `in_array()` (пример 4.16).

Пример 4.16. Проверка наличия в массиве элемента с конкретным значением

```
$meals = array('Walnut Bun' => 1,
               'Cashew Nuts and White Mushrooms' => 4.95,
               'Dried Mulberries' => 3.00,
               'Eggplant with Chili Sauce' => 6.50,
```

```
                'Shrimp Puffs' => 0);
$books = array("The Eater's Guide to Chinese Characters",
                'How to Cook and Eat in Chinese');

// Следующая проверка дает истинное значение:
// по ключу Dried Mulberries в массиве имеется значение 3.00
if (in_array(3, $meals)) {
    print 'There is a $3 item.';
}

// Следующая проверка дает истинное значение
if (in_array('How to Cook and Eat in Chinese', $books)) {
    print "We have How to Cook and Eat in Chinese";
}

// Следующая проверка дает ложное значение:
// в функции in_array() учитывается регистр букв
if (in_array("the eater's guide to Chinese characters", $books)) {
    print "We have the Eater's Guide to Chinese Characters.";
}
```

Функция `in_array()` возвращает логическое значение `true`, если обнаруживает в массиве элемент с заданным значением. Сравнение символьных строк в этой функции выполняется с учетом регистра букв. Функция `array_search()` действует аналогично функции `in_array()`, но она возвращает ключ искомого элемента вместо логического значения `true`. Так, в примере 4.17 функция `array_search()` возвращает название блюда стоимостью 6,50 доллара.

Пример 4.17. Поиск элемента в массиве по конкретному значению

```
$meals = array('Walnut Bun' => 1,
                'Cashew Nuts and White Mushrooms' => 4.95,
                'Dried Mulberries' => 3.00,
                'Eggplant with Chili Sauce' => 6.50,
                'Shrimp Puffs' => 0);
$dish = array_search(6.50, $meals);

if ($dish) {
    print "$dish costs \$6.50";
}
```

При выполнении кода из примера 4.17 на экран выводится следующий результат:

```
Eggplant with Chili Sauce costs $6.50
```

Модификация массивов

Отдельными элементами массива можно оперировать, как и обычными скалярными переменными, применяя арифметические, логические и прочие операции. Выполнение некоторых операций над элементами массива демонстрируется в примере 4.18.

Пример 4.18. Оперирование элементами массива

```
$dishes['Beef Chow Foon'] = 12;
$dishes['Beef Chow Foon']++;
$dishes['Roast Duck'] = 3;

$dishes['total'] = $dishes['Beef Chow Foon']
         + $dishes['Roast Duck'];

if ($dishes['total'] > 15) {
    print "You ate a lot: ";
}
print 'You ate ' . $dishes['Beef Chow Foon']
                . ' dishes of Beef Chow Foon.';
```

При выполнении кода из примера 4.18 на экран выводится следующий результат:

```
You ate a lot: You ate 13 dishes of Beef Chow Foon.
```

Вставка значений элементов, заключаемых в двойные кавычки, или встраиваемых документов осуществляется аналогично вставке чисел и символьных строк. Вставить элемент массива проще всего, введя его ключ в символьную строку, но не заключая этот ключ в кавычки, как показано в примере 4.19.

Пример 4.19. Вставка элементов массива в символьные строки, заключаемые в двойные кавычки

```
$meals['breakfast'] = 'Walnut Bun';
$meals['lunch'] = 'Eggplant with Chili Sauce';
$amounts = array(3, 6);

print "For breakfast, I'd like $meals[breakfast]
       and for lunch,\n";
print "I'd like $meals[lunch]. I want $amounts[0]
       at breakfast and\n";
print "$amounts[1] at lunch.";
```

При выполнении кода из примера 4.19 на экран выводится следующий результат:

```
For breakfast, I'd like Walnut Bun and for lunch,
I'd like Eggplant with Chili Sauce. I want 3 at breakfast and
6 at lunch.
```

Вставка в примере 4.19 осуществляется только по ключам массива, состоящим исключительно из букв, чисел и знаков подчеркивания. Если же в массиве имеется ключ, содержащий пробел или знак препинания, то для вставки элемента массива по такому ключу последний следует заключить в фигурные скобки, как в примере 4.20.

Пример 4.20. Вставка элементов массива в символьные строки по ключам, заключаемым в фигурные скобки

```
$meals['Walnut Bun'] = '$3.95';
$hosts['www.example.com'] = 'website';

print "A Walnut Bun costs [$meals['Walnut Bun']}.\n";
print "www.example.com is a {$hosts['www.example.com']}.";
```

При выполнении кода из примера 4.20 на экран выводится следующий результат:

```
A Walnut Bun costs $3.95.
www.example.com is a website.
```

В символьной строке, заключаемой в двойные кавычки, или во встраиваемом документе сначала вычисляется выражение, указанное в фигурных скобках, а затем вставляется его значение в указанном месте строки или документа. Так, в примере 4.20 подобные выражения состоят только из элементов массива, и поэтому значения элементов массива вставляются в символьные строки в указанных для них местах.

Чтобы удалить элемент из массива, достаточно вызвать функцию `unset()`:

```
unset($dishes['Roast Duck']);
```

При удалении элемента с помощью функции `unset()` не просто устанавливается нулевое значение элемента или пустая символьная строка. После вызова функции `unset()` элемент отсутствует в массиве, и его уже будет недоставать при обходе массива или подсчете количества его элементов. Применить функцию `unset()` к массиву, где хранятся запасы товаров на складе, — это все равно, что сказать: данного товара на складе больше нет. А установить нулевое значение или пустую строку в элементе массива — это все равно, что сказать: товара временно нет в наличии.

Если требуется вывести на экран значения сразу всех элементов массива, то для этого проще всего вызвать функцию `implode()`. Эта функция формирует символьную строку, в которой объединяются все значения элементов массива, разделяемые заданным для строк ограничителем. Так, в примере 4.21 на экран выводится разделяемый запятыми список блюд китайской кухни “дяньсинь”.

Пример 4.21. Формирование символьной строки из элементов массива с помощью функции `implode()`

```
$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');
$menu = implode(', ', $dimsum);
print $menu;
```

При выполнении кода из примера 4.21 на экран выводится следующий результат:

```
Chicken Bun, Stuffed Duck Web, Turnip Cake
```

Чтобы вывести весь массив без разделения его элементов заданным ограничителем, достаточно указать пустую строку в качестве первого аргумента при вызове функции `implode()`:

```
$letters = array('A','B','C ,'D');
print implode('', $letters);
```

При выполнении приведенных выше строк кода на экран выводится следующий результат:

```
ABCD
```

В примере 4.22 наглядно показано, как воспользоваться функцией `implode()`, чтобы упростить вывод строк HTML-таблицы на экран.

Пример 4.22. Вывод строк HTML-таблицы на экран с помощью функции `implode()`

```
$dimsum = array('Chicken Bun','Stuffed Duck Web','Turnip Cake');
print '<tr><td>' . implode ('</td><td>', $dimsum) . '</td></tr>';
```

При выполнении кода из примера 4.22 на экран выводится следующий результат:

```
<tr><td>Chicken Bun</td><td>Stuffed Duck Web</td>
<td>Turnip Cake</td></tr>
```

Функция `implode()` разделяет выводимое значение каждого элемента массива заданным ограничителем. Поэтому необходимо также вывести открывающие дескрипторы перед значением первого элемента массива и закрывающие дескрипторы вслед за последним его элементом, чтобы полностью сформировать строку HTML-таблицы.

С функцией `implode()` сопряжена функция `explode()`, разбивающая символьную строку на элементы массива. В качестве аргумента, обозначающего ограничитель, указывается подстрока, которую следует искать для разделения исходной строки на элементы массива. Применение функции `explode()` демонстрируется в примере 4.23.

Пример 4.23. Преобразование символьной строки в массив с помощью функции `explode()`

```
$fish = 'Bass, Carp, Pike, Flounder';
$fish_list = explode(', ', $fish);
print "The second fish is $fish_list[1]";
```

При выполнении кода из примера 4.23 на экран выводится следующий результат:

```
The second fish is Carp
```

Сортировка массивов

Массивы можно отсортировать несколькими способами. Выбор конкретной функции для сортировки массива зависит от требующегося порядка сортировки и типа массива.

Функция `sort()` сортирует массив по значениям его элемента. Ее можно применять только к числовым массивам, поскольку в противном случае она сбрасывает исходные ключи в массиве по ходу сортировки. Состояние массивов до и после сортировки с помощью функции `sort()` показано в примере 4.24.

Пример 4.24. Сортировка массива с помощью функции `sort()`

```
$dinner = array('Sweet Corn and Asparagus',
                'Lemon Chicken',
                'Braised Bamboo Fungus');
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');

print "Before Sorting:\n";
foreach ($dinner as $key => $value) {
    print " $dinner: $key $value\n";
}
foreach ($meal as $key => $value) {
print " $meal: $key $value\n";
}

sort($dinner);
sort($meal);
```

```
print "After Sorting:\n";
foreach ($dinner as $key => $value) {
    print " $dinner: $key $value\n";
}
foreach $meal as $key => $value) {
    print "    \$meal: $key $value\n";

}
```

При выполнении кода из примера 4.24 на экран выводится следующий результат:

```
Before Sorting:
[ До сортировки ]
 $dinner: 0 Sweet Corn and Asparagus
 $dinner: 1 Lemon Chicken
 $dinner: 2 Braised Bamboo Fungus
   $meal: breakfast Walnut Bun
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
After Sorting:
[ После сортировки ]
 $dinner: 0 Braised Bamboo Fungus
 $dinner: 1 Lemon Chicken
 $dinner: 2 Sweet Corn and Asparagus
   $meal: 0 Cashew Nuts and White Mushrooms
   $meal: 1 Dried Mulberries
   $meal: 2 Eggplant with Chili Sauce
   $meal: 3 Walnut Bun
```

Оба массива в данном примере были упорядочены по возрастающей значений элементов. Теперь первый элемент массива `$dinner` содержит строковое значение **`Braised Bamboo Fungus`**, а первый элемент массива `$meal` — строковое значение **`Cashew Nuts and White Mushrooms`**. Ключи в массиве `$dinner` не претерпели никаких изменений, поскольку до сортировки это был числовой массив. А вот ключи в массиве `$meal` были заменены на числовые от **`0`** до **`3`**.

Чтобы отсортировать ассоциативный массив по значениям его элементов, следует воспользоваться функцией `asort()`, которая сохраняет ключи вместе с их значениями. В примере 4.25 демонстрируется сортировка массива `$meal` из примера 4.24 с помощью функции `asort()`.

Пример 4.25. Сортировка массива с помощью функции asort()

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');

print "Before Sorting:\n";
foreach ($meal as $key => $value) {
    print " \$meal: $key $value\n";
}

asort($meal);
```

```
print "After Sorting:\n";
foreach ($meal as $key => $value) {
    print " \$meal: $key $value\n";
}
```

При выполнении кода из примера 4.25 на экран выводится следующий результат:

```
Before Sorting:
   $meal: breakfast Walnut Bun
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
After Sorting:
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
   $meal: breakfast Walnut Bun
```

В данном примере значения были отсортированы в массиве с помощью функции `asort()` таким же образом, как и в предыдущем примере с помощью функции `sort()`. Но на этот раз ключи в массиве сохранились в прежнем состоянии.

Если функции `asort()` и `sort()` сортируют массивы по значениям их элементов, то с помощью функции `ksort()` их можно отсортировать по ключам. При этом пары "ключ-значение" сохраняются вместе, но в то же время упорядочиваются по ключам. В примере 4.26 демонстрируется сортировка массива `$meal` с помощью функции `ksort()`.

Пример 4.26. Сортировка массива с помощью функции `ksort()`

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');

print "Before Sorting:\n";
foreach ($meal as $key => $value) {
    print "    \$meal: $key $value\n";
}

ksort($meal);

print "After Sorting:\n";
foreach ($meal as $key => $value) {
    print "    \$meal: $key $value\n";
}
```

При выполнении кода из примера 4.26 на экран выводится следующий результат:

```
Before Sorting:
   $meal: breakfast Walnut Bun
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
After Sorting:
```

```
   $meal: breakfast Walnut Bun
   $meal: dinner Eggplant with Chili Sauce
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
```

Массив упорядочивается таким образом, чтобы ключи следовали теперь в алфавитном порядке по возрастающей. Каждый элемент массива остается без изменения, и поэтому значение, которое было связано с каждым ключом до сортировки, остается привязанным к нему и после сортировки. Если отсортировать числовой массив с помощью функции `ksort()`, его элементы расположатся в алфавитном порядке по возрастающей. Именно в таком порядке они и располагаются при создании числового массива с помощью языковой конструкции `array()` или `[]`.

У функций сортировки массивов `sort()`, `asort()` и `ksort()` имеются аналоги, сортирующие массивы по убывающей и соответственно называемые `rsort()`, `arsort()` и `krsort()`. Они действуют аналогично функциям `sort()`, `asort()` и `ksort()`, но сортируют массив таким образом, чтобы ключи или значения располагались в массиве по убывающей, начиная с самого большого (в алфавитном порядке) ключа или значения. В примере 4.27 показано применение функции `arsort()`.

Пример 4.27. Сортировка массива с помощью функции `arsort()`

```
$meal = array('breakfast' => 'Walnut Bun',
              'lunch' => 'Cashew Nuts and White Mushrooms',
              'snack' => 'Dried Mulberries',
              'dinner' => 'Eggplant with Chili Sauce');

print "Before Sorting:\n";
foreach ($meal as $key => $value) {
    print "   \$meal: $key $value\n";
}

arsort($meal);

print "After Sorting:\n";
foreach ($meal as $key => $value) {
    print "   \$meal: $key $value\n";
}
```

При выполнении кода из примера 4.27 на экран выводится следующий результат:

```
Before Sorting:
   $meal: breakfast Walnut Bun
   $meal: lunch Cashew Nuts and White Mushrooms
   $meal: snack Dried Mulberries
   $meal: dinner Eggplant with Chili Sauce
After Sorting:
   $meal: breakfast Walnut Bun
   $meal: dinner Eggplant with Chili Sauce
   $meal: snack Dried Mulberries
   $meal: lunch Cashew Nuts and White Mushrooms
```

Функция `arsort()` сохраняет связь ключей со значениями в массиве аналогично функции `asort()`, но она располагает элементы в противоположном (по значению) порядке. Теперь первым в массиве оказывается элемент, строковое значение которого начинается с буквы **W**, а последним — элемент, строковое значение которого начинается с буквы **C**.

Применение многомерных массивов

Как упоминалось ранее в разделе “Основы организации массивов”, в качестве значения элемента одного массива может служить другой массив. Это удобно в том случае, если требуется сохранить данные, имеющие более сложную структуру, чем просто ключ и единственное значение. Стандартная пара “ключ-значение” вполне пригодна для сопоставления названия трапезы (например, **breakfast** или **lunch**) с единственным блюдом (например, **Walnut Bun** или **Chicken with Cashew Nuts**). Но как быть, если каждая трапеза состоит из нескольких блюд? В таком случае значениями элементов должны быть массивы, а не символьные строки.

Для создания одних массивов, содержащих в качестве значений своих элементов другие массивы, можно воспользоваться языковой конструкцией `array()` или сокращенным синтаксисом `[]`, как в примере 4.28.

Пример 4.28. Создание многомерных массивов с помощью языковой конструкции `array()` или сокращенного синтаксиса `[]`

```
$meals = array('breakfast' => ['Walnut Bun','Coffee'],
               'lunch' => ['Cashew Nuts', 'White Mushrooms'],
               'snack' => ['Dried Mulberries','Salted Sesame Crab']);

$lunches = [ ['Chicken','Eggplant','Rice'],
             ['Beef','Scallions','Noodles'],
             ['Eggplant','Tofu'] ];

$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese' => array('hot' => 'mustard',
                                    'pepper-salty' => 'prickly ash'));
```

Для доступа к элементам в этих массивах массивов следует указать дополнительные квадратные скобки, обозначающие отдельные элементы. Каждый ряд квадратных скобок обозначает один уровень вхождения в массив. В примере 4.29 демонстрируется порядок доступа к элементам многомерных массивов, определенных в примере 4.28.

Пример 4.29. Доступ к элементам многомерного массива

```
print $meals['lunch'][1]; // White Mushrooms (Белые грибы)
print $meals['snack'][0]; // Dried Mulberries (Сушеная шелковица)
print $lunches[0][0]; // Chicken (Цыпленок)
print $lunches[2][1]; // Tofu (Соевый сыр)
print $flavors['Japanese']['salty']; // soy sauce (соевый соус)
print $flavors['Chinese']['hot']; // mustard (горчица)
```

Каждый уровень в массиве называется его *размерностью*. Все массивы, представленные ранее в этой главе, были *одномерными*. У каждого из них был лишь один уровень ключей. А такие массивы, как `$meals`, `$lunches` и `$flavors`, представленные в примерах 4.28 и 4.29, называются *многомерными*, потому что каждый из них имеет больше одной размерности.

Создавать или модифицировать многомерные массивы можно с помощью того же самого синтаксиса квадратных скобок. В примере 4.30 демонстрируется порядок манипулирования многомерными массивами.

Пример 4.30. Манипулирование многомерными массивами

```
$prices['dinner']['Sweet Corn and Asparagus'] = 12.50;
$prices['lunch']['Cashew Nuts and White Mushrooms'] = 4.95;
$prices['dinner']['Braised Bamboo Fungus'] = 8.95;

$prices['dinner']['total'] =
        $prices['dinner']['Sweet Corn and Asparagus'] +
        $prices['dinner']['Braised Bamboo Fungus'];
$specials[0][0] = 'Chestnut Bun';
$specials[0][1] = 'Walnut Bun';
$specials[0][2] = 'Peanut Bun';
$specials[1][0] = 'Chestnut Salad';
$specials[1][1] = 'Walnut Salad';
// Если опустить индекс, новый элемент будет введен в конце массива.
// В следующей строке кода создается элемент массива $specials[1][2]
$specials[1][] = 'Peanut Salad';
```

Для перебора многомерного массива по каждой его размерности следует организовать вложенные циклы с помощью языковой конструкции `foreach()` или `for()`. Так, в примере 4.31 с помощью языковой конструкции `foreach()` организуется вложенный цикл для перебора многомерного ассоциативного массива.

Пример 4.31. Перебор многомерного массива во вложенном цикле, организованном с помощью языковой конструкции `foreach()`

```
$flavors = array('Japanese' => array('hot' => 'wasabi',
                                     'salty' => 'soy sauce'),
                 'Chinese' => array('hot' => 'mustard',
                                    'pepper-salty' => 'prickly ash'));

// Переменная $culture содержит ключ, а переменная
// $culture_flavors - значение (в данном случае — массив)
foreach ($flavors as $culture => $culture_flavors) {
    // Переменная $flavor содержит ключ, а переменная
    // $example — значение
    foreach ($culture_flavors as $flavor => $example) {
       print "A $culture $flavor flavor is $example.\n";
    }
}
```

При выполнении кода из примера 4.31 на экран выводится следующий результат:

```
A Japanese hot flavor is wasabi.
A Japanese salty flavor is soy sauce.
A Chinese hot flavor is mustard.
A Chinese pepper-salty flavor is prickly ash.
```

В первом (внешнем) цикле, организуемом с помощью языковой конструкции `foreach()`, массив `$flavors` перебирается по первой его размерности. Ключи, хранящиеся в переменной `$culture`, представлены символьными строками `Japanese` и `Chinese`, а значения, хранящиеся в переменной `$culture_flavors`, являются массивами, представляющими значения элементов массива в данной размерности. В следующем (внутреннем) цикле, организуемом с помощью языковой конструкции

foreach(), массив $flavors перебирается по его второй размерности, т.е. по массивам в качестве значений его элементов. При этом ключи вроде hot и salty копируются в переменную $flavor, а значения вроде wasabi и soy sauce — в переменную $example. Переменные обоих циклов (внешнего и внутреннего) применяются в блоке кода второго (внутреннего) цикла для вывода на экран полностью сформированного сообщения.

Перебор многомерного числового массива во вложенном цикле, организуемом с помощью языковой конструкции for(), осуществляется таким же образом, как и во вложенном цикле, организуемом с помощью языковой конструкции foreach(). В примере 4.32 наглядно показано, как это делается.

Пример 4.32. Перебор многомерного массива во вложенном цикле, организуемом с помощью языковой конструкции for()

```
$specials = array( array('Chestnut Bun',
                         'Walnut Bun',
                         'Peanut Bun'),
                   array('Chestnut Salad',
                         'Walnut Salad',
                         'Peanut Salad') );

// Переменная $num_specials содержит значение 2: количество
// элементов в первой размерности массива $specials
for ($i = 0, $num_specials = count($specials); $i < $num_specials;
             $i++) {
    // Переменная $num_sub содержит значение 3: количество
    // элементов в каждом подмассиве
    for ($m = 0, $num_sub = count($specials[$i]);
                 $m < $num_sub; $m++) {
        print "Element [$i][$m] is " . $specials[$i][$m] . "\n";
    }
}
```

При выполнении кода из примера 4.32 на экран выводится следующий результат:

```
Element [0][0] is Chestnut Bun
Element [0][1] is Walnut Bun
Element [0][2] is Peanut Bun
Element [1][0] is Chestnut Salad
Element [1][1] is Walnut Salad
Element [1][2] is Peanut Salad
```

Во внешнем цикле, организуемом с помощью языковой конструкции for() в примере 4.32, перебираются два элемента массива $specials. А во внутреннем цикле, организуемом с помощью языковой конструкции for(), перебирается каждый элемент подмассивов, в которых хранятся разные символьные строки. В операторе print переменная $i служит индексом в первой размерности (элементов массива $specials), а переменная $m — индексом во второй размерности (подмассива).

Чтобы вставить значение элемента многомерного массива в символьную строку, заключаемую в двойные кавычки, или во встраиваемый документ, следует воспользоваться синтаксисом фигурных скобок, как показано в примере 4.20. В примере 4.33 фигурные скобки применяются для вставки с целью получить такой же результат, как и в примере 4.32. На самом деле исходный код в примере 4.33 отличается от исходного кода в примере 4.32 единственной строкой с оператором print.

Пример 4.33. Вставка значения элемента многомерного массива

```
$specials = array( array('Chestnut Bun',
                         'Walnut Bun',
                         'Peanut Bun'),
                   array('Chestnut Salad',
                         'Walnut Salad',
                         'Peanut Salad') );
// Переменная $num_specials содержит значение 2: количество
// элементов в первой размерности массива $specials
for ($i = 0, $num_specials = count($specials);
             $i < $num_specials; $i++) {
    // Переменная $num_sub содержит значение 3: количество
    // элементов в каждом подмассиве
    for ($m = 0, $num_sub = count($specials[$i]);
                 $m < $num_sub; $m++) {
        print "Element [$i][$m] is $specials[$i][$m]\n";
    }
}
```

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о составляющих массива: элементах, ключах и значениях.
- Два способа определения массива в программах на PHP: с помощью языковой конструкции `array()` и сокращенного синтаксиса массивов.
- Ввод элементов в массив с помощью квадратных скобок.
- Представление о сокращенном синтаксисе доступа к массивам с числовыми ключами.
- Подсчет количества элементов в массиве.
- Обход каждого элемента массива с помощью языковой конструкции `foreach()`.
- Чередование классов CSS в строках HTML-таблицы с помощью языковой конструкции `foreach()` и массива классов имен.
- Модификация значений элементов массива в блоке кода, выполняемом в цикле, организуемом с помощью языковой конструкции `foreach()`.
- Обход каждого элемента числового массива с помощью языковой конструкции `for()`.
- Чередование классов CSS в строках HTML-таблицы с помощью языковой конструкции `for()` и операции взятия модуля (`%`).
- Представление о порядке, в котором осуществляется обход элементов массива в циклах, организуемых с помощью языковых конструкций `foreach()` и `for()`.
- Проверка элемента массива по конкретному ключу.
- Проверка элемента массива по конкретному значению.
- Вставка значений элементов массива в символьные строки.

- Удаление элемента из массива.
- Формирование символьной строки из массива с помощью функции `implode()`.
- Формирование массива из символьной строки с помощью функции `explode()`.
- Сортировка массива с помощью функции `sort()`, `asort()` или `ksort()`.
- Сортировка массива в обратном порядке.
- Определение многомерного массива.
- Доступ к отдельным элементам многомерного массива.
- Обход каждого элемента многомерного массива с помощью функции `foreach()` или `for()`.
- Вставка значений элементов многомерного массива в символьные строки.

Упражнения

1. Согласно данным Бюро переписи населения США в 2010 году, самыми крупными в Соединенных Штатах Америки были следующие города:

   - Нью-Йорк (8175133 человек)
   - Лос-Анджелес, шт. Калифорния (3792621 человек)
   - Чикаго, шт. Иллинойс (2695598 человек)
   - Хьюстон, шт. Техас (2100263 человек)
   - Филадельфия, шт. Пенсильвания (1526006 человек)
   - Феникс, шт. Аризона (1445632 человек)
   - Сан-Антонио, шт. Техас (1327407 человек)
   - Сан-Диего, шт. Калифорния (1307402 человек)
   - Даллас, шт. Техас (1197816 человек)
   - Сан-Хосе, шт. Калифорния (945942 человек)

   Определите один массив (или ряд массивов), хранящий местоположение и население перечисленных выше городов. Выведите на экран таблицу со сведениями о местоположении и населении, а также общее население всех десяти городов.

2. Видоизмените выполнение задания в предыдущем упражнении таким образом, чтобы строки в результирующей таблице были упорядочены сначала по населению, а затем по названиям городов.

3. Видоизмените выполнение задания в первом упражнении таким образом, чтобы таблица содержала также строки с общим населением каждого штата, упомянутого в перечне самых крупных городов США.

4. Выясните, как хранить каждый из приведенных ниже видов информации в массиве, а затем предоставьте пример кода, в котором создается такой массив, состоящий из нескольких элементов. Например, в следующем ассоциативном массиве в качестве ключа служит Ф.И.О. учащегося, а качестве значения — ассоциативный массив, состоящий из классов и идентификационных номеров учащихся:

```
$students =
       [ 'James D. McCawley' => [ 'grade' => 'A+', 'id' => 271231 ],
         'Buwei Yang Chao' => [ 'grade' => 'A', 'id' => 818211] ];
```

- Классы и идентификационные номера учащихся в классе.
- Количество каждого товара в запасах на складе.
- Школьные обеды, состоящие из разных блюд (закуски, салаты, напитки и т.д.), а также их стоимость на каждый день недели.
- Имена членов вашей семьи.
- Имена, возраст и родство членов вашей семьи.

ГЛАВА 5

Группирование логики в функциям и файлам

При написании компьютерных программ леность превращается в достоинство. Повторное использование уже написанного кода заметно облегчает труд программирующего. И главную роль в повторном использовании кода играют функции. Всякая *функция* состоит из ряда операторов, которые можно выполнять, просто вызывая функцию по имени, вместо того, чтобы набирать эти операторы снова. Благодаря этому экономится время и предотвращаются ошибки. Кроме того, функции упрощают применение кода, написанного другими, как вы уже имели возможность убедиться сами, анализируя приведенные ранее примеры кода, где использовались встроенные функции, написанные авторами интерпретатора PHP.

В следующем разделе "Объявление и вызов функций", с которого начинается эта глава, излагаются основы определения и применения собственных функций. Вызывая функцию, можно передать ей ряд значений, которыми она должна оперировать. Так, если написать функцию для проверки прав доступа пользователя к текущей вебстранице, то при ее вызове необходимо указать имя пользователя и наименование текущей веб-страницы. Эти значения называются *аргументами*. Ниже, в разделе "Передача аргументов функциям" поясняется, как писать функции, принимающие аргументы, и как пользоваться аргументами в самих функциях.

Некоторые функции подобны улицам с односторонним движением. Им можно передавать аргументы, но ничего не получать взамен. Так, функция `print_header()`, выводящая на экран заголовок HTML-страницы, может принимать аргумент, содержащий заглавие страницы, но она не предоставляет никакой информации после своего выполнения, а только отображает результат. Большинство функций перемещают информацию в двух направлениях. Примером тому служит упоминавшаяся выше функция управления доступом к веб-странице. Эта функция возвращает логическое значение `true`, если доступ разрешен, а иначе — логическое значение `false`. Такое значение называется *возвращаемым*. Значением, возвращаемым функцией, можно пользоваться точно так же, как и любым другим значением или переменной. Возвращаемые значения обсуждаются далее, в разделе "Возврат значений из функций".

Переменные могут использоваться в операторах, выполняемых в самой функции, таким же образом, как и в операторах, выполняемых вне функции. Но переменные в самой функции и за ее пределами имеют разные области действия. Интерпретатор PHP трактует одну переменную `$name` в самой функции и другую переменную `$name` за ее пределами как две не связанные вместе переменные. Правила употребления переменных в разных частях программы поясняются далее, в разделе "Представление об области действия переменных". Эти правила очень важно уяснить и соблюдать, чтобы правильно инициализировать и употреблять переменные в своем коде. Программные ошибки, связанные с неверным употреблением переменных, выявить очень трудно.

Функции вполне пригодны для повторного использования, и поэтому удобно создать сначала отдельные файлы с определениями функций, а затем обращаться к этим файлам из программ на

PHP. Благодаря этому в разных программах (и разных частях одной и той же программы) можно пользоваться общими функциями, не дублируя их определения. В разделе "Выполнение кода из другого файла" далее в этой главе поясняется, каким образом многие файлы связываются вместе в программе на PHP.

Объявление и вызов функций

Для создания новой функции служит ключевое слово `function`, после которого следует имя функция и ее тело, заключаемое в фигурные скобки. Так, в примере 5.1 объявляется функция `page_header()`.[1]

Пример 5.1. Объявление функции

```
function page_header() {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}
```

Функции именуются по тем же правилам, что и переменные. Их имена должны начинаться с буквы или знака подчеркивания, а остальными символами в имени функции могут быть буквы, числа и знаки подчеркивания. И хотя интерпретатор PHP не препятствует употреблению в программе переменной и функции с одинаковым именем, этого следует всячески избегать, поскольку наличие многих элементов с одинаковыми именами в программе затрудняет ее понимание.

Функцию `page_header()`, объявленную в примере 5.1, можно вызвать таким же образом, как и любую встроенную функцию. В примере 5.2 эта функция применяется для вывода на экран всей страницы.

Пример 5.2. Вызов функции

```
page_header();
print "Welcome, $user";
print "</body></html>";
```

Функции можно определять до и после их вызова. Интерпретатор PHP читает весь файл программы и выявляет в нем определения всех функций, прежде чем выполнять любые команды из этого файла. Обе функции, `page_header()` и `page_footer()`, благополучно выполняются в коде из примера 5.3, несмотря на то, что функция `page_header()` объявлена до ее вызова, а функция `page_footer()` — после.

Пример 5.3. Объявление функций до и после их вызова

```
function page_header() {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#ffffff">';
}

page_header();
print "Welcome, $user";
page_footer();
```

[1] Строго говоря, круглые скобки не относятся к имени функции, но их все же рекомендуется указывать при обращении к функциям. Благодаря этому функции легче отличать от переменных и прочих языковых конструкций.

```
function page_footer() {
    print '<hr>Thanks for visiting.';
    print ' </body></html>';
}
```

Передача аргументов функциям

Если одни функции (например, функция `page_header()` из предыдущего раздела) всегда выполняют одни и те же действия, то другие оперируют входными значениями, которые могут изменяться. Входные значения, предоставляемые функции, называются *аргументами*. Аргументы повышают эффективность функций, поскольку делают их более гибкими. Функцию `page_header()` можно видоизменить, чтобы она принимала цвет страницы в качестве аргумента. Видоизмененное объявление этой функции приведено в примере 5.4.

Пример 5.4. Объявление функции с одним аргументом

```
function page_header2($color) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

В объявлении приведенной выше функции введена переменная `$color`, заключаемая в круглые скобки после имени функции. Благодаря этому в коде, составляющем тело данной функции, можно воспользоваться переменной `$color`, хранящей значение, передаваемое функции в качестве аргумента при ее вызове. Эту функцию можно, например, вызвать следующим образом:

```
page_header2('cc00cc');
```

Таким образом, в теле функции `page_header2()` устанавливается значение **`cc00cc`** переменной `$color`. И в конечном счете эта функция выводит на экран следующий результат:

```
<html><head><title>Welcome to my site</title>
  </head><body bgcolor="#cc00cc">
```

Если объявить функцию, принимающую аргумент, как демонстрируется в примере 5.4, то при ее вызове следует непременно передать ей аргумент. Если же вызвать такую функцию, не указав значение в качестве ее аргумента, интерпретатор PHP выдаст предупреждение об отсутствии обязательного аргумента в вызове функции. Так, если вызвать функцию `page_header2()` следующим образом:

```
page_header2();
```

интерпретатор PHP выведет предупреждающее сообщение, аналогичное следующему:

```
PHP Warning: Missing argument 1 for page_header2()

[ Предупреждение PHP: отсутствует аргумент 1
  для функции page_header2() ]
```

Во избежание подобного предупреждения функцию можно определить таким образом, чтобы она принимала необязательный аргумент, указав значение этого аргумента по умолчанию в объявлении функции. Если при вызове такой функции предоставляется значение аргумента, то в ее теле используется именно это значение. А если такое значение не предоставлено при вызове функции, то в ее

теле используется значение, устанавливаемое по умолчанию при ее объявлении. Так, в примере 5.5 по умолчанию устанавливается значение **cc3399** переменной `$color`, которое передается объявляемой функции в качестве аргумента.

Пример 5.5. Указание значения аргумента функции, устанавливаемого по умолчанию

```
function page_header3($color = 'cc3399') {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

Вызов функции `page_header3('336699')` приводит к такому же результату, как и вызов функции `page_header2('336699')`. При выполнении кода в теле каждой из этих функций переменная `$color` принимает значение **336699**, задающее цвет фона выводимой на экран страницы в атрибуте `bgcolor` дескриптора `<body>`. Но если вызов функции `page_header2()` без аргумента приведет к появлению упомянутого выше предупреждения, то функцию `page_header3()` можно выполнить и без аргумента. В этом случае переменной `$color` будет присвоено значение **cc3399**, устанавливаемое по умолчанию.

Значения аргументов по умолчанию должны задаваться лишь как литералы, например: **12**, **cc3399** или **Shredded Swiss Chard**. Их нельзя задавать как переменные. Так, приведенная ниже функция объявлена неверно, и поэтому при ее вызове механизм PHP аварийно завершит выполнение программы.

```
$my_color = '#000000';

// Эта функция объявлена неверно, так как значение по умолчанию
// не может быть задано как переменная
function page_header_bad($color = $my_color) {
    print '<html><head><title>Welcome to my site</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

Чтобы определить функцию, принимающую несколько аргументов, их следует указать списком через запятую в объявлении функции. Так, в примере 5.6 объявляется функция `page_header4()`, принимающая два аргумента: `$color` и `$title`.

Пример 5.6. Объявление функции с двумя аргументами

```
function page_header4($color, $title) {
    print '<html><head><title>
                  Welcome to ' . $title . '</title></head>';
    print '<body bgcolor="#' . $color . '">';
}
```

Чтобы передать функции несколько аргументов при ее вызове, их следует указать тем же самым списком через запятую. Так, в примере 5.7 функция `page_header4()` вызывается со значениями переменных `$color` и `$title` в качестве аргументов.

Пример 5.7. Вызов функции с двумя аргументами

```
page_header4('66cc66','my homepage');
```

При выполнении кода из примера 5.7 на экран выводится следующий результат:

```
<html><head><title>
  Welcome to my homepage</title></head><body bgcolor="#66cc66">
```

В примере 5.8 оба аргумента являются обязательными. Аналогичный синтаксис можно употреблять и в функциях, принимающих несколько аргументов, чтобы обозначить значения аргументов по умолчанию, как это обычно делается в объявлениях функций, принимающих единственный аргумент. Но все необязательные аргументы должны следовать после любых обязательных аргументов. В примере 5.6 демонстрируются верные способы объявления функций, принимающих три аргумента, один, два или три из которых являются необязательными.

Пример 5.8. Объявление функций с несколькими необязательными аргументами

```
// Объявление функции с одним необязательным аргументом,
// который должен быть указан последним
function page_header5($color, $title, $header = 'Welcome') {
    print '<html><head><title>
                  Welcome to ' . $title . '</title></head>';
    print '<body bgcolor="#' . $color . '">';
    print "<h1>$header</h1>";
}
// Допустимые способы вызова данной функции:
page_header5('66cc99','my wonderful page');
    // В этом вызове используется значение аргумента $header,
    // устанавливаемое по умолчанию
page_header5('66cc99','my wonderful page','This page is great!');
    // В этом вызове значение по умолчанию не используется

// Объявление функции с двумя необязательными аргументами,
// которые должны быть указаны последними
function page_header6($color, $title = 'the page',
                      $header = 'Welcome') {
   print '<html><head><title>
                 Welcome to ' . $title . '</title></head>';
   print '<body bgcolor="#' . $color . '">';
   print "<h1>$header</h1>";
}

// Допустимые способы вызова данной функции:
page_header6('66cc99'); // В этом вызове используются значения
    // аргументов $title и $header, устанавливаемые по умолчанию
page_header6('66cc99','my wonderful page');
    // В этом вызове используется значение аргумента $header,
    // устанавливаемое по умолчанию
page_header6('66cc99','my wonderful page','This page is great!');
    // В этом вызове значения по умолчанию не используются

// Объявление функции со всеми тремя необязательными аргументами
function page_header7($color = '336699', $title = 'the page',
                      $header = 'Welcome') {
    print '<html><head><title>
                  Welcome to ' . $title . '</title></head>';
    print '<body bgcolor="#' . $color . '">';
    print "<h1>$header</h1>";
```

```
}

// Допустимые способы вызова данной функции:
page_header7(); // В этом вызове используются значения всех
                // аргументов, устанавливаемые по умолчанию
page_header7('66cc99'); // В этом вызове используются значения
    // аргументов $title и $header, устанавливаемые по умолчанию
page_header7('66cc99', 'my wonderful page'); // В этом вызове
                       // используется значение аргумента $header,
                       // устанавливаемое по умолчанию
page_header7('66cc99','my wonderful page','This page is great!');
    // В этом вызове значения по умолчанию не используются
```

Все необязательные аргументы должны быть указаны в конце списка аргументов функции во избежание неоднозначности их интерпретации. Так, если определить функцию `page_header7()` с первым обязательным аргументом `$color`, вторым необязательным аргументом `$title` и третьим обязательным аргументом `$header`, то что будет означать вызов `page_header7('33cc66', 'Good Morning')`? Ведь аргумент `'Good Morning'` может быть значением переменной `$title` или `$header`. Этого недоразумения можно избежать, указав все необязательные аргументы после любых обязательных аргументов.

Любые изменения, вносимые в переменную, передаваемую функции в качестве аргумента, не оказывают влияния на саму переменную за пределами функции[2]. Так, в примере 5.9 значение переменной `$counter` за переделами вызываемой функции не изменяется.

Пример 5.9. Изменение значений аргументов в функции

```
function countdown($top) {
    while ($top > 0) {
        print "$top..";
        $top--;
    }
    print "boom!\n";
}

$counter = 5;
countdown($counter);
print "Now, counter is $counter";
```

При выполнении кода из примера 5.9 на экран выводится следующий результат:

```
5..4..3..2..1..boom!
Now, counter is 5
```

Передача переменной `$counter` в качестве аргумента функции `countdown()` указывает интерпретатору PHP на необходимость скопировать значение переменной `$counter` в переменную `$top` в начале выполнения функции, поскольку именем `$top` обозначен аргумент данной функции. Все, что происходит с переменной `$top` в теле функции, не оказывает никакого влияния на переменную `$counter`. Как только значение переменной `$counter` будет скопировано в переменную `$top`, переменная `$counter` останется незатронутой в процессе выполнения функции.

Видоизменение аргументов не оказывает влияния на переменные за переделами функции, даже если аргумент имеет такое же имя, как и внешняя переменная. Если изменить функцию `countdown()`

[2] За исключением объектов. Если передать функции объект, то изменения, вносимые в объект в теле функции, окажут влияние на его состояние за пределами вызываемой функции. Объекты рассматриваются в главе 6.

в примере 5.9 таким образом, чтобы ее аргумент назывался `$counter`, а не `$top`, то значение внешней переменной `$counter` все равно не изменится. В данном случае аргумент и внешняя переменная просто называются одинаково, но никак не связаны вместе.

Возврат значений из функций

Функция вывода на экран заголовка веб-страницы, рассмотренная ранее в этой главе, предпринимает определенное действие, отображая результат. Помимо таких действий, как вывод данных или сохранение информации в базе данных, функции способны также вычислять значение, называемое *возвращаемым* и применяемое далее в программе. Чтобы возвратить значение из функции, результат ее вызова следует присвоить переменной. Так, в примере 5.10 значение, возвращаемое встроенной функцией `number_format()`, сохраняется в переменной `$number_to_display`.

Пример 5.10. Сохранение значения, возвращаемого функцией

```
$number_to_display = number_format(321442019);
print "The population of the US is about: $number_to_display";
```

Как и в примере 1.6, при выполнении кода из примера 5.10 на экран выводится следующий результат:

```
The population of the US is about: 321,442,019
```

Значение, возвращаемое функцией, присваивается переменной таким же образом, как и символьная строка или число. В частности, оператор `$number = 57` означает следующее: сохранить значение **57** в переменной `$number`, а оператор `$number_to_display = number_format (321442019)` — вызвать функцию `number_format()` с аргументом **321442019** и сохранить возвращаемое значение в переменной `$number_to_display`. Как только значение, возвращаемое функцией, будет присвоено переменной, значением этой переменной можно воспользоваться таким же образом, как и значением любой другой переменной в программе.

Чтобы возвратить значения из создаваемых функций, следует воспользоваться ключевым словом `return`, указав после него возвращаемое значение. Как только выполнение функции дойдет до оператора `return`, оно остановится и будет возвращено значение, связанное с этим оператором. В примере 5.11 определяется функция, возвращающая общую сумму в ресторанном счете с учетом налога на добавленную стоимость и чаевых.

Пример 5.11. Возврат значения из функции

```
function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;

    return $total_amount;
}
```

Значением, возвращаемым функцией `restaurant_check()`, можно воспользоваться таким же образом, как и любым другим значением в программе. Так, в примере 5.12 возвращаемое значение используется в условном операторе `if()`.

Пример 5.12. Применение возвращаемого значения в условном операторе if()

```
// Рассчитать общую стоимость трапезы на 15,22 долларов США
// с учетом налога на добавленную стоимость (8.25%) и чаевых (15%)
$total = restaurant_check(15.22, 8.25, 15);

print 'I only have $20 in cash, so...';
if ($total > 20) {
    print "I must pay with my credit card.";
} else {
    print "I can pay with cash.";
}
```

В отдельном операторе `return` из функции можно возвратить только одно значение, но не несколько. Оператор вроде `return 15, 23` в PHP недопустим. Если же требуется возвратить несколько значений из функции, их следует сначала разместить в массиве, а затем возвратить массив.

В примере 5.13 приведен видоизмененный вариант функции `restaurant.check()`, возвращающей двухэлементный массив, содержащий общую сумму как без учета, так и с учетом налога на добавленную стоимость и чаевых.

Пример 5.13. Возврат массива из функции

```
function restaurant_check2($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_notip = $meal + $tax_amount;
    $total_tip = $meal + $tax_amount + $tip_amount;

    return array($total_notip, $total_tip);
}
```

А в примере 5.14 демонстрируется применение массива, возвращаемого функцией `restaurant_check2()`.

Пример 5.14. Применение массива, возвращаемого функцией

```
$totals = restaurant_check2(15.22, 8.25, 15);

if ($totals[0] < 20) {
    print 'The total without tip is less than $20.';
}

if ($totals[1] < 20) {
    print 'The total with tip is less than $20.';
}
```

Несмотря на то что в одном операторе `return` допускается возвращать только одно значение из функции, в ее теле можно употребить несколько подобных операторов. Первый же оператор `return`, достигнутый в ходе выполнения функции, остановит ее выполнение и возвратит из нее значение. В примере 5.15 логика определения способа оплаты ресторанного счета вынесена из примера 5.12 в новую функцию, где выясняется один из двух способов оплаты: наличными или кредитной карточкой.

Пример 5.15. Применение нескольких операторов `return` в функции

```
function payment_method($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return 'credit card';
    } else {
      return 'cash';
    }
}
```

Новая функция `payment_method()` применяется в примере 5.16, где ей передается результат выполнения функции `restaurant_check()`.

Пример 5.16. Передача возвращаемого значения другой функции

```
total = restaurant_check(15.22, 8.25, 15);
$method = payment_method(20, $total);
print 'I will pay with ' . $method;
```

При выполнении кода из примера 5.16 на экран выводится следующий результат:

```
I will pay with cash
```

Такой результат получается потому, что общая сумма, возвращаемая функцией `restaurant_check()`, меньше **20**. Эта сумма передается далее функции `payment_method()` в качестве аргумента `$total`. Первое же сравнение переменных `$amount` и `$cash_on_hand` в теле функции `payment_method()` дает логическое значение `false`, поэтому далее выполняется блок кода в условном операторе `else`. В итоге функция `payment_method()` возвращает строковое значение `cash`.

Правила обработки логических значений `truth` и `else`, обсуждавшиеся в главе 3, распространяются и на функции. Эти правила позволяют выгодно применять функции в условных операторах `if()` и других языковых конструкциях, управляющих ходом выполнения программы. Так, в примере 5.17 решение о том, что делать дальше, принимается на основании значения, возвращаемого в результате вызова функции `restaurant_check()` в проверочном выражении условного оператора `if()`.

Пример 5.17. Применение возвращаемых значений в условном операторе `if()`

```
if (restaurant_check(15.22, 8.25, 15) < 20) {
    print 'Less than $20, I can pay cash.';
} else {
    print 'Too expensive, I need my credit card.';
}
```

Чтобы вычислить проверочное выражение в коде из примера 5.17, интерпретатор PHP сначала вызывает функцию `restaurant_check()`. Значение, возвращаемое этой функцией, затем сравнивается с заданным значением **20**, как будто это значение переменной или литерала. Если функция `restaurant_check()` возвращает число меньше **20**, как в данном примере, то выполняется первый оператор `print`. В противном случае выполняется второй оператор `print`.

Проверочное выражение может также состоять из одного только вызова функции без сравнения или иной операции. В таком проверочном выражении значение, возвращаемое функцией, преобразуется в логическое значение `true` или `false` по правилам, разъясненным в разделе “Общее представление об истинности или ложности” главы 3. Если из функции возвращается логическое

значение `true`, то проверочное выражение оказывается истинным. А если из функции возвращается логическое значение `false`, то проверочное выражение оказывается ложным. Кроме того, из функции можно явным образом возвратить логическое значение `true` или `false`, чтобы сделать более очевидным ее применение в проверочном выражении. Именно так и делается в функции `can_pay_cash()` из примера 5.18, где определяется способ оплаты за трапезу в ресторане.

Пример 5.18. Возврат логического значения `true` или `false` из функции

```
function can_pay_cash($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return false;
    } else {
      return true;
    }
}

$total = restaurant_check(15.22,8.25,15);
if (can_pay_cash(20, $total)) {
    print "I can pay in cash.";
} else {
    print "Time for the credit card.";
}
```

В функции `can_pay_cash()` из примера 5.18 сравниваются два ее аргумента. Если значение аргумента `$amount` оказывается больше, то данная функция возвращает логическое значение `true`, а иначе — логическое значение `false`. Условный оператор `if()`, в котором вызывается данная функция, целенаправленно выполняет собственное назначение, выявляя истинное значение в своем проверочном выражении. А поскольку это выражение состоит только из вызова функции, то в нем вызывается функция `can_pay_cash()` с двумя аргументами **20** и `$total`. Значение, возвращаемое данной функцией, оказывается истинным, определяя вывод соответствующего сообщения на экран.

Значение, возвращаемое функцией, можно вводить в проверочное выражение аналогично переменной. Но в любом случае, когда вызывается функция, возвращающая значение, ее вызов, например, `restaurant_check(15.22,8.25,15)`, заменяется возвращаемым значением при выполнении программы.

Нередко применяемый способ сокращения кода состоит в том, чтобы сочетать вызов функции с операцией присваивания в проверочном выражении, опираясь на то обстоятельство, что результатом присваивания оказывается присваиваемое значение. Это дает возможность вызвать функцию, сохранить возвращаемое ею значение и проверить, является ли оно истинным (`true`), за один раз. В примере 5.19 наглядно демонстрируется, как это делается.

Пример 5.19. Сочетание вызова функции с операцией присваивания проверочном выражении

```
function complete_bill($meal, $tax, $tip, $cash_on_hand) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;
    if ($total_amount > $cash_on_hand) {
       // Счет больше, чем имеется наличных
       return false;
    } else {
       // этот счет можно оплатить наличными
       return $total_amount;
    }
```

```
}

if ($total = complete_bill(15.22, 8.25, 15, 20)) {
    print "I'm happy to pay $total.";
} else {
    print "I don't have enough money. Shall I wash some dishes?";
}
```

В примере 5.19 функция `complete_bill()` возвращает логическое значение `false`, если рассчитанная общая сумма ресторанного счета с учетом налога на добавленную стоимость и чаевых больше, чем наличная сумма, указанная в аргументе `$cash_on_hand`. Если общая сумма ресторанного счета будет меньше или равна наличной сумме, указанной в аргументе `$cash_on_hand`, то возвращается общая сумма ресторанного счета. Когда же вычисляется проверочное выражение в условном операторе `if()` за пределами вызываемой функции, то происходит следующее.

1. Вызывается функция `complete_bill()` с аргументами **`15.22`**, **`8.25`**, **`15`** и **`20`**.
2. Значение, возвращаемое функцией `complete_bill()`, присваивается переменной `$total`.
3. Результат присваивания, который, следует напомнить, равен присваиваемому значению, преобразуется в логическое значение `true` или `false` и далее используется как окончательный результат вычисления проверочного выражения.

Представление об области действия переменных

Как демонстрировалось в примере 5.9, изменения, происходящие в теле функции с переменными, хранящими ее аргументы, не оказывают никакого влияния на переменные за пределами функции. Дело в том, что операции, выполняемые в теле функции, происходят в другой *области действия*. Переменные, определенные за пределами функции, называются *глобальными* и существуют в одной области действия. А переменные, определенные в теле функции, называются *локальными*, и у каждой из них имеется своя область действия.

Допустим, что каждая функция соответствует филиалу крупной компании, а код за пределами любой функции — штаб-квартире данной компании. Сотрудники филадельфийского филиала называют друг друга по имени, например: “Алиса отлично справилась с этим отчетом” или “Боб никогда не положит нужного количества сахара в кофе”. В этих примерах утверждений речь идет о сотрудниках филадельфийского филиала компании, которых можно сравнить с локальными переменными в одной функции, но ничего не говорится об Алисе и Бобе, работающих в другом филиале (т.е. о локальных переменных в другой функции) или штаб-квартире компании (т.е. о глобальных переменных).

Локальные и глобальные переменные действуют одинаково. Так, переменная `$dinner` в теле функции, будь она аргументом функции или нет, никак не связана с переменной `$dinner` за пределами функции, а также с переменной `$dinner` в теле другой функции. Отсутствие такой связи между переменными в разных областях действия наглядно демонстрируется в примере 5.20.

Пример 5.20. Область действия переменных

```
$dinner = 'Curry Cuttlefish';

function vegetarian_dinner() {
    print "Dinner is $dinner, or ";
    $dinner = 'Sauteed Pea Shoots';
    print $dinner;
    print "\n";
```

```
}

function kosher_dinner() {
    print "Dinner is $dinner, or ";
    $dinner = 'Kung Pao Chicken';
    print $dinner;
    print "\n";
}

print "Vegetarian ";
vegetarian_dinner();
print "Kosher ";
kosher_dinner();
print "Regular dinner is $dinner";
```

При выполнении кода из примера 5.20 на экран выводится следующий результат:

```
Vegetarian Dinner is , or Sauteed Pea Shoots
Kosher Dinner is , or Kung Pao Chicken
Regular dinner is Curry Cuttlefish
```

Перед установкой значения в теле обеих функций переменная `$dinner` не содержит значение. Глобальная переменная `$dinner` не оказывает никакого влияния в теле функции. Но как только локальная переменная `$dinner` будет установлена в теле функции, она не будет оказывать никакого влияния на глобальную переменную `$dinner`, установленную за пределами любой функции, или на локальную переменную `$dinner` в теле другой функции. Имя `$dinner` в теле каждой функции обозначает локальный вариант переменной `$dinner` и совершенно отделено от переменной, которая имеет такое же самое имя в другой функции.

Но аналогия между областью действия переменных и организацией компании несовершенна, как и всякая аналогия. В компании можно без особого труда называть по имени сотрудников из других ее филиалов. Например, сотрудники из филадельфийского филиала могут говорить об “Алисе из штаб-квартиры” или о “Бобе из Атланты”, а руководители из штаб-квартиры могут решить судьбу “Алисы из Филадельфии” или “Майкла из Чарльстона”. Тем не менее с помощью переменных можно получить доступ к глобальным переменным за пределами функции, тогда как локальные переменные недоступны вне их функции. Это можно сравнить с возможностью для сотрудников из филиала общаться с сотрудниками из штаб-квартиры, но не с сотрудниками из других филиалов компании, а сотрудников из штаб-квартиры — с сотрудниками любого филиала.

Глобальная переменная может быть доступна из тела функции двумя способами. Самый простой способ состоит в том, чтобы искать глобальные переменные в специальном массиве `$GLOBALS`. Каждая глобальная переменная доступна в качестве элемента этого массива. В примере 5.21 демонстрируется, как пользоваться массивом `$GLOBALS`.

Пример 5.21. Доступ к глобальным переменным из массива `$GLOBALS`

```
$dinner = 'Curry Cuttlefish';

function macrobiotic_dinner() {
    $dinner = "Some Vegetables";
    print "Dinner is $dinner";
    // насладиться дарами океана
    print " but I'd rather have ";
    print $GLOBALS['dinner'];
    print "\n";
```

```
}

macrobiotic_dinner();
print "Regular dinner is: $dinner";
```

При выполнении кода из примера 5.21 на экран выводится следующий результат:

```
Dinner is Some Vegetables but I'd rather have Curry Cuttlefish
Regular dinner is: Curry Cuttlefish
```

В коде из примера 5.21 доступ к глобальной переменной `$dinner` осуществляется из функции по ссылке на элемент массива `$GLOBALS['dinner']`. В массиве `$GLOBALS` можно также изменять глобальные переменные, как показано в примере 5.22.

Пример 5.22. Видоизменение глобальной переменной в массиве `$GLOBALS`

```
$dinner = 'Curry Cuttlefish';

function hungry_dinner() {
    $GLOBALS['dinner'] .= ' and Deep-Fried Taro';
}

print "Regular dinner is $dinner";
print "\n";
hungry_dinner();
print "Hungry dinner is $dinner";
```

При выполнении кода из примера 5.22 на экран выводится следующий результат:

```
Regular dinner is Curry Cuttlefish
Hungry dinner is Curry Cuttlefish and Deep-Fried Taro
```

В теле функции `hungry_dinner()` элемент массива `$GLOBALS['dinner']` может быть видоизменен таким же образом, как и любая другая переменная, а в конечном итоге изменяется и глобальная переменная `$dinner`. В данном случае элемент массива `$GLOBALS['dinner']` содержит символьную строку, присоединенную к нему с помощью операции сцепления, демонстрируемой в примере 2.19.

В качестве второго способа доступа к глобальной переменной из функции можно воспользоваться ключевым словом `global`. Оно сообщает интерпретатору PHP, что при последующем употреблении именованной переменной в теле функции следует ссылаться по указанному имени на глобальную, а не на локальную переменную. Это, по существу, означает введение переменной в локальную область действия. Применение ключевого слова `global` показано в примере 5.23.

Пример 5.23. Доступ к глобальным переменным с помощью ключевого слова `global`

```
$dinner = 'Curry Cuttlefish';

function vegetarian_dinner() {
    global $dinner;
    print "Dinner was $dinner, but now it's ";
    $dinner = 'Sauteed Pea Shoots';
    print $dinner;
    print "\n";
}
```

```
print "Regular Dinner is $dinner.\n";
vegetarian_dinner();
print "Regular dinner is $dinner";
```

При выполнении кода из примера 5.23 на экран выводится следующий результат:

```
Regular Dinner is Curry Cuttlefish.
Dinner was Curry Cuttlefish, but now it's Sauteed Pea Shoots
Regular dinner is Sauteed Pea Shoots
```

Первый оператор `print` в коде из примера 5.23 отображает неизмененное значение глобальной переменной `$dinner`. Строка кода `global $dinner` в теле функции `vegetarian_dinner()` означает, что любое применение переменной `$dinner` в теле данной функции происходит по ссылке на глобальную, а не на локальную переменную `$dinner`. Таким образом, первый оператор `print` в теле данной функции выводит на экран уже установленное глобальное значение, а в следующей строке кода это глобальное значение изменяется в результате присваивания. Вследствие этого последний оператор `print` за пределами данной функции выводит на экран то же самое измененное значение.

Ключевое слово `global` можно применить сразу к нескольким именам переменных, разделив их запятой, как показано в следующем примере кода:

```
global $dinner, $lunch, $breakfast;
```

Как правило, для доступа к глобальным переменным из функций лучше пользоваться массивом `$GLOBALS`, а не ключевым словом `global`. Ссылка на массив `$GLOBALS` всякий раз напоминает, что приходится иметь дело с глобальной переменной. Ведь применяя ключевое слово `global`, можно очень легко забыть, что оперировать приходится глобальной переменной, если, конечно, речь не идет о написании короткой функции. Это может в конечном итоге вызвать недоумение по поводу неверного поведения кода. И хотя для пользования массивом `$GLOBALS` придется ввести чуть больше кода, наградой за эти труды станет удобочитаемость написанного кода.

Вы, возможно, обратили внимание на некоторую странность приведенных выше примеров, демонстрирующих применение массива `$GLOBALS`. В этих примерах массив `$GLOBALS` применяется в теле функции, но не вводится в локальную область действия, как это делается с помощью ключевого слова `global`. Массив `$GLOBALS` всегда остается в глобальной области действия, применяется ли он в теле функции или за ее пределами. Дело в том, что массив `$GLOBALS` является особого рода предопределенной переменной, называемой *автоглабальной*. Такими переменными можно пользоваться в любом месте программы на PHP, не вводя их в область действия. Они подобны сотруднику, которого знают все в компании (от штаб-квартиры до филиалов), называя его по имени.

Автоглобальные переменные всегда являются массивами, автоматически заполняемыми данными. Они, в частности, содержат данные из переданной на обработку формы, значения из cookie-файла и сведения о текущем сеансе. Особенности авто-глобальных переменных, применяемых в разных контекстах, подробнее описываются в главах 7 и 10.

Соблюдение правил относительно аргументов и возвращаемых значений

На типы и величины аргументов функций и возвращаемых значений не накладывается никаких ограничений, если не указать интерпретатору PHP иное. Так, в функции `countdown()` из

примера 5.9 предполагается, что в качестве аргумента ей передается число. Но этой функции в качестве аргумента можно передать и символьную строку, например `"Caramel"`, и интерпретатор PHP воспримет это как должное.

Объявления типов означают способ наложения ограничений на значения аргументов. Они указывают интерпретатору PHP допустимое значение аргумента, чтобы он мог выдать предупреждение, если предоставлено неверное значение. В табл. 5.1 перечислены различные виды объявлений типов, распознаваемые интерпретаторы PHP, а также версия PHP, в которой внедрена их поддержка.

Таблица 5.1. Объявления типов

Объявление	Правило для аргумента	Минимальная версия PHP
`array`	Должен быть массивом	5.1.0
`bool`	Должен быть логическим значением **`true`** или **`false`**	7.0.0
`callable`	Должен быть чем-то, представляющим функцию или метод, которые можно вызвать[1]	5.4.0
`float`	Должен быть числом с плавающей точкой	7.0.0
`int`	Должен быть целым числом	7.0.0
`string`	Должен быть символьной строкой	7.0.0
Имя класса	Должен быть экземпляром класса (подробнее о классах и их экземплярах см. в главе 6)	5.0.0

[1] Это может быть символьная строка, содержащая допустимое имя функции; двухэлементный массив, где первым элементом является экземпляр объекта, а вторым элементом – символьная строка, содержащая имя метода; или что-нибудь другое. Подробнее об этом см. по адресу `http://www.php.net/language.types.callable`.

При определении функции объявление типа указывается перед именем аргумента. В примере 5.24 демонстрируется функция из примера 5.9 с указанным на своем месте подходящим объявлением типа `int`.

Пример 5.24. Объявление типа аргумента

```
function countdown(int $top) {
    while ($top > 0) {
        print "$top..";
        $top--;
    }
    print "boom!\n";
}

$counter = 5;
countdown($counter);
print "Now, counter is $counter";
```

Единственное отличие в коде из примеров 5.9 и 5.24 состоит в объявлении типа `int` после имени функции `countdown()` и перед аргументом `$top`. Если функции `countdown()` передается допустимое целочисленное значение (например, **`5`**), код из примера 5.24 выполняется верно. А если ей передается значение иного типа, то интерпретатор PHP выдает соответствующее предупреждение. Так, если сделать вызов `countdown ("grunt");` в коде, написанном для версии PHP 7, то появится сообщение об ошибке, аналогичное следующему:

```
PHP Fatal error: Uncaught TypeError: Argument 1 passed to countdown()
must be of the type integer, string given, called in decl-error.php
on line 2 and defined in countdown.php:2
Stack trace:
#0 decl-error.php(2): countdown('grunt')
#1 main
thrown in countdown.php on line 2

[ Неисправимая ошибка PHP: ошибка неперехватываемого типа:
  первый аргумент, передаваемый функции countdown(), должен
  относиться к целочисленному типу, а задана символьная
  строка, ошибка вызвана из файла decl-error.php в строке кода 2
  и определена в файле countdown.php:2
  Трассировка стека:
  #0 decl-error.php(2): countdown('grunt')
  #1 main
  сгенерирована в строке кода 2 из файла countdown.php ]
```

В сообщении об ошибке интерпретатор PHP сообщает о типе ошибки (исключение `TypeError`) с указанием на несоответствие типа аргумента (`1`), переданного функции (`countdown()`), а также на то, что предполагался тип аргумента (`integer`), а на самом деле был передан аргумент другого типа (`string`). Кроме того, предоставляется информация о месте, где был обнаружен проблематичный вызов функции и где была определена вызываемая функция.

В версии PHP 7 исключение `TypeError` может быть перехвачено обработчиком исключений. Подробнее о том, как перехватываются исключения в программе, речь пойдет в разделе "Индикация ошибок с помощью исключений" главы 6[3].

В версии PHP 7 поддерживаются также объявления типов для значений, возвращаемых функцией. Чтобы проверить тип значения, возвращаемого функцией, сначала следует указать знак `:` после скобки `)`, закрывающей список аргументов, а затем объявление возвращаемого типа. Так, в примере 5.25 демонстрируется функция `restaurant_check()` из примера 5.11, дополненная объявлением возвращаемого типа. Если функция из примера 5.25 возвратит все, что угодно, только не значение типа float, интерпретатор PHP сгенерирует исключение `TypeError`.

Пример 5.25. Объявление возвращаемого типа

```
function restaurant_check($meal, $tax, $tip): float {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;
    return $total_amount;
}
```

Выполнение кода из другого файла

В приведенных до сих пор примерах демонстрировался код PHP из отдельных автономных файлов. Любые переменные или функции, использовавшиеся в коде из этих примеров, были определены в том же самом файле. Но по мере разрастания программ их проще организовать, распределив исходный код по разным файлам. Для этой цели служит директива `require`, предписывающая

[3] В прежних версиях PHP нарушения объявлений типов парадоксальным образом обозначались как `Catchable fatal error` (Перехватывая неисправимая ошибка). Такие ошибки приводят к прекращению выполнения программы, если они не обработаны специальным обработчиком ошибок. О том, как реализовать собственный обработчик ошибок в подобных ситуациях, можно подробнее узнать по адресу `http://www.php.net/set_error_handler`.

интерпретатору PHP загрузить код, находящийся в другом файле. Благодаря этому упрощается повторное использование кода во многих местах программы.

Рассмотрим в качестве примера ряд функций, определенных ранее в этой главе. Их можно было бы объединить в одном файле и сохранить под именем `restaurant-functions.php`, как показано в примере 5.26.

По умолчанию объявления скалярных типов в версии PHP 7 совсем не обязательно соблюдаются совершенно строго. Даже при объявлении типов в версии PHP 7 предпринимается попытка преобразовать тип аргумента или возвращаемого значения, которое *фактически* не соответствует объявлению типа, но *могло бы* ему соответствовать. Числовые значения негласно преобразуются в символьные строки, а символьные строки, содержащие числа, – в значения соответствующего числового типа.

Этот нескладный режим по умолчанию можно отключить в конкретном файле, введя в самом его начале строку кода `declare(strict_types=1);`. В таком случае аргументы и значения, возвращаемые любой функцией, вызываемой в данном файле, должны строго соответствовать объявлениям типов. Хотя в качестве аргумента с объявленным типом `float` можно передать и целочисленное значение.

Строгую типизацию нельзя соблюсти глобально. Ее придется объявить в каждом файле, где требуется ее применить.

Пример 5.26. Определение функций в отдельном файле

```
<?php

function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;
    return $total_amount;
}

function payment_method($cash_on_hand, $amount) {
    if ($amount > $cash_on_hand) {
        return 'credit card';
    } else {
        return 'cash';
    }
}

?>
```

Если сохранить код из примера 5.26 в файле `restaurant-functions.php`, то к нему можно обратиться из другого файла с помощью директивы в строке кода `require 'restaurantfunctions .php';` (пример 5.27).

Пример 5.27. Обращение к отдельному файлу в исходном коде

```
require 'restaurant-functions.php';

/* Счет на 25 долларов США плюс налог на добавленную
   стоимость (8,875%) и чаевые (20%) */
```

```
$total_bill = restaurant_check(25, 8.875, 20);

/* Имеется 30 долларов США наличными */
$cash = 30;
print "I need to pay with " . payment_method($cash, $total_bill);
```

Директива в строке кода `require 'restaurantfunctions.php';` из примера 5.27 предписывает интерпретатору PHP прервать чтение команд из текущего файла и перейти к чтению команд из указанного файла `restaurantfunctions.php`, а затем вернуться к текущему файлу и продолжить его обработку. В файле `restaurantfunctions.php` из примера 5.27 определен лишь ряд функций, но файл, загружаемый с помощью директивы `require`, может содержать любой код, допустимый в PHP. Так, если загружаемый подобным образом файл содержит операторы `print`, интерпретатор PHP выведет на экран все, что указано в этих операторах.

Если не удастся найти загружаемый файл, указанный в директиве `require`, или же если такой файл найден, но не содержит допустимый в PHP код, интерпретатор PHP прервет выполнение программы. Код из другого файла можно также загрузить с помощью директивы `include`, но если возникнет затруднение при загрузке файла, то выполнение программы не будет прервано.

Каким образом интерпретатор PHP обнаруживает файлы

Если в директиве **`require`** или **`include`** указан абсолютный путь к файлу (в Mac OS X и Linux он начинается со знака **`/`**, а в Windows – с литеры диска или знака **`\`**), то интерпретатор PHP будет искать файл только в том месте, которое указано в пути.

А если в этих директивах указан относительный путь, начинающийся со знаков **`./`** для обозначения текущего каталога или со знаков **`../`** для обозначения родительского каталога для текущего каталога, то интерпретатор PHP будет искать файл только там, где указано в пути.

Но что касается других предоставляемых имен файлов и путей, то интерпретатор PHP обращается к директиве конфигурации **`include_path`**. Если файл не удастся найти ни в одном из указанных в ней каталогов, интерпретатор PHP проверит каталог, содержащий файл, в котором требуется или включается искомый файл.

Организация исходного кода в отдельных файлах упрощает повторное использование типичных функций и определений, и на этом основывается материал последующих глав данной книги. Кроме того, применение директив `require` и `include` открывает возможность легко воспользоваться библиотеками кода, написанного другими (подробно об этом — в главе 16).

Резюме

В этой главе были рассмотрены следующие вопросы.

- Определение и вызов функций в программах.
- Определение функций с обязательными аргументами.
- Определение функций с необязательными аргументами.
- Возврат значения из функции.
- Представление об области действия переменных.
- Обращение к глобальным переменным в теле функций.
- Представление об объявлениях типов.

- Применение объявлений типов аргументов.
- Применение объявлений типов возвращаемых значений.
- Организация исходного кода PHP в отдельных файлах.

Упражнения

1. Напишите функцию, возвращающую дескриптор `<img />` разметки HTML-страницы. Эта функция должна принимать URL изображения в качестве обязательного аргумента, а также текст надписи, ширину и высоту изображения в качестве необязательных аргументов `alt`, `height` и `width` соответственно.

2. Видоизмените функцию из предыдущего упражнения таким образом, чтобы передавать ей только имя файла в качестве аргумента `URL`. Введите глобальную переменную в теле данной функции, чтобы дополнить указанное имя файла до полного URL. Так, если данной функции передано имя файла `photo.png`, а глобальная переменная содержит путь `/images/`, то атрибут возвращаемого дескриптора `<img>` будет содержать путь `/images/photo.png`. Такая функция упрощает поддержание правильности дескрипторов изображений даже в том случае, если изображения размещаются по новому пути или на другом сервере. Для этого достаточно изменить содержимое глобальной переменной, например, с `/images/` на `http://images.example.com/`.

3. Разместите функцию из предыдущего упражнения в отдельном файле. Затем создайте еще один файл, загружающий первый файл, используя его для вывода на экран ряд дескрипторов `<img />`.

4. Что выводится на экран в приведенном ниже фрагменте кода?

```
<?php

function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    return $meal + $tax_amount + $tip_amount;
}

$cash_on_hand = 31;
$meal = 25;
$tax = 10;
$tip = 10;

while(($cost = restaurant_check($meal, $tax,$tip))
       < $cash_on_hand) {
    $tip++;
    print "I can afford a tip of $tip% ($cost)\n";
}

?>
```

5. Такие цвета веб-палитры, как, например, **`#ffffff`** и **`#cc3399`**, составляются из шестнадцатеричных значений красной, зеленой и синей составляющих цвета. Напишите функцию, принимающую в качестве аргументов десятичные значения красной, зеленой и синей составляющих

цвета и возвращающую символьную строку, содержащую подходящий цвет для применения на веб-странице. Так, если указаны аргументы **255**, **0** и **255** при вызове данной функции, она должна возвратить символьную строку **#ff00ff**. Для написания данной функции может оказаться полезной встроенная функция `dechex()`, документацию на которую можно найти по адресу `http://www.php.net/dechex`.

ГЛАВА 6

Оперирование объектами, объединяя данные и логику

Рассмотренных до сих пор основ организации данных и логики их обработки должно быть достаточно, чтобы сделать немало полезного в PHP. А дополнительный принцип *объектно-ориентированного программирования* (ООП), объединяющий данные и логику, которая ими оперирует, помогает лучше организовать исходный код. В частности, объекты отлично подходят для создания повторно используемых фрагментов кода, и поэтому, умея обращаться с ними, можно эффективнее пользоваться многими существующими дополнениями и библиотеками PHP.

В области программирования *объект* означает структуру, объединяющую данные о каком-нибудь предмете (например, ингредиенты блюда) с действиями над этим предметом (например, выяснением наличия определенных ингредиентов в блюде). Применение объектов в программе обеспечивает организационную структуру для группирования связанных вместе переменных и функций.

Ниже вкратце поясняются основные термины, которые полезно знать для оперирования объектами.

Класс — это шаблон или образец, описывающий переменные и функции для конкретного вида объекта. Например, класс `Entree` может состоять из переменных, содержащих наименование блюда и его ингредиенты. А функции в классе `Entree` определяют порядок приготовления и подачи блюда к столу, а также наличие в нем отдельных ингредиентов.

Метод — это функция, определенная в классе.

Свойство — это переменная, определенная в классе.

Экземпляр — это отдельный пример применения класса. Если на обед подаются три блюда, то в описывающей их программе следует создать три экземпляра класса `Entree`. И хотя каждый из этих экземпляров основывается на одном и том же классе, они имеют внутренние отличия в значениях их свойств. Методы в каждом экземпляре состоят из одинаковых инструкций, но производят разные результаты, поскольку каждый из них полагается на конкретные значения свойств в их экземплярах. Создание нового экземпляра класса называется *получением экземпляра объекта*.

Конструктор — это специальный метод, который выполняется автоматически при получении экземпляра объекта. Как правило, в конструкторах устанавливаются свойства объектов и выполняются прочие служебные операции, подготавливающие объект к применению.

Статический метод — это особая разновидность метода, который можно вызывать, не получая экземпляр класса. Статические методы не зависят от значений свойств конкретного экземпляра.

Основы организации объектов

В примере 6.1 определяется класс Entree, представляющий блюдо.

Пример 6.1. Определение класса

```
class Entree {
    public $name;
    public $ingredients = array();
    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

В примере 6.1 определение класса начинается со специального ключевого слова `class`, после которого следует имя, присваиваемое классу. Все, что заключено в фигурные скобки после имени класса, относится к его определению, в том числе свойства и методы класса. У данного класса имеются два свойства, `$name` и `$ingredients`, а также один метод `hasIngredient()`. Ключевое слово `public` сообщает интерпретатору PHP, из каких частей программы разрешается доступ к конкретному свойству или методу, в объявлении которого указано это ключевое слово. Подробнее об этом речь пойдет далее, в разделе "Доступность свойств и методов".

Метод `hasIngredient()` очень похож на определение обычной функции, но его тело содержит нечто новое. Это специальная переменная `$this`, ссылающаяся на тот экземпляр класса, из которого вызывается метод. В примере 6.2 наглядно показано, как это происходит на практике с двумя разными экземплярами.

Пример 6.2. Создание и применение объектов

```
// создать экземпляр и присвоить его переменной $soup
$soup = new Entree;
// установить свойства экземпляра в переменной $soup
$soup->name = 'Chicken Soup';
$soup->ingredients = array('chicken', 'water');

// создать отдельный экземпляр и присвоить его
// переменной $sandwich
$sandwich = new Entree;
// установить свойства экземпляра в переменной $sandwich
$sandwich->name = 'Chicken Sandwich';
$sandwich->ingredients = array('chicken', 'bread');

foreach (['chicken','lemon','bread','water'] as $ing) {
    if ($soup->hasIngredient($ing)) {
        print "Soup contains $ing.\n";
    }
    if ($sandwich->hasIngredient($ing)) {
        print "Sandwich contains $ing.\n";
    }
}
```

Операция new возвращает новый объект типа `Entree`, поэтому переменные `$soup` и `$sandwich` в коде из примера 6.2 ссылаются на разные экземпляры класса `Entree`. Операция "стрелка" (`->`) служит для доступа к свойствам (т.е. к переменным) и методам (т.е. к функциям) в объекте. Так,

для доступа к свойству достаточно указать операцию “стрелка” после имени объекта, а после этой операции — имя свойства. Для вызова метода следует указать операцию “стрелка” аналогичным образом, а после нее — имя метода и круглые скобки, обозначающие вызов функции вместе с аргументами, если таковые имеются.

Следует, однако, иметь в виду, что операция “стрелка”, предназначенная для доступа к свойствам и методам, отличается своим обозначением от операции, разделяющей ключи и значения в языковой конструкции `array()` или `foreach()`. Для обозначения доступа к элементам массива служат знаки **`=>`**, тогда как для обозначения доступа к элементам объекта — знаки **`->`**.

Значение присваивается свойству таким же образом, как и любой другой переменной, но для обозначения имени свойства используется синтаксис операции “стрелка”. В частности, выражение `$soup->name` означает свойство `name` из объекта, экземпляр которого хранится в переменной `$soup`, а выражение `$sandwich->ingredients` — свойство `ingredients` из объекта, экземпляр которого хранится в переменной `$sandwich`.

В цикле, организуемом с помощью языковой конструкции `foreach()`, вызывается метод `hasIngredient()` из каждого объекта. Этому методу передается наименование ингредиента, а из него возвращается признак наличия или отсутствия данного ингредиента в списке, хранящемся в объекте. Специальная переменная `$this` действует в этом методе следующим образом. При вызове `$soup->hasIngredient()` переменная `$this` ссылается в теле метода `hasIngredient()` на экземпляр объекта, хранящийся в переменной `$soup`. А при вызове `$sandwich->hasIngredient()` переменная `$this` ссылается на экземпляр объекта, хранящийся в переменной `$sandwich`. Переменная `$this` не всегда ссылается на один и тот же экземпляр объекта. Напротив, она ссылается на тот экземпляр, из которого вызывается метод. Это означает, что при выполнении кода из примера 6.2 на экран выводится следующий результат:

```
Soup contains chicken.
Sandwich contains chicken.
Sandwich contains bread.
Soup contains water.
```

Если аргумент `$ing` принимает строковое значение **`chicken`** в коде из примера 6.2, то в результате обоих вызовов, `$soup->hasIngredient($ing)` и `$sandwich->hasIngredient($ing)`, возвращается логическое значение `true`. Свойства `$ingredients` обоих объектов содержат элемент массива со значением **`chicken`**. Но только свойство `$soup->ingredients` содержит элемент массива со значением **`water`**, а свойство `$sandwich->ingredients` — элемент массива со значением **`bread`**. А элемент массива со значением **`lemon`** отсутствует в свойствах `ingredients` обоих объектов.

Классы могут также содержать статические методы, в которых нельзя пользоваться специальной переменной `$this`, поскольку они выполняются не в контексте конкретного экземпляра объекта, а в самом классе. Статические методы удобны для реализации поведения, соответствующего назначению класса, но ни одному из объектов. В примере 6.3 демонстрируется ввод в класс `Entree` статического метода, возвращающего возможные размеры блюд.

Пример 6.3. Определение статического метода

```
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }

    public static function getSizes() {
```

```
        return array('small','medium','large');
    }
}
```

Объявление статического метода в примере 6.3 очень похоже на объявления других методов, а отличается оно добавлением ключевого слова `static` перед ключевым словом `function`. Чтобы вызвать статический метод, следует вставить знаки **::** между именами класса и метода вместо знаков **->**, как показано в примере 6.4.

Пример 6.4. Вызов статического метода

```
$sizes = Entree::getSizes();
```

Конструкторы

У класса может быть специальный метод, называемый *конструктором* и выполняемый при создании объекта. Конструкторы обычно выполняют установочные и служебные операции для подготовки объекта к применению. В качестве примера можно внести изменения в класс `Entree`, назначив для него конструктор. Такой конструктор принимает следующие аргументы: наименование блюда и список его ингредиентов. Передавая эти значения конструктору, можно избежать установки свойств после создания объекта. Метод-конструктор класса в PHP всегда называется `__construct()`. В примере 6.5 демонстрируется измененный класс вместе с его методом-конструктором.

Пример 6.5. Инициализация объекта в конструкторе

```
class Entree {
    public $name;
    public $ingredients = array();

    public function __construct($name, $ingredients) {
        $this->name = $name;
        $this->ingredients = $ingredients;
    }

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

Как показано в примере 6.5, метод-конструктор `__construct()` принимает два аргумента и присваивает их значения свойствам класса. Ради удобства аргументам этого метода-конструктора присвоены такие же имена, как и у свойств создаваемого объекта, хотя интерпретатор PHP этого и не требует. А в теле данного конструктора обращение к конкретному экземпляру создаваемого объекта осуществляется по ссылке `$this`.

Вызывая операцию `new`, значения аргументов следует указывать после нее в круглых скобках, а для передачи аргументов конструктору — рассматривать имя класса как имя функции. В примере 6.6 демонстрируется применение конструктора класса `Entree` для создания экземпляров объектов `$soup` и `$sandwich`, аналогичных упоминавшимся выше.

Пример 6.6. Вызов конструктора

```
// Суп, его название и ингредиенты
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// Сэндвич, его название и ингредиенты
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));
```

Вызов конструктора в операции `new` — это лишь часть того, что интерпретатор PHP делает для создания нового объекта, но сам конструктор не создает объект. Это означает, что метод-конструктор не возвращает значение и не может использовать возвращаемое значение для уведомления о неверном ходе выполнения кода, поскольку это задача *исключений*, рассматриваемых в следующем разделе.

Индикация ошибок с помощью исключений

А что произойдет в коде из примера 6.5, если в качестве аргумента `$ingredients` передать конструктору не массив, а что-нибудь другое? В том виде, в каком код написан в примере 6.5, ничего не произойдет! Свойству `$this->ingredients` создаваемого объекта будет присвоено значение аргумента `$ingredients`, каким бы оно ни было. Но если это не массив, то при вызове метода `hasIngredient()` возникнет затруднение, поскольку в данном методе предполагается, что свойство `$ingredients` содержит массив.

Конструкторы вполне пригодны для проверки достоверности типов предоставляемых аргументов, но им нужно каким-то образом сообщить об ошибке, если она будет обнаружена. И для этой цели служит *исключение* — специальный объект, который может быть использован для указания на то, что произошло нечто исключительное. Возникновение исключения приводит к прерыванию работы интерпретатора PHP и его установке на другой путь выполнения кода.

В примере 6.7 демонстрируется конструктор класса `Entree`, в который внесены изменения для генерирования исключения в том случае, если аргумент `$ingredients` не содержит массив. (*Генерирование* исключения означает применение исключения для уведомления интерпретатора PHP о неверном ходе выполнения кода.)

Пример 6.7. Генерирование исключения

```
class Entree {
    public $name;
    public $ingredients = array();
    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
            throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

Исключения представлены в классе Exception. В качестве первого аргумента конструктору класса Exception передается символьная строка, описывающая причину сбоя в коде. Таким образом, в строке кода

```
throw new Exception('$ingredients must be an array');
```

создается новый объект типа `Exception`, который затем передается языковой конструкции `throw` с целью прервать работу интерпретатора PHP.

Если аргумент `$ingredients` содержит массив, то код выполняется, как и прежде, а иначе генерируется исключение. В примере 6.8 демонстрируется создание объекта типа `Entree` с неверно указанным аргументом `$ingredients`.

Пример 6.8. Создание условий для генерирования исключения

```
$drink = new Entree('Glass of Milk', 'milk');
if ($drink->hasIngredient('milk')) {
    print "Yummy!";
}
```

При выполнении кода из примера 6.8 на экран выводится сообщение об ошибке, аналогичное приведенному ниже, при условии, что код находится в файле `exception-use.php`, а определение класса `Entree` — в файле `constructexception.php`.

```
PHP Fatal error: Uncaught Exception: $ingredients must be an array
in construct-exception.php:9
Stack trace:
#0 exception-use.php(2): Entree->_construct('Glass of Milk', 'milk')
#1 main
thrown in construct-exception.php on line 9

[ Неисправимая ошибка PHP: ошибка неперехватываемого типа:
 аргумент $ingredients должен содержать массив в ƒайле
 construct-exception.php:9
 Трассировка стека:
 #0 exception-use.php(2):Entree->_construct('Glass of Milk', 'milk')
 #1 main
 сгенерировано в строке кода 9 из ƒайла construct-exception.php ]
```

В приведенном выше результате вывода ошибок можно выявить две разные составляющие. Первая из них содержит следующее сообщение об ошибке, получаемое из интерпретатора PHP:

```
PHP Fatal error: Uncaught Exception: $ingredients must be an array
in construct-exception.php:9
```

Это сообщение означает, что в строке кода 9 из файла `construct-exception.php`, где определен класс `Entree`, было сгенерировано исключение. А поскольку дополнительный код для обработки этого исключения отсутствует (далее будет показано, каким образом такой код составляется), то такое исключение называется *неперехватываемым* и приводит к тому, что интерпретатор PHP сразу прекращает свою работу. Это *неисправимая* ошибка, моментально прерывающая нормальное выполнение программы.

И второй составляющей в приведенном выше результате вывода ошибок является *трассировка стека* — перечень всех функций, которые были активны в момент прерывания работы интерпретатора PHP. В данном случае это лишь конструктор класса `Entree`, вызываемый в операции `new Entree`. Строка `main` в трассировке стека представляет первый уровень выполнения программы прежде остального. Этот уровень всегда находится в нижней части любой трассировки стека.

Конечно, совсем не плохо, что вызов метода `hasIngredient()` удалось предотвратить таким образом, чтобы он не оперировал ингредиентами, не находящимися в массиве. Но полная остановка программы со столь резким сообщением об ошибке — это неоправданная крайность. Обратной

стороной медали генерирования исключений является их *перехват*, прежде чем они достигнут интерпретатора PHP и будут обработаны в нем как неисправимые ошибки.

Чтобы самостоятельно обработать исключение, необходимо выполнить следующие действия.

1. Разместить в блоке оператора `try` код, в котором может быть сгенерировано исключение.
2. Разместить блок оператора `catch` после кода, в котором может быть сгенерировано исключение, чтобы обработать исключение и устранить возникшую ошибку.

В примере 6.9 демонстрируется порядок ввода в исходный код блоков операторов `try` и `catch` для обработки исключения.

Пример 6.9. Обработка исключения

```
try {
    $drink = new Entree('Glass of Milk', 'milk');
    if ($drink->hasIngredient('milk')) {
        print "Yummy!";
    }
} catch (Exception $e) {
    print "Couldn't create the drink: " . $e->getMessage();
}
```

В коде из примера 6.9 блоки операторов `try` и `catch` действуют совместно. Операторы в блоке `try` выполняются по очереди до тех пор, пока не возникнет исключение. В этом случае интерпретатор PHP перейдет сразу к блоку оператора `catch`, присвоив переменной `$e` созданный при этом объект типа `Exception`. А в коде, находящемся в блоке `catch`, автоматически вызывается метод `getMessage()` из класса `Exception` для извлечения текста сообщения, заданного для исключения при его создании. При выполнении кода из примера 6.9 на экран выводится следующий результат:

```
Couldn't create the drink: $ingredients must be an array
```

Расширение объектов

Объекты удобны для организации кода, в частности, тем, что они подлежат *подклассификации*, которая позволяет повторно использовать класс, дополняя его специальными функциональными возможностями. Подкласс, иногда еще называемый *порожденным*, наследует все методы и свойства от существующего класса, называемого *родительским*, а далее он изменяет их или вводит собственные свойства и методы.

Рассмотрим в качестве примера блюдо, которое не является простым, а состоит из ряда других блюд, например, тарелки супа и сендвича. Это положение можно было бы смоделировать в существующем классе `Entree`, отнеся суп и сендвич в числу ингредиентов или перечислив все ингредиенты супа и сендвича в данном составном блюде. Но ни одно из этих решений не является идеальным. Ведь суп и сендвич, по существу, не являются ингредиентами, а для повторного перечисления ингредиентов блюда придется обновить код во многих местах при изменении любого ингредиента.

Данную задачу можно решить более изящно, создав подкласс, производный от класса `Entree`. Он должен получать экземпляры объектов класса `Entree` в качестве ингредиентов, видоизменив метод `hasIngredient()` для проверки этих экземпляров на наличие в них ингредиентов. Определение этого производного класса `ComboMeal` приведено в примере 6.10.

Пример 6.10. Расширение класса ***Entree***

```
class ComboMeal extends Entree {
    public function hasIngredient($ingredient) {
        foreach ($this->ingredients as $entree) {
            if ($entree->hasIngredient($ingredient)) {
                return true;
            }
        }
        return false;
    }
}
```

В коде из примера 6.10 после имени класса `ComboMeal` следует предложение `extends Entree`, которое сообщает интерпретатору PHP, что класс `ComboMeal` должен наследовать от класса `Entree` все его методы и свойства. Для интерпретатора PHP это означает переопределение класса `Entree` в определении класса `ComboMeal`. Хотя это делается автоматически, избавляя программиста от рутинной работы, и ему остается лишь внести изменения и дополнения в фигурных скобках определения класса `ComboMeal`. В данном примере единственным изменением является новый метод `hasIngredient()`. Вместо того чтобы исследовать свойство `$this->ingredients` как массив, в данном классе оно рассматривается как массив объектов типа `Entree`, и для каждого из этих объектов вызывается метод `hasIngredient()`. Если в результате любого из таких вызовов возвращается логическое значение `true`, это означает, что одно из блюд в составном блюде содержит указанный ингредиент, а следовательно, метод `hasIngredient()` из класса `ComboMeal` возвращает логическое значение `true`. Если же после перебора всех блюд логическое значение `true` вообще не возвращается, то данный метод возвратит логическое значение `false`, а это означает, что ни одно из блюд не содержит ингредиенты. Применение подкласса `ComboMeal` наглядно демонстрируется в примере 6.11.

Пример 6.11. Применение подкласса

```
// Суп, его название и ингредиенты
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// Сендвич, его название и ингредиенты
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));

// Составное блюдо
$combo = new ComboMeal('Soup + Sandwich', array($soup, $sandwich));

foreach (['chicken','water','pickles'] as $ing) {
    if ($combo->hasIngredient($ing)) {
        print "Something in the combo contains $ing.\n";
    }
}
```

Суп и сэндвич содержат ингредиент **chicken** (цыпленок). Но в то же время суп содержит ингредиент **water** (вода), и в нем отсутствует ингредиент **pickles** (маринованные огурцы). Поэтому при выполнении кода из примера 6.11 на экран выводится следующий результат:

```
Something in the combo contains chicken.
Something in the combo contains water.
```

Такой подход вполне пригоден, но он не гарантирует, что аргументы, передаваемые конструктору класса ComboMeal, действительно являются объектами класса Entree. Ведь если они таковыми не являются, то вызов для них метода hasIngredient() может привести к ошибке. Чтобы устранить этот недостаток, необходимо ввести в класс ComboMeal специальный конструктор, проверяющий данное условие и вызывающий обычный конструктор класса Entree с целью правильно установить свойства его объекта. Вариант класса ComboMeal с таким конструктором приведен в примере 6.12.

Пример 6.12. Ввод конструктора в подкласс

```
class ComboMeal extends Entree {

    public function __construct($name, $entrees) {
        parent::__construct($name, $entrees);
        foreach ($entrees as $entree) {
            if (! $entree instanceof Entree) {
              throw new Exception(
                'Elements of $entrees must be Entree objects');
            }
        }
    }

    public function hasIngredient($ingredient) {
        foreach ($this->ingredients as $entree) {
            if ($entree->hasIngredient($ingredient)) {
                return true;
            }
        }
        return false;
    }
}
```

В конструкторе из примера 6.12 применяется специальный синтаксис parent::__construct() для ссылки на конструктор из класса Entree. Аналогично ссылке $this, этот синтаксис имеет особое назначение в методах объектов. В частности, ключевое слово parent означает ссылку на родительский класс. А поскольку класс ComboMeal расширяет класс Entree, то при вызове parent::__construct() в порожденном классе ComboMeal происходит обращение к конструктору __construct() родительского класса Entree.

Не следует забывать, что конструктор родительского класса нужно вызывать в конструкторах подкласса явным образом. Так, если опустить вызов parent::__construct(), родительский конструктор вообще не будет вызван, и его важное, вероятно, поведение так и не будет воспроизведено в интерпретаторе PHP. В данном случае конструктор класса Entree вызывается, чтобы проверить, содержит ли аргумент $ingredients массив, а затем установить свойства $name и $ingredients.

После вызова parent::__construct() в конструкторе класса ComboMeal проверяется, является ли каждый предоставляемый ингредиент составного блюда объектом класса Entree. И для этой цели применяется операция instanceof. Условное выражение $entree instanceof Entree оказывается истинным, если переменная $entree ссылается на экземпляр объекта класса Entree.[1] Если же любой из предоставляемых ингредиентов, которые для класса ComboMeal на самом деле являются блюдами, не является объектом класса Entree, то в рассматриваемом здесь коде генерируется исключение.

[1] Результат выполнения операции instanceof также оказывается истинным, если объект в левой части данной операции относится к подклассу, производному от класса, имя которого указывается в правой ее части. Рассматриваемый здесь код пригоден, например, для одних составных блюд, состоящих из других составных блюд.

Доступность свойств и методов

Конструктор класса ComboMeal из примера 6.12 делает немало для того, чтобы класс ComboMeal получал экземпляры класса Entree только в качестве ингредиентов составного блюда. Но что происходит дальше? В последующем коде значение свойства $ingredients может быть изменено на все, что угодно: массив любых объектов, кроме класса Entree, число или даже логическое значение false.

Чтобы воспрепятствовать такому нежелательному поведению, следует изменить *доступность* свойств. В частности, их можно объявить закрытыми (private) или защищенными (protected), а не открытыми (public). Два первых модификатора доступа не оказывают влияния на то, что код класса может делать в нем самом, например, читать и записывать значения его свойств. В частности, модификатор доступа private препятствует любому коду за пределами класса обращаться к его свойству. А модификатор доступа protected означает, что к свойству класса можно обращаться за его пределами только из кода его подклассов.

В примере 6.13 демонстрируется видоизмененный вариант класса Entree, в котором свойство $name объявлено как private, а свойство $ingredients — как protected.

Пример 6.13. Изменение доступности свойств

```
class Entree {
    private $name;
    protected $ingredients = array();

    /* Свойство $name объявлено закрытым, и поэтому ниже
       предоставляется метод для чтения его значения */
    public function getName() {
        return $this->name;
    }

    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
          throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

С одной стороны, свойство $name объявлено в коде из примера 6.13 как private, и поэтому оно недоступно для чтения или записи за пределами класса Entree. Но в класс Entree введен метод getName() для чтения значения свойства $name из кода за пределами данного класса. Это так называемый *метод доступа*. Он обеспечивает доступ к свойству, который иначе запрещен. В данном случае сочетание модификатора доступа private с методом доступа, возвращающим значение свойства, позволяет любому коду прочитать значение свойства $name, но не изменить его, как только оно будет установлено в самом классе Entree.

С другой стороны, свойство $ingredients объявлено как protected, и поэтому оно доступно из подклассов, производных от класса Entree. Этим обеспечивается правильность функционирования метода hasIngredient() в классе ComboMeal.

Те же самые модификаторы доступа применяются и к методам. В частности, методы, объявленные как `public`, можно вызывать из любого кода, а методы, объявленные как `private`, — только из другого кода в том же самом классе. И, наконец, методы, объявленные как `protected`, можно вызывать только из другого кода в том же самом классе или из его подклассов.

Пространства имен

Начиная с версии 5.4, интерпретатор PHP позволяет организовывать код в *пространствах имен*. В пространствах имен можно группировать связанный вместе код, исключая конфликты имен классов, одинаково названных разными их авторами[2].

Уметь правильно пользоваться пространствами имен очень важно для внедрения в свои программы пакетов, разработанных другими. В этом разделе дается общее представление о синтаксисе пространств имен. А в главе 16 подробно описывается система управления пакетами Composer.

Пространства имен можно рассматривать в качестве контейнера, где допускается хранить определения классов или даже другие пространства имен. Это делается ради удобства, а не ради предоставления новых функциональных возможностей. Когда в исходном коде встречается ключевое слово `namespace` и путь к классу, это означает вхождение в пространство имен PHP.

Чтобы определить класс в конкретном пространстве имен, достаточно указать ключевое слово `namespace` с наименованием пространства имен в самом начале исходного файла. И тогда определение класса далее в исходном файле будет отнесено к указанному пространству имен. Так, в примере 6.14 класс `Fruit` определяется в пространстве имен `Tiny`.

Пример 6.14. Определение класса в пространстве имен

```
namespace Tiny;

class Fruit {
    public static function munch($bite) {
        print "Here is a tiny munch of $bite.";
    }
}
```

Чтобы воспользоваться классом в пространстве имен, необходимо внедрить это пространство имен в порядок обращения к классу. Самый простой способ сделать это однозначно — указать сначала знак `\`, обозначающий пространство имен верхнего уровня, затем имя пространства имен, в котором находится класс, далее еще один знак `\` и имя класса. Например, чтобы вызвать статический метод `munch()` из класса `Fruit`, определенного в примере 6.14, достаточно написать следующую строку кода:

```
\Tiny\Fruit::munch("banana");
```

Одни пространства имен могут также состоять из других пространств имен. Если бы код из примера 6.14 начинался со строки `namespace Tiny\Eating;`, то обратиться к классу `Fruit` следовало бы следующим образом: `\Tiny\Eating\Fruit`.

Если не указать в начале знак `\`, то обращение к классу осуществляется в *текущем пространстве имен*, т.е. в том пространстве имен, которое оказывается активным в момент обращения. Если пространство имен не объявлено в самом начале исходного файла PHP, то к верхнему уровню относится текущее пространство имен, а имена классов ведут себя, как обычные имена классов, встречавшиеся в приведенных ранее примерах кода. Но ключевое слово `namespace` изменяет текущее пространство имен. Так, в объявлении `namespace Tiny;` текущее пространство имен изменяется

[2] Пространства имен охватывают также функции и прочие языковые средства, не относящиеся к классам, но в этом разделе они рассматриваются лишь в связи с классами.

на Tiny. Именно поэтому определение class Fruit в коде из примера 6.14 приводит к тому, что класс Fruit размещается в пространстве имен Tiny.

Но это означает также, что ссылка на любой *другой* класс в том же самом исходном файле разрешается относительно пространства имен Tiny. Метод из класса Tiny\Fruit, содержащий следующую строку кода:

```
$soup = new Entree('Chicken Soup', array('chicken','water'));
```

предписывает интерпретатору PHP искать класс Entree в пространстве имен Tiny. Это все равно, что написать такую строку кода:

```
$soup = new \Tiny\Entree('Chicken Soup', array('chicken','water'));
```

Чтобы сделать однозначным обращение к классу в пространстве имен верхнего уровня, необходимо указать знак \ перед именем этого класса. Безусловно, набирать многократно знаки \ в исходном коде довольно утомительно. Поэтому интерпретатор PHP предоставляет ради упрощения ключевое слово use. В примере 6.15 наглядно показано, как пользоваться ключевым словом use.

Пример 6.15. Применение ключевого слова use

```
use Tiny\Eating\Fruit as Snack;

use Tiny\Fruit;

// Приведенная ниже строка кода равнозначна
// такой строке кода: \Tiny\Eating\Fruit::munch();
Snack::munch("strawberry");

// Приведенная ниже строка кода равнозначна
// такой строке кода: \Tiny\Fruit::munch();
Fruit::munch("orange");
```

В строке кода use Tiny\Eating\Fruit as Snack; из примера 6.15 интерпретатору PHP предписывается считать имя Snack в остальной части исходного файла как \Tiny\Eating\Fruit. Если не указать предложение as, интерпретатор PHP выведет "псевдоним" класса из последнего элемента, определенного для применения с помощью ключевого слова use. Таким образом, в строке кода use Tiny\Fruit; интерпретатору PHP предписывается считать имя Fruit в остальной части исходного файла как \Tiny\Fruit.

Такого рода объявления с помощью ключевого слова use особенно удобны для многих современных библиотек и каркасов PHP, где различные классы размещаются в пространствах и подпространствах имен. Введя несколько строк кода с ключевым словом use в самом начале исходного файла, можно превратить многословные обращения к классам вроде \Symfony\Component\Http Foundation\Response в более краткие наподобие Response.

Резюме

В этой главе были рассмотрены следующие вопросы:

- Представление о том, как объекты помогают организовать код.
- Определение класса с методами и свойствами.
- Создание объекта с помощью операции new.

- Доступ к методам и свойствам с помощью операции “стрелка”.
- Определение и вызов статического (`static`) метода.
- Инициализация объекта в конструкторе.
- Генерирование исключений для индикации ошибок.
- Перехват исключений для обработки ошибок.
- Расширение класса производным от него подклассом.
- Управление доступом к свойствам и методами с помощью модификаторов доступа.
- Организация кода в пространствах имен.

Упражнения

1. Создайте класс `Ingredient`. Каждый экземпляр этого класса должен представлять отдельный ингредиент блюда, а также отслеживать наименование ингредиента и его стоимость.
2. Введите в свой новый класс `Ingredient` метод, изменяющий стоимость ингредиента блюда.
3. Создайте подкласс, производный от представленного в этой главе класса `Entree`. Этот подкласс должен принимать объекты типа `Ingredient` вместо символьной строки с наименованиями ингредиентов для их обозначения. Введите в этот подкласс метод, возвращающий общую стоимость блюда.
4. Разместите свой класс `Ingredient` в собственном пространстве имен и внесите изменения в другой код, где применяется класс `Ingredient`, чтобы этот код функционировал надлежащим образом.

ГЛАВА 7

Создание веб-форм для обмена данными с пользователями

Обработка форм является важной составляющей практически любого веб-приложения. *Формы* определяют порядок взаимодействия пользователей с веб-сервером, включая регистрацию по новой учетной записи, поиск в форуме всех публикаций на конкретную тему, восстановление утраченного пароля, обнаружение местоположения ближайшего ресторана или обувного магазина или приобретение книг.

Формы применяются в программах на PHP в два этапа. На первом этапе форма отображается. С этой целью выполняется HTML-разметка формы дескрипторами для описания соответствующих элементов пользовательского интерфейса, в том числе текстовых полей, флажков и экранных кнопок. Если вы не знакомы с особенностями HTML-разметки, требующейся для создания форм, рекомендуется прочитать книгу *HTML5 и CSS3 для чайников*, упоминавшуюся предисловии к данной книге.

Когда страница с формой появится перед пользователем, он должен ввести информацию, запрашиваемую в форме, а затем щелкнуть на кнопке Submit (Передать) или нажать клавишу <Enter>, чтобы отправить форму обратно на веб-сервер. Обработка информации из переданной формы и составляет второй этап применения формы.

В примере 7.1 демонстрируется страница, обращающаяся к пользователю с приветствием `"Hello"`. Если эта страница загружается в ответ на передачу формы, то на ней отображается данное приветствие. В противном случае на странице отображается форма, в которой пользователь должен ввести свое имя и затем передать форму на обработку.

Пример 7.1. Отображение приветствия на странице

```
if ('POST' == $_SERVER['REQUEST_METHOD']) {
    print "Hello, ". $_POST['my_name'];
} else {
    print<<<_HTML_
    <form method="post" action="$_SERVER[PHP_SELF]">
       Your name: <input type="text" name="my_name" >
    <br>
    <input type="submit" value="Say Hello">
    </form>
    _HTML_;
}
```

Напомним, что порядок взаимодействия клиента и сервера пояснялся в главе 1. На рис. 7.1 наглядно показано взаимодействие клиента и сервера, требующееся для отображения и обработки

формы из примера 7.1. Первая пара “запрос-ответ” приводит к отображению формы в окне браузера. Во второй паре “запрос-ответ” сервер обрабатывает данные из переданной формы, а браузер отображает результаты их обработки.

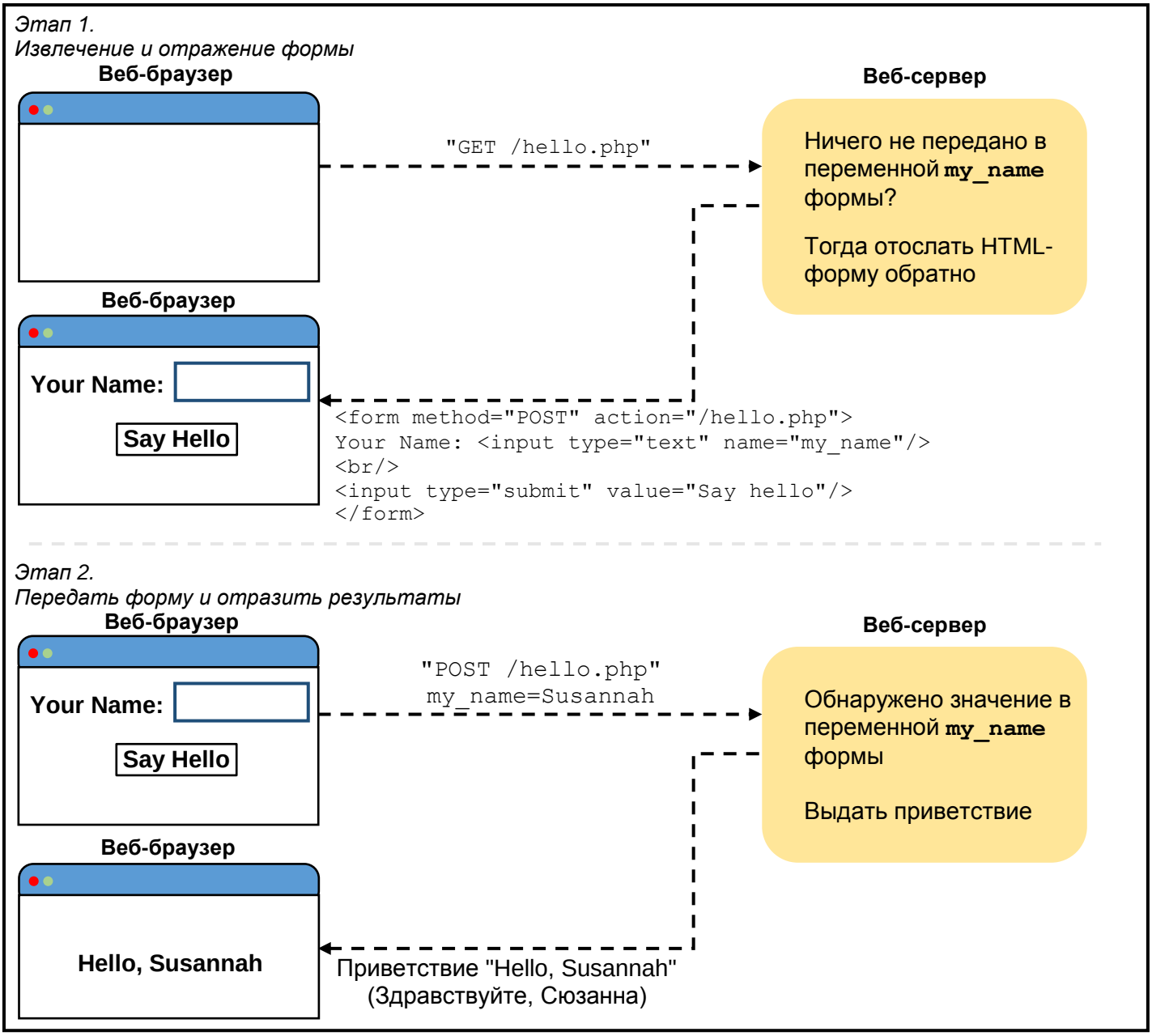


Рис. 7.1: Отображение и обработка простой формы

В ответ на первый запрос посылается некоторая HTML-разметка формы. На рис. 7.2 показано, что отображается в окне браузера при получении такого ответа.

Рис. 7.2: Простая форма

В ответ на второй запрос посылается результат обработки данных из переданной формы. На рис. 7.3 показан результат, выводимый после обработки переданной формы, где в текстовом поле было введено имя пользователя **Susannah**.

Рис. 7.3: Результат обработки переданной формы

Порядок обработки формы, приведенный на рис. 7.1, весьма характерен для простых программ и состоит в следующем: если данные в заполняемой форме переданы, обработать их, а иначе — вывести форму снова. При построении элементарной формы код для отображения и обработки формы следует размещать на одной и той же странице, чтобы упростить синхронизацию формы и связанной с ней логики. Когда же дело дойдет до более сложных форм далее в этой главе, создаваемая форма будет разделена на отдельные файлы с логикой отображения и обработки.

Форма, переданная на обработку, посылается обратно по тому же URL, по которому она первоначально запрашивалась, потому что в атрибуте `action` дескриптора `<form>` указана специальная переменная `$_SERVER['PHP_SELF']`. Авто-глобальный массив `$_SERVER` содержит самую

разную информацию о сервере и текущем запросе, обрабатываемом интерпретатором PHP, а элемент PHP_SELF массива $_SERVER — путевую часть URL текущего запроса. Так, если сценарий PHP доступен по адресу http://www.example.com/store/catalog.php, то элемент массива $_SERVER['PHP_SELF'][1] содержит путь /store/catalog.php к файлу сценария на странице, доступной по данному URL.

Кроме того, простая форма упрощает применение элемента массива $_SERVER['REQUEST_METHOD'], содержащего метод сетевого протокола HTTP, применяемый на веб-сервере для запроса текущей страницы. Для обычных страниц это всегда метод GET или POST. Как правило, метод GET означает извлечение обычной страницы, а метод POST — передачу формы. Значение элемента массива $_SERVER['REQUEST_METHOD'] всегда указывается прописными буквами независимо от того, каким образом задано значение атрибута action в дескрипторе <form>. Таким образом, проверка элемента массива $_SERVER['REQUEST_METHOD'] на наличие в нем метода POST позволяет выяснить, была ли передана на обработку форма, или же это был запрос обычной страницы.

Автоглобальный массив $_POST содержит данные из формы, переданной на обработку. Ключи в этом массиве обозначают элементы формы, а соответствующие им значения равны значениям в элементах формы. Так, если ввести имя в текстовом поле формы из примера 7.1 и щелкнуть на кнопке передачи этой формы на обработку, элементу массива $_POST['my_name'] будет присвоено то значение, которое было введено в текстовом поле, поскольку в атрибуте name разметки данного поля указано значение **my_name**.

Структура кода из примера 7.1 служит прочным основанием для рассмотрения особенностей обработки форм в этой главе. Но ей присущ следующий недостаток: выводить входные данные из внешнего источника в необработанном виде опасно, как это, например, делается со значением, введенным в текстовом поле **my_name** из переданной на обработку формы, в следующей строке кода:

```
print "Hello, ". $_POST['my_name'];
```

Ведь данные, поступающие в программу из внешнего источника (например, параметр, введенный в переданной на обработку форме), могут содержать встроенный код HTML-разметки или сценария JavaScript. В разделе “HTML и JavaScript” далее в этой главе поясняется, как обезопасить программу, предварительно очистив входные данные из внешнего источника.

В остальной части этой главы подробно рассматриваются различные особенности обработки форм. Так, в разделе “Доступ к параметрам формы” описывается порядок обработки различных видов данных, вводимых в форме, в том числе параметров формы, которые могут принимать разные значения. Затем в разделе “Обработка форм с помощью функций” рассматривается гибкая структура обработки форм на основе функций, упрощающая некоторые задачи сопровождения форм. Такая структура позволяет также проверять данные из переданной на обработку формы, чтобы убедиться в том, что они не содержат ничего неожиданного. Далее в разделе “Проверка достоверности данных” поясняются разные способы проверки данных из переданной на обработку формы. А в разделе “Отображение устанавливаемых по умолчанию значений” показано, как снабжать элементы формы устанавливаемыми по умолчанию значениями и сохранять введенные пользователем значения при повторном отображении формы. И, наконец, в разделе “Собирая все вместе” демонстрируется полноценная форма, вобравшая в себя все, что было рассмотрено прежде в этой главе: организация структуры обработки формы на основе функций, проверка достоверности данных и отображение сообщений об ошибках, установка значений по умолчанию и сохранение введенных пользователем данных, а также обработка переданных данных.

[1] Как пояснялось при рассмотрении примера 1.4, ключ PHP_SELF элемента массива $_SERVER указывается во встраиваемом документе без кавычек для правильной вставки его значения в документ.

Полезные серверные переменные

Помимо переменных PHP_SELF и REQUEST_METHOD, автоглобальный массив $_SERVER содержит целый ряд полезных элементов, предоставляющих информацию о веб-сервере и текущем запросе. Некоторые из этих элементов перечислены в табл. 7.1.

Таблица 7.1. Элементы в массиве $_SERVER

Элемент	Пример	Описание
QUERY_S TRING	category=kitchen&price=5	Часть URL после знака вопроса, где указываются параметры URL. В следующем URL демонстрируется пример указания строки запроса: **http://www.example.com/catalog/store.php?category=kitchen&price=5**
PATH_INFO	/browse	Дополнительная информация о пути, присоединяемая в конце URL после знака косой черты. Именно таким способом информация передается сценарию, не прибегая к строке запроса. В следующем URL демонстрируется пример указания элемента **PATH_INFO: http://www.example.com/catalog/ store.php/browse**
SERVER_NAME	www.example.com	Наименование веб-сайта, на котором выполняется интерпретатор PHP. Если на вебсервере размещаются самые разные домены, то данный элемент содержит имя отдельного виртуального домена, к которому осуществляется доступ
DOCUMENT_ROOT	/usr/local/htdocs	Каталог на компьютере веб-сервера, содержащий документы, имеющиеся на веб-сайте. Если корневой каталог документов для вебсайта, доступного по адресу **http://www.example.com**, находится по пути **/usr/local/htdocs**, то запрос по адресу **http://www.example.com/ catalog/store.php** соответствует файлу, находящемуся по пути **/usr/local/htdocs/catalog/store.php**
REMOTE_ADDR	175.56.28.3	IP-адрес пользователя, посылающего запрос на веб-сервер
REMOTE_HOST	pool0560.cvx.dialup.verizon.net	Если веб-сервер настроен на преобразование IP-адресов в имена хостов (т.е. веб-узлов), этот элемент содержит имя хоста пользователя, посылающего запрос на веб-сервер. А поскольку такое преобразование адреса в имя хоста обходится недешево (с точки зрения времени вычисления), то на большинстве веб-серов оно не выполняется
HTTP_REFERER[1]	http://shop.oreilly.com/product/0636920029335.d	Если кто-нибудь щелкнет на ссылке для доступа к странице по текущему URL, элемент **HTTP_REFERER** содержит URL страницы с данной ссылкой. Значение этого элемента может быть подделано, поэтому не пользуйтесь им как единственным критерием для предоставления доступа к закрытым вебстраницам. Тем не менее он может оказать помощь в поиске тех, кто связывается с текущей страницей
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv: 37.0) Gecko/ 20100101 Fire fox/37.0	Веб-браузер, извлекающий страницу. В приведенном примере показано значение, обозначающее сигнатуру браузера Firefox версии 37 для Mac OS X. Как и значение элемента **HTTP_REFERER**, значение этого элемента может быть подделано, хотя оно и полезно для анализа

[1] Правильно пишется **HTTP_REFERRER**. Но в первоначальной спецификации Интернета имя этого элемента было указано с ошибкой, и поэтому оно часто встречается в веб-разработке.

Доступ к параметрам формы

В начале каждого запроса интерпретатор PHP устанавливает ряд автоглобальных массивов, содержащих значения любых параметров, переданных в форме или в URL. Параметры URL и форм, извлекаемых методом GET, размещаются в массиве $_GET, а параметры форм, передаваемых методом POST, — в массиве $_POST.

Например, по следующему URL:

```
http://www.example.com/catalog.php?product_id=21&category=fryingpan
```

в массиве `$_GET` размещаются перечисленные ниже значения:

- значение 21 в элементе массива `$_GET['product_id']`;
- значение `fryingpan` в элементе массива `$_GET['category']`.

Передача формы на обработку в примере 7.2 приводит к тому, что в массиве `$_POST` размещаются одни и те же значения, при условии, что в текстовом поле формы введено значение **21**, а из списка выбран элемент **Frying Pan** (Сковородка).

Пример 7.2. Двухэлементная форма

```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
```

Форма из примера 7.2 внедряется в исходный код программы на PHP из примера 7.3, где на экран выводятся соответствующие значения из массива `$_POST` после отображения формы. В атрибуте `action` дескриптора `<form>` разметки формы из примера 7.3 указано значение **catalog.php**, поэтому данную программу нужно сохранить в файле `catalog.php` на веб-сервере. Если же ее требуется сохранить в файле под другим именем, откорректируйте соответственно значение атрибута `action`.

Пример 7.3. Вывод на экран параметров из формы, переданной на обработку

```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
Here are the submitted values:

product_id: <?php print $_POST['product_id'] ?? '' ?>
<br/>
category: <?php print $_POST['category'] ?? '' ?>
```

Во избежание появления предупреждающего сообщения от интерпретатора PHP о том случае, если переменные не были переданы методом `POST`, в примере 7.3 применяется *нулеобъединяющая* операция **??**. Так, в выражении `$_POST['product_id'] ?? ' '` вычисляется значение элемента `$_POST['product_id']`, если в нем вообще что-нибудь содержится, а иначе — пустая строка (`' '`). Если не принять этой меры предосторожности, появится сообщение вроде `"PHP Notice: Undefined index: product_id"` (Предупреждение PHP: неопределенный индекс: `product_id`), если страница была извлечена методом `GET`, но переменные, передаваемые методом `POST`, не установлены.

Нулеобъединяющая операция была внедрена в версии PHP 7. Если вы пользуетесь более ранней версией PHP, применяйте вместо этой операции функцию `isset()` следующим образом:

```
if (isset($_POST['product_id'])) {
    print $_POST['product_id'];
}
```

Если элемент формы может принимать несколько значений, после его имени следует указать квадратные скобки (**`[]`**). Этим интерпретатору PHP предписывается считать несколько значений элементами массива. Так, в списке, размеченном дескриптором `<select>` в примере 7.4, имеется несколько переданных значений, размещаемых в элементе массива `$_POST['lunch']`.

Пример 7.4. Элементы формы с несколькими значениями

```
<form method="POST" action="eat.php">
<select name="lunch[]" multiple>
<option value="pork">BBQ Pork Bun</option>
<option value="chicken">Chicken Bun</option>
<option value="lotus">Lotus Seed Bun</option>
<option value="bean">Bean Paste Bun</option>
<option value="nest">Bird-Nest Bun</option>
</select>
<input type="submit" name="submit">
</form>
```

Если передать на обработку форму из примера 7.4 с выбранными вариантами **`Chicken Bun`** (Булочка с цыпленком) и **`Bird-Nest Bun`** (Булочка “Птичье гнездо”), то значение элемента массива `$_POST['lunch']` превратится в двухэлементный массив, содержащий значения **`chicken`** и **`nest`**. Доступ к этим значениям осуществляется с помощью обычного синтаксиса многомерных массивов. В примере 7.5 форма из примера 7.4 внедряется в программу, выводящую на экран каждый вариант, выбранный из списка. (Здесь применяется то же самое правило к имени файла и атрибуту `action`. Сохраните исходный код из примера 7.5 в файле `eat.php` или откорректируйте значение атрибута `action` в дескрипторе `<form>`, если потребуется изменить имя файла.)

Пример 7.5. Доступ к нескольким значениям элемента формы

```
<form method="POST" action="eat.php">
<select name="lunch[]" multiple>
<option value="pork">BBQ Pork Bun</option>
<option value="chicken">Chicken Bun</option>
<option value="lotus">Lotus Seed Bun</option>
<option value="bean">Bean Paste Bun</option>
<option value="nest">Bird-Nest Bun</option>
</select>
<input type="submit" name="submit">
</form>
Selected buns:
<br/>
<?php
if (isset($_POST['lunch'])) {
    foreach ($_POST['lunch'] as $choice) {
```

```
        print "You want a $choice bun. <br/>";
    }
}
?>
```

Если из списка выбраны варианты **Chicken Bun** и **Bird-Nest Bun**, то при выполнении кода из примера 7.5 на экран (после самой формы) выводится следующий результат:

```
Selected buns:
You want a chicken bun.
You want a nest bun.
```

Элемент формы `lunch[]` можно рассматривать как преобразуемый в приведенный ниже код PHP при передаче формы на обработку. При этом предполагается, что переданы значения `chicken` и `nest` элемента формы. Как было показано в примере 4.6, такой синтаксис добавляет элемент в конце массива.

```
$_POST['lunch'][] = 'chicken';
$_POST['lunch'][] = 'nest';
```

Обработка форм с помощью функций

Элементарную форму из примера 7.1 можно сделать более гибкой, перенеся код отображения и обработки в отдельные функции. Этот вариант формы из примера 7.1 демонстрируется в примере 7.6 с применением функций. Чтобы изменить форму или то, что с ней происходит, когда она передается на обработку, внесите изменения в тело функции `process_form()` или `show_form()`.

Пример 7.6. Отображение приветствия "Hello" на странице с применением функций

```
// Логика выполнения верных действий на
// основании метода запроса
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    process_form();
} else {
    show_form();
}

// сделать что-нибудь, когда форма передана на обработку
function process_form() {
    print "Hello, ". $_POST['my_name'];
}

// отобразить форму
function show_form() {
    print<<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
      Your name: <input type="text" name="my_name">
    <br />
    <input type="submit" value="Say Hello">
    </form>
    _HTML_;
}
```

Разделение обработки и отображения формы на отдельные функции упрощает также внедрение стадии проверки достоверности данных, которая рассматривается более подробно далее, в разделе “Проверка достоверности данных”. А до тех пор достаточно сказать, что проверка достоверности данных является очень важной функцией любого веб-приложения, принимающего данные, вводимые в форме. После передачи формы на обработку введенные в ней данные должны быть непременно проверены на достоверность, прежде чем приступать к их обработке. В примере 7.7 показано внедрение функции проверки достоверности в код из примера 7.6.

Пример 7.7. Проверка достоверности данных из формы

```
// Логика выполнения верных действий на
// основании метода запроса
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (validate_form()) {
        process_form();
    } else {
      show_form();
    }
} else {
    show_form();
}

// сделать что-нибудь, когда форма передана на обработку
function process_form() {
    print "Hello, ". $_POST['my_name'];
}

// отобразить форму
function show_form() {
    print<<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
      Your name: <input type="text" name="my_name">
    <br/>
    <input type="submit" value="Say Hello">
    </form>
    _HTML_;
}

// проверить данные из формы
function validate_form() {
    // Содержит ли имя, введенное в текстовом поле my_name
    // хотя бы три символа?
    if (strlen($_POST['my_name']) < 3) {
        return false;
    } else {
        return true;
    }
}
```

Функция `validate_form()` из примера 7.7 возвращает логическое значение `false`, если длина строкового значения элемента массива `$_POST['my_name']` меньше трех символов, а иначе — логическое значение `true`. Функция `validate_form()` вызывается в самом начале страницы,

когда форма передана на обработку. Если она возвращает логическое значение `true`, то вызывается функция `process_form()`, а иначе — функция `show_form()`. Это означает, что если передать форму на обработку с введенным именем длиной не меньше трех символов (например, `Bob` или `Bartholomew`), произойдет то же самое, что и в предыдущих примерах: вывод сообщения `"Hello, Bob"` или `"Hello, Bartholomew"`. Если же передать форму с именем короче трех символов (например, `BJ`) или оставить текстовое поле пустым, функция `validate_form()` возвратит логическое значение `false`, а следовательно, функция `process_form()` вообще не будет вызвана. Вместо нее будет вызвана функция `show_form()` для повторного отображения формы.

В коде из примера 7.7 ничего не сообщается, в чем ошибка, если ввести имя, которое не проходит проверку в функции `validate_form()`. В идеальном случае ошибка должна быть пояснена при повторном отображении формы, если пользователь передаст форму с данными, не прошедшими проверку достоверности. И если это уместно, введенное пользователем значение следует повторно отобразить в соответствующем элементе формы. В следующем разделе поясняется, как выводить сообщения о подобных ошибках, а далее, в разделе “Отображение устанавливаемых по умолчанию значений” — как безопасно отображать значения, повторно введенные пользователем.

Проверка достоверности данных

Проверка достоверности данных — одна из самых важных функций вебприложения. Посторонние, неверные или наносящие ущерб данные проявляются там, где этого меньше всего можно ожидать. Пользователи способны проявлять большую небрежность, злонамеренность и невероятную изобретательность (зачастую произвольно), чем можно себе представить, когда разрабатывается веб-приложение. Трудно даже переоценить, насколько важна строгая проверка достоверности любого фрагмента данных, поступающих в приложение из внешнего источника. Некоторые из этих внешних источников вполне очевидны, поскольку большинство входных данных поступает в веб-приложение из заполняемой формы. Но имеется и немало других путей поступления данных в программы: из баз данных, совместно используемых разными приложениями, веб-служб и удаленных серверов и даже URL и их параметров.

Как упоминалось ранее, в коде из примера 7.7 не указывается, что неверно введено в форме, если проверка в функции `validate_form()` не пройдет. Поэтому в примере 7.8 внесены изменения в функции `validate_form()` и `show_form()` для составления и вывода на экран массива возможных сообщений об ошибках.

Пример 7.8. Отображение сообщений об ошибках вместе с формой

```
// Логика выполнения верных действий на
// основании метода запроса
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    if($form_errors = validate_form()) {
        show_form($form_errors);
    } else {
        process_form();
    }
} else {
    show_form();
}
// сделать что-нибудь, когда форма передана на обработку
function process_form() {
    print "Hello, ". $_POST['my_name'];
}
```

```
// отобразить форму
function show_form($errors = ) {
    // Если переданы ошибки, вывести их на экран
    if ($errors) {
        print 'Please correct these errors: <ul><li>';
        print implode ('</li><li>', $errors);
        print ' </li></ul>';
    }

    print <<<_HTML_
    <form method="POST" action="$_SERVER[PHP_SELF]">
      Your name: <input type="text" name="my_name">
    <br/>
    <input type="submit" value="Say Hello">
    </form>
    _HTML_;
}

// проверить данные из формы
function validate_form() {
    // начать с пустого массива сообщений об ошибках
    $errors = array();

    // добавить сообщение об ошибке, если введено слишком
    // короткое имя
    if(strlen($_POST['my_name']) < 3) {
        $errors[ ] = 'Your name must be at least 3 letters long.';
    }

    // возвратить (возможно, пустой) массив сообщений об ошибках
    return $errors;
}
```

В коде из примера 7.8 выгодно используется тот факт, что пустой массив вычисляется как ложный (`false`). В следующей строке кода определяется, следует ли снова вызывать функцию `show_form()` и передавать ей массив сообщений об ошибках или же нужно вызвать функцию `process_form()`:

```
if ($form_errors = validate_form())
```

Массив, возвращаемый функцией `validate_form()`, присваивается переменной `$form_errors`. Истинное значение проверочного выражения в условном операторе `if()` является результатом этого присваивания, поскольку присваивается конкретное значение, как пояснялось выше, в разделе "Общее представление об истинности или ложности" главы 3. Таким образом, проверочное выражение в условном операторе `if()` оказывается истинным (`true`), если переменная `$form_errors` содержит какие-нибудь элементы массива. А если переменная `$form_errors` пуста, то проверочное выражение оказывается ложным (`false`). Функция `validate_form()` возвратит пустой массив, если не обнаружатся никакие ошибки.

Достоверность всех элементов формы целесообразно проверить за один проход, чтобы не отображать повторно форму всякий раз, когда недостоверные данные будут обнаружены в одном элементе формы. Пользователь должен обнаружить все свои ошибки после передачи формы на обработку, чтобы не повторять ее передачу каждый раз, когда появится новое сообщение об ошибке. С этой

целью в функции validate_form() из примера 7.8 по каждой ошибке, обнаруженной в форме, вводится отдельный элемент в массив $errors. А в функции show_form() на экран выводятся соответствующие сообщения об ошибках.

Все методы проверки достоверности в данном примере сосредоточены в функции validate_form(). Если элемент формы не проходит проверку достоверности, то соответствующее сообщение вводится в массив $errors.

Обязательные элементы формы

Чтобы выяснить, было ли что-нибудь введено в обязательном элементе формы, следует проверить длину значения в этом элементе с помощью функции strlen(), как показано в примере 7.9.

Пример 7.9. Проверка достоверности данных в обязательном элементе

```
if (strlen($_POST['email']) == 0) {
    $errors[] = "You must enter an email address.";
}
```

При проверке обязательного элемента формы очень важно воспользоваться функцией strlen() вместо условного оператора if(). Ведь при проверке вроде if(! $_POST['quantity']) значение, вычисляемое как ложное (false), интерпретируется как ошибка. А если применить функцию strlen(), то пользователи смогут ввести в обязательном элементе даже такое значение, как **0**.

Числовые или строковые элементы формы

Чтобы убедиться, что переданное на обработку значение является целым числом или числом с плавающей точкой, следует вызвать функцию filter_input() с подходящим фильтром. С помощью функции filter_input() интерпретатору PHP можно указать, какими именно входными данными следует оперировать, а также наименование переданного значения во входных данных и правило, которому это значение должно соответствовать. В частности, фильтры FILTER_VALIDATE_INT и FILTER_VALIDATE_FLOAT осуществляют проверку на целые числа и числа с плавающей точкой соответственно. В примере 7.10 демонстрируется применение фильтра целых чисел.

Пример 7.10. Фильтрация целочисленных входных данных

```
$ok = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if (is_null($ok) || ($ok === false)) {
    $errors[] = 'Please enter a valid age.';
}
```

В строке кода filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT) из примера 7.10 интерпретатору PHP предписывается проверить данные из переданной на обработку формы (INPUT_POST), особенно из поля формы age, и проверить их относительно фильтра целых чисел (FILTER_VALIDATE_INT). Функции filter_input() указывается, где искать проверяемые данные (аргумент INPUT_POST) и какое поле следует проверять (аргумент age), а не элемент массива вроде $_POST['age'], чтобы она могла надлежащим образом обработать отсутствующие данные, избежав недоразумений на тот случай, если программа PHP изменит значения в массиве $_POST.

Если функция filter_input() обнаружит, что значение в указанном элементе введено верно, она возвратит это значение. Если значение в указанном элементе не введено, она возвратит пустое значение null. А если значение в указанном элементе введено, но недостоверно в соответствии с заданным фильтром, то данная функция возвратит логическое значение false. В проверочном выражении условного оператора if() из примера 7.10 переменная $ok сравнивается с логическим

значением `false` посредством операции `===`, называемой *операцией тождественности*. При сравнении значений в этой операции вычисляется логическое значение true, если оба значения одинаковы и однотипны. Как было показано в примере 3.11, при сравнении двух разнотипных значений (например, строкового и целочисленного или логического и целочисленного) интерпретатор PHP может изменить типы значений, чтобы сравнить их. Так, если в переданной на обработку форме было введено нулевое значение, которое является достоверным целым числом, то значение переменной `$ok` будет равно нулю. И тогда обычное сравнение значения переменной `$ok` с логическим значением `false` даст истинный результат, поскольку нулевое значение вычисляется как ложное (`false`). А операция тождественности даст ложный результат, поскольку типы сравниваемых значений не совпадают.

Это означает, что в массив `$errors` вводится сообщение об ошибке, если значение в поле формы `age` отсутствует (проверка `is_null($ok)`) или не является целочисленным (проверка `$ok === false`). Аналогичным образом осуществляется фильтрация чисел с плавающей точкой, как показано в примере 7.11.

Пример 7.11. Фильтрация числовых входных данных с плавающей точкой

```
$ok = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
if (is_null($ok) || ($ok === false)) {
    $errors[] = 'Please enter a valid price.';
}
```

При проверке достоверности данных (особенно строковых) в элементах заполняемой формы нередко полезно удалить начальные и конечные пробелы с помощью функции `trim()`. Эту функцию можно применять вместе с функцией `strlen()` для проверки обязательных элементов, чтобы исключить ввод одних только пробелов (пример 7.12).

Пример 7.12. Совместное применение функций `trim()` и `strlen()`

```
if (strlen(trim($_POST['name'])) == 0) {
    $errors[] = "Your name is required.";
}
```

Все URL и данные из передаваемой на обработку формы поступают в интерпретатор PHP в виде символьных строк. Если функции `filter_input()` задан числовой фильтр (и передано достоверное числовое значение), она возвратит значение, преобразуемое в целое число или число с плавающей точкой. Аналогично удалению лишних пробелов из символьных строк, пользоваться именно этими преобразованными значениями в программе нередко удобнее, чем значениями непосредственно из массива `$_POST`. Для этого достаточно составить в функции проверки достоверности массив из преобразованных значений для последующей их обработки, как показано в примере 7.13.

Пример 7.13. Составление массива из преобразованных входных данных

```
function validate_form() {
    $errors = array();
    $input = array();

    $input['age'] =
          filter_input (INPUT_POST, 'age', FILTER__VALIDATE_INT);
    if (is_null($input['age']) || ($input['age'] === false)) {
        $errors[] = 'Please enter a valid age.';
    }
```

```
    $input['price'] =
          filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
    if (is_null($input['price']) || ($input['price'] === false)) {
        $errors[] = 'Please enter a valid price.';
    }

    // воспользоваться нулеобъединяющей операцией, если
    // значение в элементе $_POST['name'] не установлено
    $input['name'] = trim($_POST['name') ?? '');
    if (strlen($input['name']) == 0) {
        $errors[] = "Your name is required.";
    }

    return array($errors, $input);
}
```

В функции `validate_form()` из примера 7.13 массив `$input` составляется из значений по мере их проверки. В ней также создается массив `$errors`, если возникают какие-нибудь ошибки. Как только эти массивы будут созданы, их необходимо возвратить, чтобы в остальной части программы можно было воспользоваться не только массивом `$errors`, но и массивом `$input`. С этой целью они связываются вместе в двухэлементный массив, который и возвращается.

Если функция `validate_form()` возвращает как входные данные, так и ошибки, вызывающий ее код должен быть видоизменен, чтобы учесть это обстоятельство. В примере 7.14 демонстрируется видоизмененный вариант начальной части кода из примера 7.8, где обрабатываются оба массива, возвращаемых функцией `validate_form()`.

Пример 7.14. Обработка ошибок и видоизмененных входных данных

```
// Логика выполнения верных действий на
// основании метода запроса
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функции validate_form() возвращает ошибки,
    // передать их функции show_form()
    list($form_errors, $input) = validate_form();
    if ($form_errors) {
        show_form($form_errors);
    } else {
        process_form($input);
    }
} else {
    show_form() ;
}
```

Конструкция `list()` применяется в коде из примера 7.14 для того, чтобы *деструктурировать* значение, возвращаемое из функции `validate_form()`. Как известно, функция `validate_form()` будет всегда возвращать массив с двумя элементами, первый из которых может оказаться пустым массивом сообщений об ошибок, а второй — массивом видоизмененных входных данных, и поэтому в конструкции `list($form_errors, $input)` интерпретатору PHP указывается присвоить первый элемент этого возвращаемого массива переменной `$form_errors`, а второй элемент — переменной `$input`. Наличие этих массивов в отдельных переменных упрощает чтение исходного кода.

После обработки возвращаемых массивов надлежащим образом применяется аналогичная логика. Если массив `$errors` оказывается непустым, вызывается функция `show_form()` с массивом

$errors в качестве аргумента. В противном случае вызывается функция обработки формы. Единственное отличие заключается в том, что теперь функции обработки формы передается массив видоизмененных значений для последующего применения. Это означает, что функция process_form() теперь должна обращаться к элементу массива $input['my_name'], а не к элементу массива $_POST['my_name'] для поиска выводимых на экран значений.

Диапазоны чисел

Чтобы проверить, находится ли число в определенном диапазоне, следует воспользоваться вариантами min_range и max_range выбора фильтра FILTER_VALIDATE_INT. Эти варианты выбора передаются в качестве четвертого аргумента функции filter_input(), как показано в примере 7.15.

Пример 7.15. Проверка целочисленного диапазона

```
$input['age'] = filter_input(INPUT_POST, 'age',
                FILTER_VALIDATE_INT,
                array('options' => array('min_range' => 18,
                                         'max_range' => 65)));
if (is_null($input['age']) || ($input['age'] === false)) {
    $errors[] = 'Please enter a valid age between 18 and 65.';
}
```

Обратите внимание на то, что в качестве четвертого аргумента функции filter_input() передаются не сами варианты выбора фильтра, а одноэлементный массив с ключом options и значением из конкретного массива вариантов выбора и их значений. В фильтре FILTER_VALIDATE_FLOAT не поддерживаются варианты выбора min_range и max_range, и поэтому сравнивать их необходимо самостоятельно, как показано ниже.

```
$input['price'] = filter_input(INPUT_POST, 'price',
                               FILTER_VALIDATE_FLOAT);
if (is_null($input['price']) || ($input['price'] === false) ||
    ($input['price'] < 10.00) || ($input['price'] > 50.00)) {
     $errors[] = 'Please enter a valid price between $10 and $50.';
}
```

Чтобы проверить диапазон дат, переданное значение даты следует сначала преобразовать в объект типа DateTime, а затем проверить достоверность этого значения (подробнее об объектах типа DateTime и функциях checkdate(), применяемых в примере 7.16, см. в главе 15). Объекты типа DateTime инкапсулируют всю информацию, необходимую для представления момента времени, и поэтому для применения диапазона, охватывающего месяц или год, не требуется предпринимать ничего особенного. Так, в примере 7.16 проверяется, относится ли предоставляемая дата к моменту времени менее чем шестимесячной давности.

Пример 7.16. Проверка диапазона дат

```
// создать объект типа DateTime с датой шестимесячной давности
$range_start = new DateTime('6 months ago');
// создать объект типа DateTime с текущей датой
$range_end = new DateTime();

// в элементе массива $_POST['year'] хранится год,
// состоящий из четырех цифр;
```

```
// в элементе массива $_POST['month'] хранится месяц,
// состоящий из двух цифр;
// в элементе массива $_POST['day'] хранится день,
// состоящий из двух цифр
$input['year'] = filter_input(INPUT_POST, 'year',
                 FILTER_VALIDATE_INT,
                 array('options' => array('min_range' => 1900,
                       'max_range' => 2100)));
$input['month'] = filter_input(INPUT_POST, 'month',
                  FILTER_VALIDATE_INT,
                  array('options' => array('min_range' => 1,
                        'max_range' => 12)));
$input['day'] = filter_input(INPUT_POST, 'day',
                FILTER_VALIDATE_INT,
                array('options' => array('min_range' => 1,
                      'max_range' => 31)));

// Для сравнения с логическим значением false операция ===
// не требуется, т.к. нулевое значение является недопустимым
// вариантом выбора года, месяца или дня. В функции checkdate()
// проверяется допустимое количество дней в данном месяце и году
if ($input['year'] && input ['month'] && input['day'] &&
      checkdate($input['month'], $input['day'], $input['year'])) {
                $submitted_date = new DateTime(strtotime(
                                  $input['year'] . '-' .
                                  $input['month'] . '-' .
                                  $input['day']));
    if (($range_start > $submitted_date) ||
        ($range_end < $submitted_date)) {
         $errors[] =
             'Please choose a date less than six months old.';
    }
} else {
    // Это сообщение выводится в том случае, если пользователь
    // пропустит один из параметров или введет в форме дату
    // вроде 31 февраля
    $errors[] = 'Please enter a valid date.';
}
```

Адреса электронной почты

Безусловно, проверка адреса электронной почты является самой распространенной формой проверки достоверности данных из формы. Но для проверки достоверности адреса электронной почты за один раз идеального способа не существует, поскольку "достоверность" означает совершенно разное в зависимости от преследуемой цели. Если действительно требуется убедиться, что пользователь предоставил рабочий адрес электронной почты и что он контролирует этот адрес, необходимо сделать следующее. Во-первых, отправить сообщение, содержащее произвольную символьную строку по переданному в форме адресу электронной почты. В этом сообщении можно предложить пользователю передать произвольную символьную строку в форме на веб-сайт, где она должна быть обработана. И во-вторых, можно включить в сообщение URL, на котором пользователю достаточно щелкнуть и который содержит встроенный код. Если код передан (или произведен щелчок на URL), значит, пользователь получил сообщение и контролирует адрес электронной почты, переданный в

форме на веб-сайт, где она должна быть обработана, или, по крайней мере, он знает о передаче и подтверждает ее.

Если нет желания брать на себя все хлопоты, связанные с проверкой достоверности адреса электронной почты с помощью отдельного сообщения, можно ограничиться простой синтаксической проверкой в коде, проверяющем достоверность данных в форме, чтобы искоренить неверно набранные адреса. В частности, фильтр `FILTER_VALIDATE_EMAIL` проверяет символьные строки по правилам для допустимых адресов электронной почты, как показано в примере 7.17.

Пример 7.17. Проверка синтаксиса адреса электронной почты

```
$input['email'] = filter_input(INPUT_POST, 'email',
                               FILTER_VALIDATE_EMAIL);
if (! $input['email']) {
    $errors[] = 'Please enter a valid email address';
}
```

Упрощенная проверка `if (! $input['email'])` оказывается пригодной в коде из примера 7.17 потому, что любые переданные на обработку символьные строки, которые вычисляются как ложные (например, пустая или нулевая строка), также считаются недопустимыми адресами электронной почты.

Списки, размечаемые дескриптором <select>

Когда в форме употребляется список, размечаемый дескриптором `<select>`, необходимо убедиться, что переданное на обработку значение элемента такого списка разрешено для выбора. И хотя пользователь не может переделать значение вне данного списка, используя типичный, правильно работающий браузер вроде Firefox или Chrome, атакующий злоумышленник способен составить запрос, содержащий произвольное значение, не пользуясь браузером.

Чтобы упростить отображение и проверку достоверности списков, размечаемых дескриптором `<select>`, следует ввести варианты выбора из списка в массив, а затем перебрать массив для отображения такого списка в теле функции `show_form()`. Для проверки переданного на обработку значения можно воспользоваться тем же самым массивом, что и в функции `validate_form()`. В примере 7.18 показано, как отображать подобным способом список, размечаемый дескриптором `<select>`.

Пример 7.18. Отображение списка, размечаемого дескриптором <select>

```
$sweets = array('Sesame Seed Puff'Coconut Milk Gelatin Square',
                'Brown Sugar Cake','Sweet Rice and Meat');

function generate_options($options) {
    $html = '';
    foreach ($options as $option) {
        $html .= "<option>$option</option>\n";
    }
    return $html;
}

// отобразить форму
function show_form() {
    $sweets = generate_options($GLOBALS['sweets']);
    print<<<_HTML_
    <form method="post" action="$_SERVER[PHP_SELF]">
```

```
      Your Order: <select name="order">
      $sweets
    </select>
    <br/>
    <input type="submit" value="Order">
    </form>
    _HTML_;
}
```

Ниже приведена HTML-разметка списка, выводимая на экран в функции `show_form()` из примера 7.18.

```
<form method="post" action="order.php">
Your Order: <select name="order">
<option>Sesame Seed Puff</option>
<option>Coconut Milk Gelatin Square</option>
<option>Brown Sugar Cake</option>
<option>Sweet Rice and Meat</option>

</select>
<br/>
<input type="submit" value="Order">
</form>
```

Массив вариантов выбора из списка, размечаемого дескриптором `<select>`, применяется в теле функции `validate_form()` аналогично следующему:

```
$input['order'] = $_POST['order'];
if (! in_array($input['order'], $GLOBALS['sweets'])) {
    $errors[] = 'Please choose a valid order.';
}
```

Если нужно составить список, размечаемый дескриптором `<select>`, из разных вариантов выбора элементов и их значений, то потребуется более сложный массив. В этом случае ключ к каждому элементу массива является атрибутом `value` для одного элемента списка, а соответствующим значением элемента массива — отображаемый вариант выбора данного элемента списка. В примере 7.19 значениями элементов списка являются следующие величины: **`puff`**, **`square`**, **`cake`** и **`ricemeat`**, а отображаемыми вариантами выбора — **`Sesame Seed Puff`**, **`Coconut Milk Gelatin Square`**, **`Brown Sugar Cake`** и **`Sweet Rice and Meat`**.

Пример 7.19. Список, размечаемый дескриптором <select>, с разными вариантами выбора и значениями элементов

```
sweets = array('puff' => 'Sesame Seed Puff',
               'square' => 'Coconut Milk Gelatin Square',
               'cake' => 'Brown Sugar Cake',
               'ricemeat' => 'Sweet Rice and Meat');

function generate_options_with_value ($options) {
    $html = '';
    foreach ($options as $value => $option) {
        $html .= "<option value=$value¨>$option</option>\n";
     }
```

```
    return $html;
}

// отобразить форму
function show_form() {
    $sweets = generate_options_with_value($GLOBALS['sweets']);
    print<<<_HTML_
    <form method="post" action="$_SERVER[PHP_SELF]">
       Your Order: <select name="order">
       $sweets
    </select>
    <br/>
    <input type="submit" value="Order">
    </form>
    _HTML_;
}
```

Ниже приведена форма, отображаемая в коде из примера 7.19.

```
<form method="post" action="order.php">
Your Order: <select name="order">
<option value="puff">Sesame Seed Puff</option>
<option value="square">Coconut Milk Gelatin Square</option>
<option value="cake">Brown Sugar Cake</option>
<option value="ricemeat">Sweet Rice and Meat</option>

</select>
<br/>
<input type="submit" value="Order">
</form>
```

Из списка, размечаемого дескриптором `<select>` в примере 7.19, на обработку должно быть передано одно из следующих значений: **`puff`**, **`square`**, **`cake`** или **`ricemeat`**. В примере 7.20 показано, как переданное на обработку значение проверяется на достоверность в функции `validate_form()`.

Пример 7.20. Проверка достоверности значения, передаваемого на обработку из списка, размечаемого дескриптором `<select>`

```
$input['order'] = $_POST['order'];
if (! array_key_exists($input['order'] , $GLOBALS['sweets'])) {
    $errors[] = 'Please choose a valid order.';
}
```

HTML и JavaScript

Данные из переданной на обработку формы могут содержать код HTML-разметки или сценария JavaScript, что способно вызвать серьезные осложнения. Рассмотрим в качестве примера простое блог-приложение, дающее пользователям возможность передавать комментарии к публикации в блоге на соответствующей странице, а затем отображающее список этих комментариев ниже публикации. Если пользователи ведут себя прилично и вводят только комментарии, состоящие из простого текста, то страница остается безвредной. Так, если какой-нибудь посетитель опубликует комментарий `"Cool page! I like how you list the different ways to cook fish"` (Замечательная страница! Мне нравится, как вы перечисляете различные способы приготовления рыбы!), то другой посетитель может просмотреть эту страницу и увидеть на ней данный комментарий.

Дело заметно усложняется, когда комментарии состоят не только из простого текста. Если какой-нибудь восторженный пользователь отправит комментарий `"This page <b>rules!!!!</b>"` (Эта страница выглядит круто), он будет дословно воспроизведен блог-приложением, а фраза **rules!!!!** — выделена полужирным при просмотре страницы. Веб-браузер не в состоянии отличить дескрипторы HTML-разметки, поступающие из самого блог-приложения, где комментарии могут размещаться в таблице или списке, от дескрипторов HTML-разметки, которые оказываются встроенными в комментарии, выводимые на экран в данном приложении.

И хотя просмотр текста, выделенного полужирным, вместо простого текста доставляет незначительные неудобства, отображение неотфильтрованных данных, введенных пользователем, оставляет приложение открытым для намного более серьезных осложнений. Вместо дескрипторов `<b></b>` какой-нибудь пользователь может ввести в публикуемые комментарии искаженный или незакрытый дескриптор (например, дескриптор `<a href="` без завершающего знака `"` или `>`), препятствующий браузеру правильно отображать страницу. Хуже того, такой комментарий может содержать код JavaScript, который способен сделать какую-нибудь пакость, например, отправив копии cookie-файлов в чужой почтовый ящик или тайком переадресовав читающего данный комментарий на другую веб-страницу, если этот код будет выполнен веб-браузером при просмотре страницы.

Приложение действует в качестве координатора, позволяя злонамеренному пользователю выгрузить какой-нибудь код HTML-разметки или сценария JavaScript, чтобы выполнить его далее в не подозревающем ни о чем браузере. Такого рода нарушение называется *атакой типа межсайтового выполнения сценариев*, поскольку неудачно написанное блог-приложение позволяет коду из одного источника (т.е. злонамеренного пользователя) замаскироваться под данные, поступающие из другого места (т.е. из приложения, на котором размещаются комментарии).

Чтобы предотвратить атаки типа межсайтового выполнения сценариев в программах на PHP, ни в коем случае не следует отображать входные данные из внешнего источника в необработанном виде. Необходимо удалить подозрительные части (например, дескрипторы HTML-разметки) или закодировать специальные символы таким образом, чтобы браузеры не реагировали на встроенный код HTML-разметки или сценария JavaScript. В языке PHP предоставляются две функции, упрощающие решение подобных задач. В частности, функция `strip_tags()` удаляет дескрипторы HTML-разметки из символьной строки, а функция `htmlentities()` кодирует специальные символы HTML-разметки. Применение функции `strip_tags()` демонстрируется в примере 7.21.

Пример 7.21. Очистка символьной строки от дескрипторов HTML-разметки

```
// удалить дескрипторы HTML-разметки из комментариев
$comments = strip_tags($_POST['comments']);
// а теперь вывести содержимое переменной $comments на экран
print $comments;
```

Если элемент массива `$_POST['comments']` содержит следующую информацию:

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea.
```

при выполнении кода из примера 7.21 на экран выводится следующий результат:

```
I love sweet rice & tea.
```

Все дескрипторы HTML-разметки и их атрибуты удаляются, не затрагивая заключенный в них простой текст. Функция `strip_tags()` очень удобна, но она неверно обрабатывает непарные знаки **<** и **>**. Например, исходную строку `"I <3 Monkeys"` она преобразует в строку `"I "`. Эта функция начинает очистку, как только обнаружит знак **<**, не останавливаясь на этом, даже если отсутствует соответствующий знак **>**.

Зачастую лучшие результаты дает кодирование вместо очистки дескрипторов. В примере 7.22 демонстрируется кодирование с помощью функции `htmlentities()`.

Пример 7.22. Кодирование HTML-представлений символов в строке

```
$comments = htmlentities($_POST['comments']);
// Теперь содержимое переменной $comments можно вывести на экран
print $comments;
```

Если элемент массива `$_POST['comments']` содержит следующее:

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea
```

то при выполнении кода из примера 7.22 на экран выводится такой результат:

```
I &lt;b&gt;love&lt;/b&gt; sweet &lt;div class=&quot;fancy
&quot;&gt;rice&lt;/div&gt; &amp; tea.
```

А если элемент массива `$_POST['comments']` содержит следующее:

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea
```

то при выполнении кода из примера 7.22 на экран выводится такой результат:

```
I &lt;b&gt;love&lt;/b&gt; sweet &lt;div class=&quot;fancy
&quot;&gt;rice&lt;/div&gt; samp; tea.
```

Таким образом, символы, имеющие особое назначение в HTML-разметке (<, >, `&` и `"`), изменены на следующие эквиваленты их представлений:

- знак < на HTML-представление `<`
- знак > на HTML-представление `>`
- знак `&` на HTML-представление `&`
- знак `"` на HTML-представление `&quot;`

Когда браузер обнаруживает HTML-представление `<`, он выводит на экран знак < вместо того, чтобы считать его дескриптором HTML-разметки. Это такой же принцип, хотя и с другим синтаксисом, как и экранирование знака `"` или `$` в символьной строке, заключаемой в двойные кавычки (см. раздел “Текст” главы 2). На рис. 7.4 показано, как выглядит результат выполнения кода из примера 7.22 в окне веб-браузера.

В большинстве приложений следует пользоваться функцией `htmlentities()` для “санитарной” очистки входных данных из внешнего источника. Эта функция вообще не отбрасывает содержимое и в то же время защищает от атак типа межсайтового выполнения сценариев. Например, на форуме, куда пользователи отправляют свои сообщения, обсуждая вопросы вроде “Каково назначение дескриптора `<div>`?” на тему HTML-разметки или “Если `x < y`, то `2x > z`?” по алгебре, вряд ли стоило пропускать подобные отправления через функцию `strip_tags()`. Ведь в таком случае эти вопросы выглядели бы на экране следующим образом: “Каково назначение дескриптора?” и “Если `xz`?” соответственно.

Рис. 7.4: Отображение текста, закодированного HTML-представлениями, в окне браузера

Не только синтаксис

Большинство рассмотренных до сих пор стратегий проверки достоверности данных в этой главе подразумевают проверку синтаксиса значения, переданного на обработку. Они позволяют убедиться, соответствуют ли переданные на обработку данные определенному формату. Но иногда требуется убедиться, что переданное на обработку значение имеет не только верный синтаксис, но и приемлемое смысловое назначение. Именно это и делается при проверке достоверности списка, размечаемого дескриптором <select>. Вместо того чтобы проверить, является ли символьной строкой переданное на обработку значение, в данном случае осуществляется проверка на совпадение с конкретными значениями из массива. Еще одним примером проверки не только синтаксиса служит стратегия выдачи сообщений с запросом на подтверждение, предполагающая проверку достоверности адресов электронной почты. Если убедиться лишь в том, что адрес электронной почты передан в верной форме, озорной пользователь может предоставить, например, адрес president@whitehouse.gov, который, очевидно, ему не принадлежит. Сообщение с запросом на подтверждение позволяет убедиться в правильности смыслового назначения адреса, т.е. в том, что данный адрес электронной почты действительно принадлежит тому, кто его предоставил.

Отображение значений, устанавливаемых по умолчанию

Иногда требуется отобразить форму со значением, которое уже находится в текстовом поле или предварительно установлено во флажках, кнопках-переключателях или элементах выбора из списка, размечаемого дескриптором `<select>`. Кроме того, при повторном отображении формы из-за ошибок ее заполнения полезно сохранить любую информацию, которую пользователь уже ввел. В примере 7.23 приведен код, предназначенный для этой цели.

Пример 7.23. Построение массива значений, устанавливаемых по умолчанию

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $defaults = $_POST;
} else {
    $defaults = array('delivery' => 'yes',
                      'size' => 'medium',
                      'main_dish' => array('taro','tripe'),
                      'sweet' => 'cake');
}
```

Если элемент массива `$_SERVER['REQUEST_METHOD']` содержит метод `POST`, это означает, что форма передана. В таком случае устанавливаемые по умолчанию значения должны происходить из тех данные, которые передал пользователь. В противном случае можно установить свои значения по умолчанию. Для большинства параметров формы устанавливаемым по умолчанию значением является символьная строка или число. А в тех элементах формы, у которых имеется не одно значение, устанавливаемое по умолчанию значение может быть массивом. Характерным тому примером служит список `main_dish`, размечаемый дескриптором `<select>` и состоящий из многозначных элементов.

После установки значений по умолчанию следует предоставить подходящее значение из массива `$defaults` при выводе на экран дескриптора HTML-разметки соответствующего элемента формы. Напомним, что устанавливаемые по умолчанию значения следует при необходимости закодировать с помощью функции `htmlentities()`, чтобы предотвратить атаки типа межсайтового выполнения сценариев. А структура дескрипторов HTML-разметки требует по-разному интерпретировать текстовые поля, списки, размечаемые дескриптором `<select>`, текстовые области, флажки и кнопки-переключатели.

Так, для разметки текстовых полей следует установить значение соответствующего элемента из массива `$defaults` в атрибуте `value` дескриптора `<input>`. В примере 7.24 показано, как это делается.

Пример 7.24. Установка значения по умолчанию в текстовом поле

```
print '<input type="text" name="my_name" value="' .
        htmlentities($defaults['my_name']). '">';
```

А для разметки многострочных текстовых полей значение, закодированное HTML-представлением, следует установить между дескрипторами `<textarea>` и `</textarea>`, как показано в примере 7.25.

Пример 7.25. Установка значения по умолчанию в многострочной текстовой области

```
print '<textarea name="comments">';
print htmlentities($defaults['comments']);
print '</textarea>';
```

Для разметки списков дескриптором `<select>` следует ввести проверку в цикл, выводящий на экран дескрипторы `<option>`, чтобы вывести атрибут `selected`, когда это уместно. В примере 7.26 приведен код, выполняющий эти действия для списка, размечаемого дескриптором `<select>`.

Пример 7.26. Установка значения по умолчанию в списке, размечаемого дескриптором `<select>`

```
$sweets = array('puff' => 'Sesame Seed Puff',
                'square' => 'Coconut Milk Gelatin Square',
```

```
                'cake' => 'Brown Sugar Cake',
                'ricemeat' => 'Sweet Rice and Meat');

print '<select name="sweet">';
// Знак > обозначает значение элемента выбора,
// а переменная $label — отображаемый элемент списка
foreach ($sweets as $option => $label) {
    print '<option value="' .$option . '"';
    if ($option == $defaults['sweet']) {
        print ' selected';
    }
    print "> $label</option>\n";
}
print '</select>';
```

Чтобы установить значения по умолчанию в списке, размечаемом дескриптором `<select>` и состоящем из с многозначных элементов, необходимо преобразовать массив устанавливаемых по умолчанию значений в ассоциативный массив, в котором каждый ключ обозначает выбираемый вариант выбора. Затем следует вывести атрибут `selected` для элементов списка, находящихся в данном ассоциативном массиве. В примере 7.27 показано, как это делается.

Пример 7.27. Установка значений по умолчанию в списке, размечаемом дескриптором `<select>` и состоящем из многозначных элементов

```
$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' => 'Abalone with Marrow and Duck Feet');

print '<select name="main_dish[]" multiple>';

$selected_options = array();
foreach ($defaults['main_dish') as $option) {
    $selected_options[$option] = true;
}

// вывести дескрипторы <option>
foreach ($main_dishes as $option => $label) {
    print '<option value="' . htmlentities($option) . '"';
    if (array_key_exists($option, $selected_options)) {
        print ' selected';
    }
    print '>' . htmlentities($label) . '</option>';
    print "\n";
}
print '</select>';
```

Для разметки флажков и кнопок-переключателей дескриптор `<input>` дополняется атрибутом `checked`. Синтаксис разметки флажков и кнопок-переключателей отличается лишь значением атрибута `type`. Так, в примере 7.28 на экран выводится устанавливаемый по умолчанию флажок

delivery и три кнопки-переключателя size с разными значениями, устанавливаемыми по умолчанию.

Пример 7.28. Установка значений по умолчанию в кнопках-переключателях и флажках

```
print '<input type="checkbox" name="delivery" value="yes"';
if ($defaults['delivery'] == 'yes') { print ' checked'; }
print '> Delivery?';

$checkbox_options = array('small' => 'Small',
                          'medium' => 'Medium',
                          'large' => 'Large');

foreach ($checkbox_options as $value => $label) {
    print '<input type="radio" name="size" value="'.$value. '"';
    if ($defaults['size'] == $value) { print ' checked'; }
       print "> $label ";
}
```

Собирая все вместе

Превращение скромной веб-формы в полноценное веб-приложение с проверкой достоверности данных, выводом на экран устанавливаемых по умолчанию значений и обработкой переданных результатов может оказаться непростой задачей. Чтобы облегчить решение такой задачи, в этом разделе представлен пример законченной программы, выполняющей следующие действия.

- Отображение формы с устанавливаемыми по умолчанию значениями.
- Проверка достоверности данных, переданных из формы на обработку.
- Повторное отображение формы с сообщениями об ошибках и сохраненными данными, введенными пользователем, если переданные на обработку данные недостоверны.
- Обработка переданных на обработку данных, если они достоверны.

Данный пример законченной программы основан на классе, содержащем ряд вспомогательных методов, чтобы упростить отображение и обработку элементов заполняемой формы. Исходный код этого класса приведен в примере 7.29.

Пример 7.29. Вспомогательный класс для отображения и обработки элементов заполняемой формы

```
class FormHelper {
    protected $values = array();

    public function __construct($values = array()) {
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            $this->values = $_POST;
        } else {
          $this->values = $values;
        }
    }
```

```
public function input($type, $attributes = array(),
                      $isMultiple = false) {
    $attributes['type'] = $type;
    if (($type == 'radio') || ($type == 'checkbox')) {
        if ($this->isOptionSelected($attributes['name']
                  ?? null, $attributes['value'] ?? null)) {
            $attributes['checked'] = true;
        }
    }
    return $this->tag('input', $attributes, $isMultiple);
}

public function select($options, $attributes = array()) {
    $multiple = $attributes['multiple'] ?? false;
    return
       $this->start('select', $attributes, $multiple) .
       $this->options($attributes['name'] ?? null, $options) .
       $this->end('select');
}

public function textarea($attributes = array()) {
    $name = $attributes['name'] ?? null;
    $value = $this->values[$name] ?? '';
    return $this->start('textarea', $attributes) .
           htmlentities($value) .
           $this->end('textarea');
}

public function tag($tag, $attributes = array(),
                    $isMultiple = false) {
    return
    "<$tag {$this->attributes($attributes, $isMultiple)} />";
}

public function start($tag, $attributes = array(),
                      $isMultiple = false) {
    // Дескрипторы <select> и <textarea> не получают
    // атрибуты value
    $valueAttribute =
          (! (($tag == 'select')||($tag == 'textarea')));
    $attrs = $this->attributes($attributes, $isMultiple,
                               $valueAttribute);
    return "<$tag $attrs>";
}

public function end($tag) {
    return "</$tag>";
}

protected function attributes($attributes, $isMultiple,
                              $valueAttribute = true) {
    $tmp = array();
```

```
        // Если данный дескриптор может содержать атрибут value,
        // а его имени соответствует элемент в массиве значений,
        // то установить этот атрибут
        if ($valueAttribute && isset($attributes['name'])
                        && array_key_exists($attributes['name'],
                        $this->values)) {
            $attributes['value'] =
                        $this->values[$attributes['name']];
        }
        foreach ($attributes as $k => $v) {
            // Истинное логическое значение означает
            // логический атрибут
            if (is_bool($v)) {
                if ($v) { $tmp[] = $this->encode($k); }
            }
            // иначе k = v
            else {
               $value = $this->encode($v);
               // Если это многозначный элемент, присоединить
               // квадратные скобки ([]) к его имени
               if ($isMultiple && ($k == 'name')) {
                   $value .= '[]';
               }
               $tmp[] = "$k=\"$value\"";
            }
        }
        return implode(' ', $tmp);
    }

    protected function options($name, $options) {
        $tmp = array();
        foreach ($options as $k => $v) {
            $s = "<option value=\"{$this->encode($k)}\"";
            if ($this->isOptionSelected($name, $k)) {
                $s .= ' selected';
            }
            $s .= ">{$this->encode($v)}</option>";
            $tmp[] = $s;
        }
        return implode('', $tmp);
    }

    protected function isOptionSelected($name, $value) {
         // Если для аргумента $name в массиве отсутствует
         // элемент, значит, этот элемент нельзя выбрать
         if (! isset($this->values[$name])) {
             return false;
         }
         // Если же для аргумента $name в массиве имеется
         // элемент, который сам является массивом, проверить,
         // находится значение аргумента $value в массиве
         else if (is_array($this->values[$name])) {
```

```
            return in_array($value, $this->values[$name]);
        }
        // А иначе сравнить значение аргумента $value с
        // элементом массива значений по значению аргумента $name
        else {
            return $value == $this->values[$name];
        }
    }

    public function encode($s) {
        return htmlentities($s);
    }
}
```

В методах из класса, приведенного в примере 7.29, внедрена логика, обсуждавшаяся ранее в разделе “Отображение значений, устанавливаемых по умолчанию” для отдельных видов элементов заполняемой формы. В исходном коде формы, приведенном в примере 7.30, применяются разные элементы, и поэтому код их отображения проще вынести в отдельные, повторно вызываемые функции, чтобы не дублировать его всякий раз, когда потребуется вывести отдельный элемент формы на экран.

Конструктору класса `FormHelper` в качестве аргумента должен быть передан ассоциативный массив значений, устанавливаемых по умолчанию. Если для запроса не выбран метод `POST`, то данный массив используется для выявления подходящих значений по умолчанию. В противном случае основанием для устанавливаемых по умолчанию значений служат переданные на обработку данные.

В методе `input()` из класса `FormHelper` формируется соответствующая HTML-разметка для любого элемента ввода данных в форме. В качестве первого аргумента этому методу передается тип элемента заполняемой формы (например, `submit`, `radio` или `text`), а в качестве второго необязательного аргумента — ассоциативный массив атрибутов элемента формы (например, `['name' => 'meal']`). И, наконец, в качестве третьего необязательного аргумента данному методу передается логическое значение `true`, если формируется HTML-разметка многозначного элемента формы, например, флажка.

В методе `select()` формируется разметка списка дескриптором `<select>`. В качестве первого аргумента этому методу передается массив элементов, выбираемых из списка, а в качестве второго необязательного аргумента — ассоциативный массив атрибутов дескриптора `<select>`. Для разметки списка с многозначными элементами дескриптором `<select>` следует включить пару “ключ-значение” `'multiple' => true` в массив атрибутов, передаваемый данному методу в качестве второго аргумента.

Метод `textarea()` формирует HTML-разметку текстовой области дескриптором `<textarea>`. В качестве единственного аргумента он принимает ассоциативный массив атрибутов данного дескриптора.

Три упомянутых выше метода должны взять на себя всю работу по отображению формы. Но если для ее разметки потребуются другие дескрипторы или специальная интерпретация имеющихся дескрипторов, тогда можно воспользоваться методами `tag()`, `start()` и `end()`.

В частности, метод `tag()` формирует HTML-разметку всего самозакрывающегося дескриптора вроде `<input/>`. В качестве аргументов ему передаются имя дескриптора, необязательный массив атрибутов и логическое значение `true`, если дескриптор может принимать несколько значений. Для формирования надлежащей HTML-разметки в методе `input()` вызывается метод `tag()`.

Методы `start()` и `end()` служат для формирования HTML-разметки элементов формы с отдельными начальным и конечным дескрипторами. В частности, метод `start()` формирует начальный дескриптор разметки элемента формы, принимая в качестве аргументов имя дескриптора, атрибуты и признак многозначности элемента формы. А метод `end()` принимает в качестве аргумента только имя дескриптора, возвращая закрывающий дескриптор HTML-разметки. Так, если

для HTML-разметки элемента формы применяется дескриптор <fieldset>, можно сделать вызов start('fieldset',['name'=>'adjustments']), сформировать HTML-разметку набора полей, а затем сделать вызов end('fieldset').

Остальная часть методов из класса FormHelper выполняет вспомогательную роль при формировании HTML-разметки и не предназначена для вызова за пределами данного класса. В частности, метод attributes() форматирует ряд атрибутов, чтобы ввести их надлежащим образом в дескриптор HTML-разметки. Используя значения, устанавливаемые по умолчанию в объекте данного класса, этот метод вводит соответствующий атрибут value по мере надобности. Кроме того, он присоединяет квадратные скобки ([]) к имени элемента, если этот элемент может принимать несколько значений, а также обеспечивает надлежащее кодирование значений атрибутов HTML-представлениями.

Метод options() выполняет форматирование дескрипторов <option> для разметки списка дескриптором <select>. С помощью метода isOptionSelected() в нем выявляются те элементы списка, которые должны быть помечены как выбранные (selected), а также выполняется надлежащее кодирование значений атрибутов HTML-представлениями.

Метод encode() служит оболочкой для встроенного в PHP метода htmlentities(). Это открытый (public) метод, поэтому его можно вызывать в исходном коде программы с целью обеспечить согласованность кодирования HTML-представлениями.

Исходный код из примера 7.30 основан на классе FormHelper для отображения краткой формы заказа на приготовление трапезы. Если форма передана на обработку правильно, результаты ее обработки отображаются в окне браузера и отправляются (предположительно шеф-повару, чтобы он приступил к приготовлению заказанной еды) по адресу электронной почты, определяемому в функции process_form(). Код из данного примера входит и выходит из режима PHP, и поэтому он начинается для большей ясности открывающим дескриптором <?php и оканчивается закрывающим дескриптором ?>.

Пример 7.30. Полноценная форма с отображением устанавливаемых по умолчанию значений, проверкой достоверности и обработкой переданных данных

```
<?php
// Здесь предполагается, что исходный файл FormHelper.php
// находится в том же каталоге, где и данный файл

require 'FormHelper.php';

// Установить массивы с вариантами выбора из списка,
// размечаемого дескриптором <select>. Следующие массивы
// требуются в функциях display_form(), validate_form()
// и process_form(), и поэтому они объявляются в глобальной
// области действия

$sweets = array('puff' => 'Sesame Seed Puff',
                'square' => 'Coconut Milk Gelatin Square',
                'cake' => 'Brown Sugar Cake',
                'ricemeat' => 'Sweet Rice and Meat');

$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' => 'Abalone with Marrow and Duck Feet');
```

```
// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUESTJdETHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    $defaults = array('delivery' => 'yes',
                      'size' => 'medium');
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper($defaults);

    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'complete-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();

    // обязательное имя
    $input['name'] = trim($_POST['name'] ?? '');
    if (! strlen($input['name'])) {
        $errors[] = 'Please enter your name.';
    }

    // обязательный размер блюда
    $input['size'] = $_POST['size'] ?? '';
    if (! in_array($input['size'], ['small','medium','large'])) {
        $errors[] = 'Please select a size.';
    }

    // обязательное сладкое блюдо
    $input['sweet'] = $_POST['sweet'] ?? '';
    if (! array_key_exists($input['sweet'], $GLOBALS['sweets'])) {
        $errors[] = 'Please select a valid sweet item.';
    }
```

```
    // два обязательных блюда
    $input['main_dish'] = $_POST['main_dish'] ?? array();
    if(count($input['main_dish']) != 2) {
        $errors[] = 'Please select exactly two main dishes.';
    } else {
        // Если выбрано два основных блюда, убедиться в их
        // достоверности
        if (! (array_key_exists($input['main_dish'][0],
                    $GLOBALS['main_dishes']) &&
                    array_key_exists($input['main_dish'][1],
                    $GLOBALS['main_dishes']))) {
            $errors[] =
                'Please select exactly two valid main dishes.';
        }
    }
    // Если выбрана доставка, то в комментариях должны быть
    // указаны ее подробности
    $input['delivery'] = $_POST['delivery'] ?? 'no';
    $input['comments'] = trim($_POST['comments'] ?? '');
    if (($input['delivery'] == 'yes') &&
                (! strlen($input['comments']))) {
        $errors[] = 'Please enter your address for delivery.';
    }
    return array($errors, $input);
}

function process_form($input) {
    // найти полные наименования основных и сладких блюд
    // в массивах $GLOBALS['sweets'] и $GLOBALS['main_dishes']
    $sweet = $GLOBALS['sweets'][ $input['sweet'] ];
    $main_dish_1 = $GLOBALS['main_dishes'][ $input['main_dish'][0] ];
    $main_dish_2 = $GLOBALS['main_dishes'][ $input['main_dish'][1] ];
    if (isset($input['delivery']) && ($input['delivery'] == 'yes')) {
        $delivery = 'do';
    } else {
        $delivery = 'do not';
    }
    // составить текст сообщения с заказом трапезы
    $message=<<<_ORDER_
      Thank you for your order, {$input['name']}.
      You requested the {$input['size']} size of $sweet,
      $main_dish_1, and $main_dish_2.
      You $delivery want delivery.
      _ORDER_;
    if (strlen(trim($input['comments']))) {
        $message .= 'Your comments: '.$input['comments'];
    }

    // отправить сообщение шеф-повару
    mail('chef@restaurant.example.com', 'New Order', $message);
    // вывести сообщение на экран, но закодировать его любыми
    // HTML-представлениями и преобразовать знаки перевода строки
```

```
    // в дескрипторы <br/>
    print nl2br(htmlentities($message, ENT_HTML5));
}
?>
```

Исходный код из примера 7.30 условно делится на четыре части: код из глобальной области действия, функции `show_form()`, `validate_form()` и `process_form()`. В глобальной области действия код решает три задачи. Сначала он загружает класс `FormHelper` из отдельного файла. Затем устанавливает два массива, в которых описываются варианты выбора из двух списков, размечаемых в форме дескриптором `<select>`. Эти массивы применяются в каждой из функций `show_form()`, `validate_form()` и `process_form()` и поэтому должны быть определены в глобальной области действия. И, наконец, код из глобальной области действия выполняет условный оператор `if()`, в котором принимается решение, что делать дальше: отображать, проверять на достоверность или обрабатывать форму.

Отображение формы осуществляет функция `show_form()`. Сначала она создает массив `$defaults` из значений по умолчанию. Затем этот массив передается конструктору класса `FormHelper`, а следовательно, во вновь создаваемом объекте `$form` данного класса устанавливаются нужные значения по умолчанию. Далее функция `show_form()` передает управление другому исходному файлу (`complete-form.php`), содержащему конкретный код HTML-разметки формы, а также код PHP ее отображения. Вынесение HTML-разметки в отдельный файл для такой крупной программы, как рассматриваемая в этом разделе, позволяет упростить ее усвоение, а также вносить изменения в оба файла независимо друг от друга. Содержимое исходного файла `complete-form.php` приведено в примере 7.31.

Пример 7.31. Код HTML и PHP для разметки и отображения формы

```
<form method="POST"
  action="<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
  <tr>
    <td>You need to correct the following errors:</td>
    <td><ul>
      <?php foreach ($errors as $error) { ?>
        <li><?= $form->encode($error) ?></li>
      <?php } ?>
    </ul></td>
<?php } ?>

<tr><td>Your Name:</td>
  <td><?= $form->input('text', ['name' => 'name']) ?>
</td></tr>

<tr><td>Size: </td>
  <td><?= $form->input('radio',['name' => 'size',
                        'value' => 'small']) ?>
  Small <br/>
    <?= $form->input('radio',['name' => 'size',
                      'value' => 'medium']) ?>
     Medium <br/>
    <?= $form->input('radio',['name' => 'size',
                      'value' => 'large']) ?>
    Large <br/>
```

```
  </td></tr>
<tr><td>Pick one sweet item:</td>
  <td><?= $form->select($GLOBALS['sweets'],
                        ['name' => 'sweet']) ?></td>
</tr>

<tr><td>Pick two main dishes :</td>
  <td><?= $form->select($GLOBALS['main_dishes'],
                        ['name' => 'main_dish',
                        'multiple' => true]) ?></td>
</tr>

<tr><td>Do you want your order delivered?</td>
  <td><?= $form->input('checkbox',['name' => 'delivery',
                       'value' => 'yes'])
  ?> Yes </td></tr>

<tr><td>Enter any special instructions.<br/>
  If you want your order delivered, put your address here:</td>
  <td><?= $form->textarea(['name' => 'comments']) ?></td></tr>

<tr><td colspan="2" align="center">
  <?=$form->input('submit', ['value' => 'Order']) ?>
  </td></tr>

</table>
</form>
```

Код из исходного файла `complete-form.php` выполняется так, как будто он является частью функции `show_form()`. Это означает, что такие локальные переменные, как, например, `$errors` и `$form`, доступны из данной функции в исходном файле `complete-form.php`. И как все включаемые файлы, исходный файл `complete-form.php` запускается вне любых дескрипторов PHP и поэтому позволяет вывести на экран какую-нибудь HTML-разметку, а затем перейти в режим PHP, если потребуется вызвать методы или воспользоваться логикой PHP. В данном коде в качестве сокращенного способа отображения результатов вызовов различных методов применяется *короткий эхо-дескриптор* (`<?=`). Начало блока кода PHP с короткого эхо-дескриптора `<?=` равнозначно его началу с дескриптора `<php echo`. Это удобно для HTML-разметки формы, поскольку различные методы из класса `FormHelper` возвращают HTML-разметку, которая должна быть отображена.

В главном файле функция `validate_form()` создает массив сообщений об ошибках, если переданные данные из формы не удовлетворяют подходящим критериям. Следует, однако, иметь в виду, что при проверках параметров `size`, `sweet` и `main_dish` выясняется не только, что именно было передано в качестве их значений, но и достоверность значений отдельных параметров. Так, для параметра `size` должно быть передано значение **`small`**, **`medium`** или **`large`**, а для параметров `sweet` и `main_dish` — ключи из глобальных массивов `$sweets` и `$main_dishes`. И хотя рассматриваемая здесь форма содержит устанавливаемые по умолчанию значения, входные данные все же целесообразно проверять на достоверность. Пытаясь взломать веб-сайт с данной формой, кто-нибудь может в обход обычного браузера составить запрос с произвольным значением, не относящимся к числу допустимых для выбора в списке, размечаемом дескриптором `<select>`, или в группе кнопок-переключателей.

И, наконец, функция `process_form()` предпринимает действия для обработки формы, если она передана с достоверными данными. Сначала в этой функции составляется символьная строка `$message`, содержащая описание заказа, переданного на обработку. Затем строка `$message` от-

правляется по адресу chef@restaurant.example.com электронной почты и выводится на экран. Отправку сообщения по электронной почте осуществляет встроенная функция mail(). Прежде чем вывести строку $message, функция process_form() передает ее двум другим функциям. Первой из них является упоминавшаяся ранее функция htmlentities(), кодирующая любые специальные символы HTML-представлениями. А вторая функция, nl2br(), преобразует знаки перевода строки в дескрипторы
 непосредственно в строке $message. Благодаря этому сообщение верно отображается в окне веб-браузера.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о взаимодействии веб-браузера с веб-сервером, выполняющим отображение формы, обработку переданных параметров формы и вывод полученных результатов.
- Установление связи между атрибутом action дескриптора <form> и URL, по которому передаются параметры формы.
- Применение значений из автоглобального массива $_SERVER.
- Доступ к параметрам переданной на обработку формы в автоглобальных массивах $_GET и $_POST.
- Доступ к многозначным параметрам переданной на обработку формы.
- Применение функций show_form(), validate_form() и process_form() для модульной организации обработки формы.
- Отображение сообщений об ошибках вместе с формой.
- Проверка достоверности данных в элементах формы: обязательных элементов, целых чисел, чисел с плавающей точкой, символьных строк, диапазонов дат, адресов электронной почты и списков, размечаемых дескриптором <select>.
- Очистка или удаление элементов кода HTML и JavaScript из переданных на обработку данных перед их отображением.
- Отображение значений, устанавливаемых по умолчанию в элементах формы.
- Применение вспомогательных функций для отображения элементов формы.

Упражнения

1. Что будет содержать массив $_POST после передачи на обработку приведенной ниже формы, где выбран третий элемент из списка Braised Noodles (Тушеное мясо с лапшой), первый и последний элементы из списка Sweet (Сладкое), а в текстовом поле введено значение **4**?

```
<form method="POST" action="order,php">
Braised Noodles with: <select name="noodle">
<option>crab meat</option>
<option>mushroom</option>
<option>barbecued pork</option>
<option>shredded ginger and green onion</option>
</select>
```

```
<br/>
Sweet: <select name="sweet[]" multiple>
<option value="puff"> Sesame Seed Puff
<option value="square"> Coconut Milk Gelatin Square
<option value="cake"> Brown Sugar Cake
<option value="ricemeat"> Sweet Rice and Meat
</select>
<br/>
Sweet Quantity: <input type="text" name="sweet_q">
<br/>
<input type="submit" name="submit" value="Order">
</form>
```

2. Напишите функцию `process_form()`, выводящую на экран все параметры переданной на обработку формы и их значения. Можете допустить, что параметры формы имеют только скалярные значения.

3. Напишите программу, выполняющую основные арифметические операции. С этой целью отобразите форму с текстовым полем для ввода двух операндов и список, размечаемых дескриптором `<select>`, для выбора операции сложения, вычитания, умножения или деления. Организуйте проверку достоверности вводимых данных, чтобы они были числовыми и пригодными для выполнения выбранной арифметической операции. Функция обработки вводимых данных должна отображать операнды, операцию и результат ее выполнения. Так, если введены операнды **4** и **2** и выбрана операция умножения, то функция обработки вводимых данных должна отобразить следующее: **4*2 = 8**.

4. Напишите программу, отображающую, проверяющую достоверность и обрабатывающую форму для ввода сведений о доставленной посылке. Эта форма должна содержать поля ввода адресов отправителя и получателя, а также размеров и веса посылки. При проверке достоверности данных из переданной на обработку формы должно быть установлено, что вес посылки не превышает 150 фунтов (около 68 кг), а любой из ее размеров — 36 дюймов (порядка 91 см). Можете также допустить, что в форме введены адреса США, но в таком случае проверьте правильность ввода обозначения штата и почтового индекса. Функция обработки формы в вашей программе должна выводить на экран сведения о посылке в виде организованного, отформатированного отчета.

5. Видоизмените (дополнительно) функцию `process_form()`, перечисляющую все параметры переданной на обработку формы и их значения, а также правильно обрабатывающую те параметры переданной на обработку формы, которые содержат массивы в качестве своих значений. Напомним, что элементы массива сами могут содержать массивы.

ГЛАВА 8

Хранение информации в базах данных

HTML-разметка и стилевое оформление CSS, придающие веб-сайту привлекательный вид, находятся в отдельных файлах на веб-сервере, как, впрочем, и код PHP, выполняющий обработку форм и прочие динамические чародейства. Но для нормальной работы веб-приложения требуется еще одна важная информационная составляющая: данные. И хотя данные вроде списка пользователей и сведений о продукции можно хранить в отдельных файлах, для этой цели проще и удобнее пользоваться базами данных, которым посвящена эта глава.

Большинство видов информации подпадает под обширную категорию *данных*, включая следующее.

- Сведения о пользователях: их имена и адреса электронной почты.
- Сведения о деятельности пользователей: их профильная информация и публикации в форумах.
- Содержимое веб-сайта: список альбомов грампластинок, каталог товаров и меню на обед.

Имеются следующие основные причины, по которым данные подобного рода следует хранить в базе данных, а не в файлах: удобство, одновременный доступ и безопасность. Программа базы данных намного упрощает поиск отдельных фрагментов информации и манипулирование ими. С помощью программы базы данных можно выполнять такие действия, как изменение адреса электронной почты пользователя `Duck29` на `ducky@ducks.example.com` за один раз. Если же хранить имена пользователей и адреса электронной почты в файле, изменить такой адрес будет намного сложнее, поскольку придется сначала прочитать прежнее содержимое файла, просматривать в нем каждую строку до тех пор, пока не будет найдено имя пользователя `Duck29`, изменить найденную строку и записать ее измененное содержимое обратно в файл. И если одновременно по одному запросу обновляется адрес электронной почты пользователя `Duck29`, а по другому запросу — запись для пользователя `Piggy56`, то одно обновление может быть утрачено, а еще хуже — испорчен файл, хранящий эти данные. В то же время программа базы данных способна автоматически выполнять все эти сложные операции одновременного доступа к данным.

Помимо удобства поиска информации, программа базы данных обычно предоставляет разные варианты управления доступом к информации в сравнении с файлами. Этот процесс предназначен для того, чтобы в программах на PHP можно было создавать, редактировать и удалять файлы на веб-сервере, не открывая бреши в системе его защиты для атак злоумышленников, способных внести необратимые изменения в сценарии PHP и файлы данных. Программа базы данных упрощает организацию уровней соответствующего доступа к информации. Она может быть настроена таким образом, чтобы одни виды информации можно было читать и изменять, а другие — только читать в программах на PHP. Но как бы ни было организовано управление доступом к базе данных, оно не

оказывает никакого влияния на порядок доступа к файлам на веб-сервере. Тот факт, что в программе на PHP можно изменять значения в базе данных, еще не означает, что атакующему злоумышленнику не удастся внести необратимые изменения в сами программы на PHP и HTML-файлы.

Термин *база данных* употребляется по-разному, когда речь заходит о вебприложениях. Базой данных можно считать массив структурированной информации; программу (например, MySQL или Oracle), управляющую этой структурированной информацией; или компьютер, на котором выполняется данная программа. В данной книге под термином *база данных* подразумевается массив структурированной информации. Программное обеспечение, управляющее этой информацией, называется *программой базы данных*, а компьютер, на котором выполняется эта программа, — *сервером базы данных*.

В этой главе применяется, главным образом, уровень абстракции программы базы данных в виде расширения PHP Data Objects (PDO — объекты данных PHP). Эта часть PHP упрощает взаимодействие программы на PHP с программой базы данных. Расширение PDO позволяет применять одни и те же функции в PHP для обращения к самым разным программам баз данных. Без расширения PDO придется полагаться на разные функции PHP, чтобы обращаться к отдельным программам баз данных. А некоторые оригинальные свойства программы базы данных могут быть доступны только с помощью функций, предназначенных для взаимодействия с ней.

Организация информации в базе данных

Информация в базе данных организована в *таблицах*, состоящих из строк и столбцов. (Столбцы иногда еще называют *полями*.) Каждый столбец в таблице содержит определенную категорию информации, а каждая строка — ряд значений для каждого столбца. Например, таблица, содержащая информацию о блюдах из меню, будет содержать отдельные столбцы для идентификатора, наименования, цены и наличия специй в каждом блюде, а каждая строка в данной таблице — группу значений для одного конкретного блюда, например `"1"`, `"Fried Bean Curd"` (Соевый творог из жареных бобов), `"5.50"` и `"0"` (означает отсутствие специй в блюде). Таблицу базы данных можно представить в виде простой электронной таблицы с именами столбцов в заглавии (рис. 8.1).

ID	Name	Price	Is spicy?
1	Fried Bean Curd	5.50	0
2	Braised Sea Cucumber	9.95	0
3	Walnut Bun	1.00	0
4	Eggplant with Chili Sauce	6.50	1

Рис. 8.1: Данные, организованные в таблицу

Но единственное отличие электронной таблицы от таблицы базы данных заключается в том, что строкам в таблице базы данных не присущ какой-то определенный порядок расположения. Если требуется извлечь данные из таблицы со строками, расположенными в определенном порядке (например, в алфавитном порядке по именам учащихся), то этот порядок нужно указать явно, запрашивая информацию в базе данных. О том, как это делается, см. во врезке “Урок языка SQL: команды `ORDER BY` и `LIMIT`"далее в этой главе.

Язык структурированных запросов (Structured Query Language — SQL) предназначен для составления запросов и указания команд для программы базы данных. Программа на PHP посылает соответствующие запросы SQL программе базы данных. Так, если по запросу (например, с целью найти все блюда со специями) из базы данных извлекается информация, программа базы данных

отправляет в ответ ряд строк таблицы, соответствующих данному запросу. А если по запросу (например, с целью ввести новое блюдо или удвоить цены на все блюда без специй) изменяется информация в базе данных, то программа базы данных отправляет в ответ сообщение о том, насколько успешно была выполнена данная операция.

Регистр букв по-разному учитывается в SQL. Так, в ключевых словах SQL регистр букв не учитывается, но во всех примерах из данной книги, где представлены запросы SQL, ключевые слова специально набраны прописными буквами, чтобы отличать их от остальных частей запросов. А в именах таблиц и столбцов, указываемых в запросах SQL, регистр букв обычно учитывается. Так, во всех примерах из данной книги, где представлены запросы SQL, имена таблиц и столбцов набраны строчными буквами, чтобы их легче было отличать от ключевых слов SQL. В любых литеральных значениях, указываемых в запросах SQL, также учитывается регистр букв. Например, имена `fried bean curd` и `FRIED Bean Curd` означают разные новые блюда, вводимые в базу данных.

Почти все запросы SQL, составляемые для применения в программах на PHP, основываются на одной из следующих команд SQL: `INSERT`, `UPDATE`, `DELETE` или `SELECT`. Каждая из этих команд описывается далее в этой главе. А в разделе "Создание таблицы базы данных" описывается команда `CREATE TABLE`, предназначенная для создания новых таблиц в базе данных.

Для более углубленного изучения языка SQL рекомендуется книга SQL: *полное руководство*, в которой дается общий обзор стандарта SQL, а также описание расширений SQL в MySQL, Oracle, PostgreSQL и Microsoft SQL Server. А дополнительные сведения о том, как пользоваться PHP вместе с MySQL, можно почерпнуть из книги Люка Веллинга и Лоры Томсон *Разработка веб-приложений с помощью PHP и MySQL, 4-е издание* (ISBN 978-5-8459-1574-0, пер. с англ., ИД "Вильямс 2010). И, наконец, книга *MySQL Cookbook* Поля Дюбуа (Paul DuBois; O'Reilly) служит отличным источником для поиска ответов на многие вопросы, возникающие в связи с применением SQL и MySQL. Все эти книги упоминались также в списке рекомендованной для чтения литературы, представленном в предисловии к данной книге.

Подключение к программе базы данных

Чтобы установить соединение с программой базы данных, следует создать новый объект класса `PDO`. В качестве аргумента конструктору класса `PDO` передается символьная строка, описывающая подключаемую базу данных. В итоге возвращается объект, которым можно пользоваться в остальной части программы для обмена информацией с программой базы данных.

В примере 8.1 демонстрируется вызов конструктора данного класса с помощью операции `new PDO()` для подключения к базе данных `restaurant`, развернутой на MySQL-сервере, работающем по адресу `db.example.com`. Для доступа к этой базе данных указаны имя пользователя `penguin` и пароль `top^hat`.

Пример 8.1. Подключение к программе базы данных с помощью объекта типа PDO

```
$db = new PDO('mysql:host=db.example.com;dbname=restaurant',
              'penguin','top^hat');
```

Символьная строка, передаваемая в качестве первого аргумента конструктору класса `PDO`, называется *именем источника данных* (DSN). Она начинается с префикса, обозначающего тип подключаемой программы базы данных. Затем следует двоеточие (`:`) и разделяемые точкой с запятой пары "ключ-значение", предоставляющие сведения о порядке подключения. Если для подключения к базе данных требуется указать имя пользователя и пароль, эти сведения передаются в качестве второго и третьего аргументов конструктору класса `PDO`.

Указание конкретных пар "ключ-значение" в имени DSN зависит от типа подключаемой программы базы данных. Несмотря на то что интерпретатор PHP позволяет подключаться к самым разным базам данных с помощью расширения `PDO`, такая возможность подключения должна быть разрешена при построении и установке данного интерпретатора на веб-сервере. Если при создании

объекта типа PDO вы получите сообщение "could not find driver" (не удалось найти драйвер), это означает, что при установке интерпретатора PHP не была задействована поддержка той базы данных, которой вы пытаетесь воспользоваться.

В табл. 8.1 перечислены префиксы и параметры DSN для наиболее употребительных программ баз данных, способных взаимодействовать с расширением PDO.

Таблица 8.1. Префиксы и параметры DSN для расширения PDO

Программа базы данных	Префикс DSN	Параметры DSN	Примечания
MySQL	**mysql**	**host, port, dbname, unix_socket, charset**	Параметр **unix_socket** служит для локальных подключений к MySQL. Рекомендуется использовать его или же параметры **host** и **port**, но не и то и другое вместе
PostgreSQL	**pgsql**	**host, port, dbname, user, password и прочие**	Вся строка подключения передается внутренней функции подключения к PostgreSQL, и поэтому в ней можно указывать любые параметры, перечисленные в документации на PostgreSQL, оперативно доступной по адресу **https://www.postgresql.org/docs/9.4/static/libpq-connect.html#LIBPQ-PARAMKEYWORDS**
Oracle	**oci**	**dbname, charset**	В качестве значения параметра **dbname** следует указать URI в форме ***//имя хоста:порт/база_данных*** для подключения к Oracle Instant Client или адресное имя, определенное в файле **tnsnames.ora**
SQLite	**sqlite**	**Отсутствуют**	После префикса для составления полного имени DSN следует указать путь к файлу базы данных SQLite или символьную строку "**:memory:**", чтобы воспользоваться базой данных, временно находящейся в оперативной памяти
ODBC	**odbc**	**DSN, UID, PWD**	В качестве значения для ключа DSN в символьной строке имени DSN следует указать имя, определенное в каталоге программных продуктов ODBC, или полную строку подключения к ODBC
MS SQL Server или Sybase	**mssql, Sybase, dblib**	**host, dbname, charset, appname**	В качестве значения параметра **appname** следует указать символьную строку, применяемую в статистических данных базы данных для описания подключения к ней. Префикс **mssql** указывается в том случае, если в интерпретаторе PHP применяются библиотеки SQL Server корпорации Microsoft; а префикс **sybase**, – если применяются библиотеки FreeTDS

Как было показано в примере 8.1, параметры `host` и `port` обозначают в имени DSN сетевой узел (хост) и порт подключения к серверу базы данных соответственно. Параметр `charset`, доступный в некоторых программах баз данных, обозначает порядок интерпретации символов, не относящихся к английскому алфавиту, в программе базы данных. Параметры `user` и `password` для базы данных PostgreSQL, а также параметры `UID` и `PWD` для базы данных ODBC обозначают порядок указания имени пользователя и его пароля в символьной строке с именем DSN для подключения к базе данных. Если эти параметры используются, их значения заменяют любое имя пользователя или пароль, передаваемые в качестве дополнительных аргументов конструктору класса `PDO`.

При удачном исходе операции `new PDO()` возвращается объект, применяемый для взаимодействия с базой данных. А если при подключении к ней возникнет затруднение, то будет сгенерировано исключение типа `PDOException`. Исключения, которые могут быть сгенерированы конструктором класса `PDO`, следует перехватить, чтобы проверить, удалось ли подключиться к базе данных, прежде чем продолжить выполнение программы. В примере 8.2 показано, как это обычно делается.

Пример 8.2. Перехват исключений, возникающих при ошибках подключения к базе данных

```
try {
    $db = new PDO('mysql:host=localhost;dbname=restaurant',
                  'penguin','top^hat');
    // сделать что-нибудь с объектом в переменной $db
} catch (PDOException $e) {
    print "Couldn't connect to the database: " . $e->getMessage();

}
```

Если в коде из примера 8.2 конструктор класса `PDO` сгенерирует исключение, то любой код, следующий в блоке оператора `try` после операции `new PDO()`, *не* будет далее выполнен. Вместо этого интерпретатор PHP перейдет непосредственно к блоку оператора `catch`, где отображается сообщение об ошибке

Так, если пароль `top^hat` окажется неверным для пользователя `penguin`, в коде из примера 8.2 будет выведено сообщение, аналогичное следующему:

```
Couldn't connect to the database: SQLSTATE[HY000] [1045] Access denied
for user 'penguin'@'client.example.com'
(using password: YES)

[ Не удалось подключиться к базе данных: SQLSTATE[HY000] [1045]
  Отказано в доступе пользователю 'penguin'@'client.example.com'
  (пароль использован: ДА) ]
```

Создание таблицы базы данных

Прежде чем сохранить или извлечь какую-нибудь информацию из таблицы базы данных, ее нужно создать. Как правило, это делается один раз, когда программе на PHP дается команда создать новую таблицу. А программа на PHP, пользующаяся этой таблицей, сможет читать и записывать в нее данные всякий раз, когда она выполняется. Но для этого не придется каждый раз воссоздавать таблицу. Если таблица базы данных похожа на электронную таблицу, то ее построение — на создание файла электронной таблицы. Как только такой файл будет создан, его можно неоднократно открывать для чтения и внесения в него изменений.

Для создания таблицы базы данных служит команда `CREATE TABLE` языка SQL, в которой указывается имя создаваемой таблицы, а также имена и типы всех ее столбцов. В примере 8.3 показано применение команды `CREATE TABLE` для создания таблицы `dishes`, приведенной на рис. 8.1.

Пример 8.3. Создание таблицы `dishes` в базе данных

```
CREATE TABLE dishes (
    dish_id INTEGER PRIMARY KEY,
    dish_name VARCHAR(255),
    price DECIMAL(4,2),
    is_spicy INT
)
```

В примере 8.3 создается таблица `dishes`, состоящая из четырех столбцов `dish_id`, `dish_name`, `price` и `is_spicy` и соответствующая виду, приведенному на рис. 8.1. В частности, столбцы `dish_id` и `is_spicy` содержат целочисленные значения, столбец `price` — десятичные числовые значения, а столбец `dish_name` — строковые значения.

После наименования команды `CREATE TABLE` указывается имя таблицы, а затем список разделяемых запятыми столбцов таблицы в круглых скобках. Предложение, в котором определяется каждый столбец, состоит из следующих двух частей: имени и типа столбца. Так, в команде из примера 8.3 указаны имена столбцов `dish_id`, `dish_name`, `price` и `is_spicy`, а также их типы `INTEGER`, `VARCHAR(255)`, `DECIMAL(4,2)` и `INT`.

Кроме того, для столбца `dish_id` указывается дополнительный тип `PRIMARY KEY`, сообщающий программе базы данных, что значения в данном столбце не подлежат дублированию в таблице. Это означает, что конкретное значение из столбца `dish_id` может одновременно находиться лишь в одной строке таблицы, что дает возможность программе базы данных, применяемой в примерах из этой главы, автоматически назначать в столбце `dish_id` новые единственные целочисленные значения идентификаторов блюд при вводе новых данных в таблицу `dishes`. В других программах баз данных применяется иной синтаксис для автоматического назначения единственных целочисленных значений идентификаторов. Например, в MySQL для этой цели служит ключевое слово `AUTO_INCREMENT`, в PostgreSQL — порядковые типы данных, а в Oracle — числовые последовательности.

Как правило, обозначения целочисленных типов данных `INT` и `INTEGER` употребляются в командах SQL попеременно. Но для SQLite характерна необходимость точно указывать целочисленный тип столбца как `INTEGER`, чтобы новые единственные значения автоматически назначались в соответствии с заданным типом столбца `PRIMARY KEY`.

Для некоторых типов столбцов таблицы в круглых скобках дополнительно указывается длина или информация о форматировании столбца. Например, тип `VARCHAR(255)` означает столбец символьных строк переменной длины, но не более 255 символов, а тип `DECIMAL(4,2)` — столбец десятичных чисел с двумя цифрами после десятичной точки и четырьмя цифрами в целом. В табл. 8.2 перечислены типы столбцов, наиболее употребительные в таблицах базы данных.

Таблица 8.2. Типы столбцов, наиболее употребительные в таблицах базы данных

Тип столбца	Описание
`VARCHAR(`*`length`*`)`	Символьная строка переменной длины, определяемой параметром ***`length`***
`INT`	Целое число
`BLOB`[1]	Строковые или двоичные данные длиной до **`64`** Кбайт
`DECIMAL(`*`total_digits, decimal_places`*`)`	Десятичное число, где общее количество цифр определяется параметром ***`total_digits`***, а количество цифр после десятичной точки – параметром ***`decimal_places`***
`DATETIME`[2]	Дата и время, например **`1975-03-10 19:45:03`** или **`2038-01-18 22:14:07`**

[1] Этот тип столбца в PostgreSQL называется `BYTEA` вместо `BLOB`.

[2] Этот тип столбца в Oracle называется `DATE` вместо `DATETIME`.

В различных программах баз данных поддерживаются разные типы столбцов, хотя во всех этих программах должны поддерживаться типы столбцов, перечисленные в табл. 8.2. Допустимые диапазоны чисел, которые могут поддерживаться в числовых столбцах таблиц базы данных, а также максимальные размеры текстовых столбцов зависят от применяемой программы базы данных. Например, в MySQL допускаются столбцы типа `VARCHAR` с максимальной длиной до 255 символов, тогда как в Microsoft SQL Server — до 8000 символов. Поэтому уточняйте все эти особенности в справочном руководстве по применяемой базе данных.

Чтобы создать конкретную таблицу в базе данных, необходимо послать ей запрос SQL с командой `CREATE TABLE`. С этой целью следует вызвать метод `exec()` после подключения к базе данных с помощью операции `new PDO()`, как показано в примере 8.4.

Пример 8.4. Отправка запроса SQL с командой `CREATE TABLE` программе базы данных

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $q = $db->exec("CREATE TABLE dishes (
                    dish_id INT,
                    dish_name VARCHAR(255),
                    price DECIMAL(4,2),
                    is_spicy INT
    )");
} catch (PDOException $e) {
    print "Couldn't create table: " . $e->getMessage();
}
```

Более подробно метод `exec()` поясняется в следующем разделе. А вызовом метода `$db->setAttribute()` в коде из примера 8.4 гарантируется, что расширение PDO сгенерирует исключения, если возникнут затруднения при обработке запросов, а не подключении к базе данных. Обработка ошибок средствами PDO также обсуждается в следующем разделе.

В отличие от команды `CREATE TABLE`, команда `DROP TABLE` удаляет таблицу вместе с хранящейся в ней информацией из базы данных. В примере 8.5 демонстрируется синтаксис запроса SQL на удаление таблицы `dishes` из базы данных.

Пример 8.5. Удаление таблицы из базы данных

```
DROP TABLE dishes
```

Как только таблица будет удалена из базы данных, ее уже нельзя будет восстановить. Поэтому пользуйтесь командой `DROP TABLE` осмотрительно!

Ввод информации в базу данных

При удачном исходе подключения к базе данных из операции `new PDO()` возвращается объект, предоставляющий доступ к информации в этой базе данных. Вызывая методы этого объекта, можно посылать запросы SQL программе базы данных и в ответ получать от нее результаты. Чтобы ввести какую-нибудь информацию в базу данных, следует передать запрос SQL с командой `INSERT` методу `exec()` этого объекта, как показано в примере 8.6.

Пример 8.6. Ввод информации в базу данных с помощью метода `exec()`

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $affectedRows = $db->exec(
                    "INSERT INTO dishes (dish_name, price, is_spicy)
                            VALUES ('Sesame Seed Puff', 2.50, 0)");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

Метод `exec()` возвращает количество строк таблицы, затребованных в запросе SQL, отправленном серверу базы данных. В данном случае ввод одной строки в таблицу привел к возврату значения **1**, поскольку в таблицу была введена одна затребованная строка.

Если же при выполнении команды `INSERT` возникнут какие-нибудь затруднения, то будет сгенерировано соответствующее исключение. В примере 8.7 демонстрируется попытка выполнить команду `INSERT` с неверно указанным именем `dish_size` столбца, который отсутствует в таблице `dishes`.

Пример 8.7. Проверка ошибок выполнения запроса SQL с помощью метода `exec()`

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $affectedRows = $db->exec(
                    "INSERT INTO dishes (dish_size, dish_name,
                                         price, is_spicy)
                                 VALUES ('large', 'Sesame Seed Puff',
                                         2.50, 0)");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

Благодаря тому что вызов метода `$db->setAttribute[]` предписывает расширению PDO сгенерировать исключение в любой момент, когда возникнет ошибка, при выполнении кода из примера 8.7 на экран выводится следующее сообщение:

```
Couldn't insert a row: SQLSTATE[HY000]: General error: 1 table dishes
has no column named dish_size

[ Не удалось ввести строку в таблицу: SQLSTATE[HYOOO]: Общая ошибка: 1
  В таблица dishes отсутствует столбец с именем dish_size ]
```

В расширении PDO имеются три режима выдачи ошибок: исключения, негласный и предупреждения. Режим исключения при ошибках, активизируемый при вызове метода `$db->setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)`, лучше всего подходит для целей отладки, поскольку он позволяет убедиться в отсутствии ошибок при обращении к базе данных. Если не обработать исключение, генерируемое средствами PDO, выполнение программы на PHP прервется.

Два других режима выдачи ошибок требуют проверки значений, возвращаемых из вызовов методов PDO, чтобы сначала определить, возникла ли ошибка, а затем воспользоваться дополнительными методами PDO для поиска информации о данной ошибке. Негласный режим выдачи ошибок устанавливается по умолчанию. Как и остальные методы PDO, метод `exec()` возвращает логическое значение `false`, если ему не удастся выполнить свою задачу. В примере 8.8 проверяется значение, возвращаемое методом `exec()`, а затем из расширения PDO вызывается метод `errorInfo()` для получения подробных сведений о возникшей ошибке.

Пример 8.8. Работа с базой данных в негласном режиме выдачи ошибок

```
// Конструктор всегда генерирует исключение,
// если ему не удастся выполнить свою задачу
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Couldn't connect: " . $e->getMessage();
}
$result = $db->exec("INSERT INTO dishes (dish_size, dish_name,
                                         price, is_spicy)
                                 VALUES ('large', 'Sesame Seed Puff',
```

```
                                        2.50, 0)");
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]},
                    Message={$error[2]}\n";
}
```

При выполнении кода из примера 8.8 на экран выводится следующее сообщение:

```
Couldn't insert!
SQL Error=HY000, DB Error=1, Message=
    table dishes has no column named dish_size

[ Вставить не удалось!
  Ошибка SQL=HY000, Ошибка БД=1, Сообщение=
  В таблице отсутствует столбец с именем dish_size ]
```

В коде из примера 8.8 значение, возвращаемое из метода `exec()`, сравнивается с логическим значением `false` с помощью операции тождественности (**`===`**), чтобы отличить фактическую ошибку (`false`) от успешного выполнения запроса, который просто не затронул ни одной из строк в таблице. Затем метод `errorInfo()` возвращает трехэлементный массив со сведениями о возникшей ошибке. Первый элемент этого массива содержит код ошибки `SQLSTATE`, Такие коды ошибок в основном стандартизированы в разных программах баз данных. В данном случае возвращается универсальный код **`HY000`** для всех общих ошибок. Второй элемент данного массива содержит код ошибки, характерный для конкретной применяемой базы данных. И третий элемент данного массива содержит текст сообщения, описывающего возникшую ошибку.

Режим предупреждения об ошибках активизируется при установке значения **`PDO::ERRMODE_WARNING`** в атрибуте `PDO::ATTR_ERRMODE`, как показано в примере 8.9. В этом режиме методы PDO ведут себя таким же образом, как и в негласном режиме, т.е. при появлении ошибки никаких исключений не возникает, а просто возвращается логическое значение `false`. Но, кроме того, интерпретатор PHP формирует сообщение об ошибке на уровне предупреждения. Это сообщение может быть отображено на экране или выведено в файл регистрации в зависимости о того, каким образом в программе организована обработка ошибок. О том, как организовать вывод сообщений об ошибках, речь пойдет в разделе “Управление выводом сообщений об ошибках” главы 12.

Пример 8.9. Работа с базой данных в режиме предупреждения об ошибках

```
// Конструктор всегда генерирует исключение,
// если ему не удастся выполнить свою задачу
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Couldn't connect: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$result = $db->exec("INSERT INTO dishes (dish_size, dish_name,
                                         price, is_spicy)
                             VALUES ('large', 'Sesame Seed Puff',
                                         2.50, 0)");
if (false === $result) {
    $error = $db->errorInfo();
    print "Couldn't insert!\n";
```

```
    print "SQL Error={$error[0]}, DB Error={$error[1]},
                    Message={$error[2]}\n";
}
```

Выполнение кода из примера 8.9 приводит к такому же результату, как и в примере 8.8, но на экран выводится следующее предупреждающее сообщение:

```
PHP Warning: PDO::exec(): SQLSTATE[HY000]:
    General error: 1 table dishes
has no column named dish_size in error-warning.php on line 10

[ Предупреждение PHP::exec(): SQLSTATE[HY000]: Общая ошибка: 1
  В таблице отсутствует столбец с именем dish_size, указанный в
  строке код 10 исходного файла error-warning.php ]
```

Урок языка SQL: команда INSERT

Команда **INSERT** вводит строку в таблицу базы данных. Синтаксис команды **INSERT** приведен в примере 8.10.

Пример 8.10. Ввод данных

```
INSERT INTO имя_таблицы (столбец1[,столбец2, столбец3, ...])
       VALUES (значение1[,значение2, значение3, ...])
```

По запросу SQL с командой **INSERT** из примера 8.11 новое блюдо вводится в таблицу **dishes**.

Пример 8.11. Ввод нового блюда в таблицу

```
INSERT INTO dishes (dish_id, dish_name, price, is_spicy)
       VALUES (1, 'Braised Sea Cucumber', 6.50, 0)
```

Такие строковые значения, как **Braised Sea Cucumber** (Жареный морской огурец), должны быть заключены в одиночные кавычки, когда они применяются в запросе SQL. А поскольку одиночные кавычки служат в качестве ограничителей символьных строк, то их необходимо экранировать, указав по две одиночные кавычки подряд в составляемом запросе. В примере 8.12 демонстрируется, как ввести блюдо **General Tso's Chicken** (Цыпленок генерала Цо) в таблицу **dishes**.

Пример 8.12. Заключение строкового значения в кавычки

```
INSERT INTO dishes (dish_id, dish_name, price, is_spicy)
       VALUES (2, 'General Tso''s Chicken', 6.75, 1)
```

Количество столбцов, перечисляемых в круглых скобках перед предложением **VALUES**, должно совпадать с количеством значений, указываемых в круглых скобках после предложения **VALUES**. Чтобы ввести строку, содержащую значения лишь для некоторых столбцов таблицы, достаточно перечислить в запросе SQL эти столбцы и их соответствующие значения, как в примере 8.13.

Пример 8.13. Ввод строки со значениями для некоторых столбцов таблицы

```
INSERT INTO dishes (dish_name, is_spicy)
       VALUES ('Salt Baked Scallops', 0)
```

Пример 8.14. Ввод строки со значениями для всех столбцов таблицы

```
INSERT INTO dishes
       VALUES (1, 'Braised Sea Cucumber', 6.50, 0)
```

Чтобы изменить данные в таблице, следует составить запрос SQL с командой `UPDATE` и вызвать метод `exec()`. В примере 8.15 приведены некоторые образцы запросов SQL с командой `UPDATE`, выполняемых с помощью метода `exec()`.

Пример 8.15. Изменение данных в таблице с помощью метода exec()

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // Кабачок под соусом, приправленным красным стручковым
    // перцем, считается блюдом со специями. Если не имеет
    // значения, сколько строк таблицы затронет данный запрос,
    // то сохранять значение, возвращаемое методом exec(),
    // совсем не обязательно
    $db->exec("UPDATE dishes SET is_spicy = 1
               WHERE dish_name = 'Eggplant with Chili Sauce'");
    // Омар под соусом, приправленным красным стручковым
    // перцем, считается дорогим блюдом со специями
    $db->exec("UPDATE dishes SET is_spicy = 1, price=price * 2
               WHERE dish_name = 'Lobster with Chili Sauce'");
} catch (PDOException $e) {
    print "Couldn't insert a row: " . $e->getMessage();
}
```

Чтобы удалить данные из таблицы, следует составить запрос SQL с командой `DELETE` и снова вызвать метод `exec()`. В примере 8.16 приведены два запроса SQL с командой `DELETE`, выполняемых с помощью метода `exec()`.

Пример 8.16. Удаление данных из таблицы с помощью метода exec()

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // удалить дорогие блюда из таблицы
    if ($make_things_cheaper) {
        $db->exec("DELETE FROM dishes WHERE price > 19.95");
    } else {
        // или же удалить из нее все блюда
        $db->exec("DELETE FROM dishes");
    }
} catch (PDOException $e) {
    print "Couldn't delete rows: " . $e->getMessage();
}
```

Урок языка SQL: команда UPDATE

Команда **UPDATE** изменяет данные, уже находящиеся в таблице. Синтаксис команды **UPDATE** показан в примере 8.17.

Пример 8.17. Обновление данных в таблице

```
UPDATE имя_таблицы SET столбец1= значение1[,столбец2= значение2,
       столбец3= значение3, ...] [WHERE логическое_выражение]
```

Значение, которое изменяется в столбце, может быть строковым или числовым, как показано в примере 8.18. Строки кода, начинающиеся в примере 8.18 со знака ;, содержат комментарии к запросу SQL.

Пример 8.18. Установка строкового или числового значения в столбце таблицы

```
; изменить цену на 5.50 во всех строках таблицы
UPDATE dishes SET price = 5.50
; изменить на 1 значение в столбце is_spicy во всех строках таблицы
UPDATE dishes SET is_spicy = 1
```

В качестве значения может быть также указано выражение, включающее имена столбцов таблицы. Так, по запросу SQL в примере 8.19 удваивается цена каждого блюда.

Пример 8.19. Употребление имени столбца в выражении, указываемом в команде UPDATE

```
UPDATE dishes SET price = price * 2
```

В рассмотренных до сих пор запросах SQL с командой **UPDATE** изменялись все строки в таблице **dishes**. Чтобы изменить только некоторые строки, в запрос SQL с командой **UPDATE** следует ввести предложение **WHERE** с логическим выражением, описывающим изменяемые строки. Таким образом, изменения по такому запросу произойдут только в тех строках таблицы, которые совпадают с условием, заданным в предложении **WHERE**. В примере 8.20 демонстрируются два запроса SQL с командой **UPDATE**, содержащей предложение **WHERE.**

Пример 8.20. Употребление предложения WHERE в команде UPDATE

```
; изменить состояние блюда "Кабачок под соусом, приправленным
; красным стручковым перцем" на блюдо со специями
UPDATE dishes SET is_spicy = 1
              WHERE dish_name = 'Eggplant with Chili Sauce'
; уменьшить цену на блюдо "Цыпленок генерала Цо"
UPDATE dishes SET price = price - 1
              WHERE dish_name = 'General Tso's Chicken'
```

Более подробно предложение **WHERE** поясняется далее, во врезке “Урок языка SQL: команда **SELECT**”.

Напомним, что метод `exec()` возвращает количество строк таблицы, измененных или удаленных по команде `UPDATE` или `DELETE` соответственно. Возвращаемым значением можно воспользоваться, чтобы выяснить, сколько строк таблицы было затронуто при выполнении запроса SQL. Так, в примере 8.21 сообщается, в каком количестве строк таблицы были изменены цены на блюда по запросу SQL с командой `UPDATE`.

Пример 8.21. Выявление количества строк таблицы, затронутых при выполнении запроса SQL с командой UPDATE

```
// увеличить цены на некоторые блюда
$count = $db->exec("UPDATE dishes
                    SET price * price + 5 WHERE price > 3");
print 'Changed the price of ' . $count . ' rows.';
```

Если в таблице `dishes` будут найдены две строки, где цена на блюдо превышает **3**, то при выполнении кода из примера 8.21 на экран будет выведен следующий результат:

```
Changed the price of 2 rows.
```

Урок языка SQL: команда `DELETE`

Команда **`DELETE`** удаляет строки из таблицы. В примере 8.22 приведен синтаксис команды **`DELETE`**.

Пример 8.22. Удаление отдельных строк из таблицы

```
DELETE FROM имя_таблицы [WHERE логическое_выражение]
```

В отсутствие предложения **`WHERE`** команда **`DELETE`** удалит все строки из таблицы. Так, в примере 8.23 очищается вся таблица **`dishes`**.

Пример 8.23. Удаление всех строк из таблицы

```
DELETE FROM dishes
```

При наличии предложения **`WHERE`** команда **`DELETE`** удалит из таблицы только те строки, которые совпадают с условием, указанным в предложении **`WHERE`**. В примере 8.24 приведены два запроса SQL с командой **`DELETE`**, содержащей предложение **`WHERE`**.

Пример 8.24. Удаление некоторых строк из таблицы

```
; удалить из таблицы строки, где цена на блюда превышает 10.00
DELETE FROM dishes WHERE price > 10.00
; удалить из таблицы строки, где значение в столбце dish_name
; точно соответствует наименованию блюда "Walnut Bun" (Булочка
; с грецкими орехами)
DELETE FROM dishes WHERE dish_name = 'Walnut Bun'
```

В языке SQL отсутствует команда **`UNDELETE`**, поэтому осмотрительно пользуйтесь командой **`DELETE`**.

Безопасный ввод данных из формы

Как пояснялось в разделе “HTML и JavaScript” главы 7, вывод на экран не прошедших санитарную очистку данных из формы может сделать веб-сайт и его посетителей уязвимыми к атакам типа межсайтового выполнения сценариев. Уязвимость к аналогичным атакам типа умышленного внесения запросов SQL может вызвать применение не прошедших санитарную очистку данных из формы в запросах SQL. Рассмотрим в качестве примера форму, в которой пользователь может предложить новое блюдо. Эта форма содержит текстовый элемент `new_dish_name`, где пользователь может ввести название нового блюда. В примере 8.25 новое блюдо вводится в таблицу dishes с помощью метода `exec()`, но такой способ уязвим к атакам типа умышленного внесения запросов SQL.

Пример 8.25. Небезопасный ввод данных из формы

```
$db->exec("INSERT INTO dishes (dish_name)
                  VALUES ('$_POST[new_dish_name]')");
```

Если переданное значение приемлемо для сохранения в столбце `new_dish_name` таблицы `dishes` (например, название блюда **`Fried Bean Curd`**), то запрос будет выполнен успешно. В соответствии с обычными для PHP правилами вставки символьных строк, заключаемых в двойные кавычки, запрос с командой

```
INSERT INTO dishes (dish_name)
       VALUES ('Fried Bean Curd')
```

оказывается вполне приемлемым и достоверным. Но если в запросе имеется знак апострофа, то может возникнуть осложнение. Так, если для сохранения в столбце `new_dish_name` передано наименование блюда **`General Tso's Chicken`**, то запрос на его ввод в таблицу будет следующим:

```
INSERT INTO dishes (dish_name)
       VALUES ('General Tso's Chicken')
```

Такой запрос может вызвать недоразумение в программе базы данных. Она посчитает, что апостроф между символами `Tso` и `s` завершает символьную строку, и поэтому символы `s Chicken'`, следующие после второй одиночной кавычки, будут расценены как нежелательная синтаксическая ошибка.

Хуже того, пользователь, умышленно стремящийся вызвать осложнения, может ввести специально составленные данные, чтобы нанести серьезный ущерб. Рассмотрим следующие неприглядные входные данные:

```
x'); DELETE FROM dishes; INSERT INTO dishes (dish_name) VALUES ('y.
```

При вставке эти данные превращаются в следующий запрос:

```
INSERT INTO DISHES (dish_name) VALUES ('x');
DELETE FROM dishes; INSERT INTO dishes (dish_name) VALUES ('y')
```

В некоторых базах данных допускается передавать несколько запросов, разделяемых точками с запятой в одном вызове метода `exec()`. В таких базах данных приведенные выше входные данные могут вызвать серьезные нарушения в таблице `dishes`, поскольку сначала вводится блюдо `x`, затем удаляются все блюда и далее вводится блюдо `y`.

Передав специально составленное значение в форме, злонамеренный пользователь может внести произвольные команды SQL в программу базы данных. Во избежание этого следует экранировать специальные символы (и прежде всего, апостроф) в запросах SQL. Чтобы упростить дело, в PDO предоставляются вспомогательные средства, называемые *подготовленными операторами*.

Подготовленные операторы позволяют разделить выполнение запроса на две стадии. Сначала методу `prepare()` из расширения PDO предоставляется вариант запроса со знаком **`?`** в тех местах запроса SQL, где требуется указать значения. Этот метод возвращает объект типа `PDOStatement`. Затем для объекта типа `PDOStatement` вызывается метод `execute()`, которому передается массив значений, подставляемых вместо замещающих знаков **`?`**. Эти значения заключаются надлежащим образом в кавычки перед вводом в запрос, чтобы защитить от атак типа умышленного внесения запросов SQL. Безопасный вариант запроса SQL из примера 8.25 приведен в примере 8.26.

Пример 8.26. Безопасный ввод данных из формы

```
$stmt = $db->prepare('INSERT INTO dishes (dish_name) VALUES (?)');
$stmt->execute(array($_POST['new_dish_name']));
```

Заключать в кавычки замещающий знак в запросе не нужно, поскольку это делается в PDO автоматически. Если же требуется указать в запросе несколько значений, соответствующее количество замещающих знаков следует ввести в запрос и массив значений. Так, в примере 8.27 приведен запрос SQL с тремя замещающими знаками.

Пример 8.27. Употребление нескольких замещающих знаков в запросе SQL

```
$stmt = $db->prepare('INSERT INTO dishes (dish_name, price, is_spicy)
                      VALUES (?,?,?)');
$stmt->execute(array($_POST['new_dish_name'], $_POST['new_price'],
$_POST('is_spicy']));
```

Законченная форма для ввода записей в базу данных

Все рассмотренные до сих пор особенности взаимодействия с базой данных сочетаются в примере 8.28 с кодом обработки форм из главы 7 для построения законченной программы, отображающей форму, проверяющей достоверность переданных данных и затем сохраняющей данные в таблице базы данных. В этой форме отображаются элементы ввода наименования блюда, цены на блюдо и принадлежности его к блюдам со специями. Информация из переданной на обработку формы вводится далее в таблицу `dishes` базы данных.

Исходный код из примера 8.28 основан на классе `FormHelper`, определенный в коде из примера 7.29. Чтобы не повторяться, в примере 8.28 предполагается, что исходный код этого класса сохранен в файле `FormHelper.php`, который загружается в строке кода `require 'FormHelper.php'`, находящейся в самом начале данной программы.

Пример 8.28. Программа для ввода записей в таблицу dishes базы данных

```
<?php

// загрузить вспомогательный класс для составления форм
require 'FormHelper.php';

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Can't connect: " . $e->getMessage();
    exit();
}
// установить исключения при ошибках в базе данных
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if($errors) {
        show_form($errors);
```

```
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
     // установить свои значения по умолчанию:
     // цена составляет 5 долларов США
     $defaults = array('price' => '5.00');

     // создать объект $form с надлежащими свойствами по умолчанию
     $form = new FormHelper($defaults);

     // Ради ясности весь код HTML-разметки и отображения
     // формы вынесен в отдельный файл
     include 'insert-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();

    // обязательное наименование блюда
    $input['dish_name'] = trim($_POST['dish_name'] ?? '');
    if (! strlen($input['dish_name'])) {
        $errors[] = 'Please enter the name of the dish.';
    }

    // цена должна быть указана достоверным положительным числом
    // с плавающей точкой
    $input['price'] = filter_input(INPUT_POST, 'price',
                                   FILTER_VALIDATE_FLOAT);
    if ($input['price'] <= 0) {
        $errors[] = 'Please enter a valid price.';
    }

    // по умолчанию в элементе ввода is_spicy устанавливается
    // значение 'no'
    $input['is_spicy'] = $_POST['is_spicy'] ?? 'no';

    return array($errors, $input);
}

function process_form($input) {
    // получить в этой функции доступ к глобальной переменной $db
    global $db;
    // установить в переменной $is_spicy значение в зависимости
    // от состояния одноименного флажка
```

```
    if ($input['is_spicy'] == 'yes') {
        $is_spicy = 1;
    } else {
        $is_spicy = 0;
    }

    // ввести новое блюдо в таблицу базы данных
    try {
        $stmt = $db->prepare('INSERT INTO dishes (dish_name,
                                                  price,
                                                  is_spicy)
                                          VALUES (?,?,?)');
        $stmt->execute(array($input['dish_name'],
                             $input['price'], $is_spicy));
        // сообщить пользователю о вводе блюда в базу данных
        print 'Added ' . htmlentities($input['dish_name']) .
              ' to the database.';
    } catch (PDOException $e) {
        print "Couldn't add your dish to the database.";
    }
}

?>
```

Программа из примера 8.28 имеет такую же основную структуру, как и в примерах обработки форм, приведенных в главе 7. Она состоит из функций для отображения, проверки достоверности, обработки формы и некоторой глобальной логики, определяющей порядок вызова этих функций. Новыми фрагментами в данной программе являются глобальный код, устанавливающий соединение с базой данных, а также операции с базой данных в функции `process_form()`.

Код, устанавливающий соединение с базой данных, следует после оператора `require` и перед условным оператором `if($_SERVER['REQUEST_METHOD']=='POST')`. В операции `new PDO()` устанавливается соединение с базой данных, а в ряде последующих строк кода сначала проверяется, было ли подключение к базе данных произведено успешно, а затем устанавливается режим исключения для обработки ошибок.

Функция `show_form()` отображает HTML-разметку формы из файла `insert-form.php`. Содержимое этого файла приведено в примере 8.29.

Пример 8.29. Форма для ввода записей в таблицу `dishes` базы данных

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
        <td>You need to correct the following errors:</td>
         <td><ul>
           <?php foreach ($errors as $error) { ?>
             <li><?= $form->encode($error) ?></li>
           <?php } ?>
         </ul></td>
<?php } ?>

<tr>
```

```
  <td>Dish Name:</td>
  <td><?= $form->input('text', ['name' => 'dish_name']) ?>
  </td>
</tr>
<tr>
  <td>Price:</td>
  <td><?= $form->input('text', ['name' => 'price']) ?></td>
</tr>

<tr>
  <td>Spicy:</td>
  <td><?= $form->input('checkbox',['name' => 'is_spicy',
                        'value' => 'yes']) ?> Yes</td>
</tr>

<tr><td colspan="2" align="center">
  <?= $form->input('submit',['name' => 'save',
                    'value' => 'Order']) ?>
  </td></tr>

</table>
</form>
```

Помимо подключения, все остальное взаимодействие с базой данных происходит в функции `process_form()`. Сначала в строке кода `global $db` делается ссылка на переменную `$db` подключения к базе данных вместо менее удобной ссылки на элемент глобального массива `$GLOBALS ['db']`. Столбец `is_spicy` таблицы `dishes` содержит значение **1** в строках с блюдами со специями и значение **0** в строках с блюдами без специй. Поэтому локальной переменной `$is_spicy` присваивается далее подходящее значение в условном операторе `if()`, исходя из того, что было передано в элементе глобального массива `$input['is_spicy']`.

После этого вызываются методы `prepare()` и `execute()`, которые фактически вводят новую информацию в базу данных. Три замещающих знака в команде `INSERT` заполняются переменными `$input['dish_name']`, `$input['price']` и `$is_spicy`. Для столбца `dish_id` никакого значения указывать не нужно, поскольку он автоматически заполняется средствами самой программы базы данных SQLite. И, наконец, в функции `process_form()` выводится сообщение, извещающее пользователя о вводе блюда в таблицу базы данных. А функция `htmlentities()` защищает от появления любых дескрипторов HTML-разметки и элементов кода JavaScript в названии блюда. Методы `prepare()` и `execute()` вызываются в блоке оператора `try`, и если что-нибудь в них пойдет не так, то на экран будет выведено соответствующее сообщение об ошибке.

Извлечение информации из базы данных

Для извлечения информации из базы данных служит метод `query()`, которому передается запрос SQL для базы данных. Этот метод возвращает объект типа `PDOStatement`, предоставляющий доступ к строкам, извлекаемым из таблицы базы данных. Всякий раз, когда для этого объекта вызывается метод `fetch()`, по запросу из таблицы возвращается очередная строка. А если все строки в таблице исчерпаны, то метод `fetch()` возвратит значение, которое вычисляется как ложное (`false`), чем удобно воспользоваться в цикле `while()`, как показано в примере 8.30.

Пример 8.30. Извлечение строк из таблицы с помощью методов `query()` и `fetch()`

```
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
    print "$row[dish_name], $row[price] \n";
}
```

При выполнении кода из примера 8.30 на экран выводится следующий результат:

```
Walnut Bun, 1
Cashew Nuts and White Mushrooms, 4.95
Dried Mulberries, 3
Eggplant with Chili Sauce, 6.5
```

На первом шаге цикла `while()` метод `fetch()` возвращает массив, содержащий значения **`Walnut Bun and 1`**. Этот массив присваивается переменной `$row`. А поскольку непустой массив вычисляется как истинный (`true`), то код в теле цикла `while()` продолжает выполняться, выводя на экран данные из первой строки таблицы, возвращаемой по запросу с командой `SELECT`. И это происходит еще три раза. На каждом шаге цикла `while()` метод `fetch()` возвращает очередную строку из того ряда строк таблицы, которые возвращаются по запросу с командой `SELECT`. Когда же все возвращаемые по запросу строки будут исчерпаны, метод `fetch()` возвратит значение, которое вычисляется как ложное (`false`), и на этом цикл `while()` завершится.

По умолчанию метод `fetch()` возвращает массив с числовыми и строковыми ключами. Начиная с нуля, числовые ключи содержат значения из каждого столбца в строке. Аналогично имена строковых ключей получают имена соответствующих столбцов таблицы. Так, в примере 8.30 те же самые результаты можно было бы вывести на экран из элементов массива `$row[0]` и `$row[1]`.

Если требуется выяснить, сколько строк таблицы возвращено по запросу с командой `SELECT`, единственный надежный способ сделать это — извлечь из таблицы все нужные строки и подсчитать их. Для этой цели объект типа `PDOStatement` предоставляет метод `rowCount()`, но он пригоден не для всех баз данных. Если же в таблице имеется небольшое количество строк и все они должны быть использованы в программе, то лучше воспользоваться методом `fetchAll()`, чтобы разместить все строки таблицы в массиве, не прибегая к организации цикла, как показано в примере 8.31.

Пример 8.31. Извлечение всех строк из таблицы без организации цикла

```
$q = $db->query('SELECT dish_name, price FROM dishes');
// Переменная $rows будет содержать четырехэлементный
// массив, в каждом элементе которого находится по одной
// строке, извлекаемой из таблицы базы данных
$rows = $q->fetchAll();
```

Если же в таблице имеется столько строк, что извлекать их полностью непрактично, то программе базы данных можно направить запрос на автоматический подсчет строк с помощью функции `COUNT()` языка SQL. Например, по запросу `SELECT COUNT(*) FROM dishes` из таблицы `dishes` возвратится одна строка с единственным столбцом, значение которого равно количеству строк во всей таблице.

Урок языка SQL: команда SELECT

Команда **SELECT** извлекает информацию из таблицы базы данных. В примере 8.32 демонстрируется синтаксис команды **SELECT**.

Пример 8.32. Извлечение информации из таблицы базы данных

```
SELECT столбец1[,столбец2, столбец3, ...] FROM имя_таблицы
```

По запросу с командой **SELECT** в примере 8.33 извлекаются столбцы **dish_name** и **price** из всех строк таблицы **dishes**.

Пример 8.33. Извлечение столбцов dish_name и price

```
SELECT dish_name, price FROM dishes
```

Ради краткости вместо списка столбцов можно указать знак *****. В этом случае из таблицы будут извлечены все столбцы. Так, по запросу с командой **SELECT** в примере 8.34 из таблицы **dishes** извлекаются все столбцы.

*Пример 8.34. Применение знака * в запросе SQL с командой SELECT*

```
SELECT * FROM dishes
```

Чтобы ограничить выполнение команды **SELECT** извлечением только определенных строк из таблицы, эту команду следует дополнить предложением **WHERE**. По запросу с командой **SELECT** из таблицы возвращаются только те строки, которые соответствуют условию, указанному в предложении **WHERE**. Это предложение указывается после имени таблицы, как показано в примере 8.35.

Пример 8.35. Ограничение, накладываемое на строки, возвращаемые из таблицы по запросу SQL с командой SELECT

```
SELECT столбец1[, столбец2, столбец3, ...] FROM имя__таблицы WHERE
логическое_выражение
```

В той части запроса, где указано ***логическое_выражение***, описываются строки, которые требуется извлечь из таблицы. В примере 8.36 демонстрируется ряд запросов SQL с командой **SELECT** и предложением **WHERE**.

Пример 8.36. Извлечение определенных блюд из таблицы

```
; извлечь блюда по цене свыше 5.00
SELECT dish_name, price FROM dishes WHERE price > 5.00

; извлечь блюда, названия которых точно соответствуют "Walnut Bun"
SELECT price FROM dishes WHERE dish_name = 'Walnut Bun'

; извлечь блюда по цене свыше 5.00, но не больше 10.00
SELECT dish_name FROM dishes WHERE price > 5.00 AND price <= 10.00

; извлечь блюда по цене свыше 5.00, но не больше 10.00 или же
; блюда, названия которых точно соответствуют "Walnut Bun", но
; по любой цене
SELECT dish_name, price FROM dishes WHERE (price > 5.00 AND price <= 10.00)
       OR dish_name = 'Walnut Bun'
```

В табл. 8.3 перечислены операции, которые можно употреблять в предложении WHERE.

Таблица 8.3. Операции, употребляемые в предложении `WHERE` запроса SQL

Операция	Описание
=	Равно (соответствует операции == в PHP)
<>	Не равно (соответствует операции != в PHP)
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
AND	Логическая операция И (соответствует операции && в PHP)
OR	Логическая операция ИЛИ (соответствует операции || в PHP)
()	Группирование

Если по запросу предполагается возвратить только одну строку из таблицы, вызовы методов `query()` и `fetch()` можно связать в цепочку. Такой прием применяется в примере 8.37 для отображения самого дешевого блюда в таблице `dishes`. Части `ORDER BY` и `LIMIT` запроса в данном примере поясняются далее, во врезке "Урок языка SQL: предложения `ORDER BY` и `LIMIT`".

Пример 8.37. Извлечение строки из таблицы связыванием методов `query()` и `fetch()` в цепочку

```
cheapest_dish_info = $db->query('SELECT dish_name, price
                                 FROM dishes
                                 ORDER BY price
                                 LIMIT 1')->fetch();
print "$cheapest_dish_info[0], $cheapest_dish_info[1]";
```

При выполнении кода из примера 8.37 на экран выводится следующий результат:

```
Walnut Bun, 1
```

Урок языка SQL: предложения `ORDER BY` и `LIMIT`

Как упоминалось ранее в разделе "Организация информации в базе данных", строкам в таблице базы данных не присущ какой-то определенный порядок расположения. Сервер базы данных не обязан возвращать строки по запросу с командой **`SELECT`** в каком-то определенном порядке. Чтобы возвратить строки из таблицы в нужном порядке, команду **`SELECT`** следует дополнить предложением **`ORDER BY`**. Так, в примере 8.38 из таблицы **`dishes`** возвращаются все строки, упорядоченные по цене: от самой низкой до самой высокой.

Пример 8.38. Упорядочение строк, возвращаемых по запросу с командой `SELECT`

```
SELECT dish_name FROM dishes ORDER BY price
```

Чтобы упорядочить строки по значению: от наибольшего до наименьшего, после имени столбца, по которому происходит упорядочение, следует указать ключевое слово **`DESC`** (по убывающей). Так, в примере 8.39 из таблицы **`dishes`** возвращаются все строки, упорядоченные по цене: от самой высокой до самой низкой.

Пример 8.39. Упорядочение строк по цене: от самой высокой до самой низкой

```
SELECT dish_name FROM dishes ORDER BY price DESC
```

Для упорядочения можно указать несколько столбцов. Если же две строки содержат одинаковое значение в первом столбце, указанном в предложении **ORDER BY,** они упорядочиваются по второму столбцу. Так, строки, извлекаемые из таблицы **dishes** по запросу SQL из примера 8.40, упорядочиваются по цене: от самой высокой до самой низкой. Если же в нескольких строках окажется одинаковая цена, они упорядочиваются по имени в алфавитном порядке.

Пример 8.40. Упорядочение строк по нескольким столбцам

```
SELECT dish_name FROM dishes ORDER BY price DESC, dish_name
```

Употребление предложения **ORDER BY** в запросе SQL не изменяет порядок расположения строк в самой таблице (напомним, что они не располагаются в каком- то определенном порядке), но реорганизует результаты выполнения запроса. Это все равно, что дать кому-нибудь прочитать закуски из ресторанного меню в алфавитном порядке. Это окажет влияние не на само напечатанное меню, а на ответ по просьбе прочитать закуски в алфавитном порядке.

Как правило, по запросу SQL с командой **SELECT** возвращаются все строки, совпадающие с условием в предложении **WHERE,** или же все строки таблицы, если в запросе отсутствует предложение **WHERE.** Но иногда полезно получить в ответ определенный ряд строк, например, найти самое дешевое блюдо в таблице или вывести лишь десять результатов поиска по запросу. Чтобы ограничить результаты выполнения запроса конкретным количеством строк, в конце такого запроса следует добавить предложение **LIMIT.** Так, по запросу SQL в примере 8.41 из таблицы **dishes** возвращается строка с самым дешевым блюдом.

Пример 8.41. Ограничение количества строк, возвращаемых из таблицы по запросу SQL с командой SELECT

```
SELECT * FROM dishes ORDER BY price LIMIT 1
```

А по запросу SQL в примере 8.42 из таблицы **dishes** возвращаются первые десять строк, отсортированных по наименованиям блюд в алфавитном порядке.

Пример 8.42. Дополнительное количества строк, возвращаемых из таблицы по запросу SQL с командой SELECT

```
SELECT dish_name, price FROM dishes ORDER BY dish_name LIMIT 10
```

В общем, в запросах SQL следует употреблять лишь предложение **LIMIT,** у которого также имеется свое предложение **ORDER BY.** Если же опустить в запросе предложение **ORDER BY,** программа базы данных может возвратить строки в любом порядке. Поэтому строка, обнаруженная первой при выполнении запроса в первый раз, может таковой и не оказаться при выполнении того же самого запроса в следующий раз.

Изменение формата извлекаемых строк таблицы

В приведенных до сих пор примерах метод `fetch()` возвращал строки из базы данных в виде массивов, индексируемых как числами, так и символьными строками. С одной стороны, это упрощает вставку значений в двойных кавычках, а с другой — может усложнить дело. Например, трудно безошибочно запомнить, какому именно столбцу из запроса с командой `SELECT` соответствует шестой элемент результирующего массива. Имена некоторых столбцов, возможно, потребуется заключить в кавычки, чтобы вставить их надлежащим образом. А вынуждать интерпретатор PHP устанавливать числовые и строковые индексы было бы расточительно, если они не требуются вместе. Правда, расширение PDO позволяет указать предпочтительный способ доставки каждой результирующей строки в отдельности. Для этого в качестве аргумента методу `fetch()` или `fetchAll()` следует передать другой *стиль извлечения*, и тогда строка возвратится только в виде числового или

строкового массива или же в виде объекта.

Чтобы возвратить массив только с числовыми ключами, в качестве первого аргумента методу `fetch()` или `fetchAll()` следует передать атрибут **`PDO::FETCH_NUM`**, а для того чтобы возвратить массив только со строчными ключами — атрибут **`PDO::FETCH_ASSOC`**. Напомним, что массивы со строковыми ключами иногда еще называют *ассоциативными*.

Чтобы возвратить объект вместо массива, в качестве первого аргумента упомянутым выше методам следует передать атрибут **`PDO::FETCH_OBJ`**. Имена свойств объекта, возвращаемого для каждой строки, соответствуют именам столбцов. Применение разных стилей извлечения демонстрируется в примере 8.43.

Пример 8.43. Применение разных стилей извлечения

```
// При наличии только числовых индексов значения очень легко
// объединяются вместе
$q = $db->query('SELECT dlsh_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_NUM)) {
    print implode(', ', $row) . "\n";
}

// При наличии объекта для получения значений используется
// синтаксис доступа к свойствам этого объекта
$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch(PDO::FETCH_OBJ)) {
    print "{$row->dish_name} has price {$row->price} \n";
}
```

Если же стилем извлечения необходимо пользоваться неоднократно, его можно установить по умолчанию для конкретной команды во всех запросах, отправляемых при конкретном подключении к базе данных. Чтобы установить стиль извлечения по умолчанию, достаточно вызвать метод `setFetchMode()` для объекта типа `PDOStatement`, как показано в примере 8.44.

Пример 8.44. Установка стиля извлечения по умолчанию для команды в запросе SQL

```
$q = $db->query('SELECT dish_name, price FROM dishes');
// Теперь методу fetch() не нужно больше ничего передавать,
// т.к. обо всем позаботится метод setFetchMode()
$q->setFetchMode(PDO::FETCH_NUM);
while($row = $q->fetch()) {
    print implode(', ', $row) . "\n";
}
```

Чтобы установить стиль извлечения по умолчанию для всех запросов, достаточно вызвать метод `setAttribute()`, устанавливающий атрибут `PDO::ATTR_DEFAULT_FETCH_MODE` при подключении к базе данных:

```
// Вызывать метод setFetchMode() или передавать что-нибудь
// методу fetch() не нужно, т.к. обо всем позаботится
// метод setAttribute()
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_NUM);

$q = $db->query('SELECT dish_name, price FROM dishes');
while ($row = $q->fetch()) {
    print implode(', ', $row) . "\n";
```

```
}

$anotherQuery = $db->query(
        'SELECT dish_name FROM dishes WHERE price < 5');
// Каждый подчиненный массив в многомерном массиве $moreDishes
// также проиндексирован числами
$moreDishes = $anotherQuery->fetchAll();
```

Безопасное извлечение данных для формы

Замещающие знаки можно применять в команде SELECT таким же образом, как и в команде INSERT, UPDATE или DELETE. Вместо того чтобы вызывать метод query() непосредственно, можно вызвать методы prepare() и execute(), предоставив запрос с командой SELECT методу prepare().

Но если данные из переданной на обработку формы или входные данные из внешнего источника указываются в предложении WHERE команды SELECT, UPDATE или DELETE, то необходимо принять дополнительные меры для надлежащего экранирования любых подстановочных символов. Рассмотрим в качестве примера форму поиска с текстовым элементом dish_search, где пользователь может ввести наименование искомого блюда. При вызове метода execute() в коде из примера 8.45 используются замещающие знаки для защиты от смешивания одиночных кавычек в значении, передаваемом на обработку.

Пример 8.45. Применение замещающего знака в запросе SQL с командой SELECT

```
$stmt = $db->prepare('SELECT dish_name, price FROM dishes
                      WHERE dish_name LIKE ?');
$stmt->execute(array($_POST['dish_search']));
while ($row = $stmt->fetch()) {
    // ... сделать что-нибудь с переменной $row ...
}
```

Какое бы название блюда ни было введено в текстовом элементе dish_search, будь то **Fried Bean Curd** или **General Tso's Chicken**, замещающий знак позволяет надлежащим образом вставить введенное название блюда в запрос SQL. Но, что если в текстовом элементе dish_search введено название блюда **%chicken%**? В таком случае запрос SQL будет составлен следующим образом:

```
SELECT dish_name, price FROM dishes WHERE dish_name LIKE '%chicken%'
```

Этому запросу соответствуют все строки таблицы, содержащие символьную строку "chicken", а не только те строки, где значение в столбце dish_name точно равно **%chicken%**.

Урок языка SQL: подстановочные символы

Подстановочные символы удобны для неточного сопоставления текста, например, поиска символьных строк, оканчивающихся на **.edu** или содержащих знак @. В языке имеются две разновидности подстановочных символов: знак подчеркивания (_) для сопоставления с одним символом и знак процента (%) для сопоставления с любым количеством символов (от нуля и больше). В символьных строках подстановочные символы применяются вместе с оператором **LIKE** в предложении **WHERE.** В примере 8.46 приведены два запроса SQL с командой **SELECT,** где применяются подстановочные символы и оператор **LIKE.**

Пример 8.46. Применение подстановочных символов в запросах SQL с командой SELECT

```
; извлечь из таблицы все строки с наименованиями блюд,
; начинающихся с буквы D
SELECT * FROM dishes WHERE dish_name LIKE 'D%'

; извлечь из таблицы все строки с наименованиями блюд
; Fried Cod, Fried Bod, Fried Nod и т.д.
SELECT * FROM dishes WHERE dish_name LIKE 'Fried _od'
```

Подстановочные символы действуют и в предложениях **WHERE**, применяемых в командах **UPDATE** и **DELETE** языка SQL. Так, в запросе SQL из примера 8.47 дублируются цены на все блюда со словом **chili** (красный стручковый перец) в их наименовании.

Пример 8.47. Применение подстановочных символов в запросе SQL с командой UPDATE

```
UPDATE dishes SET price = price * 2 WHERE dish_name LIKE '%chili%'
```

В запросе SQL, приведенном в примере 8.48, из таблицы удаляются все строки, где наименования блюд в столбце **dish_name** оканчиваются на **Shrimp** (Креветка).

Пример 8.48. Применение подстановочных символов в запросе SQL с командой DELETE

```
DELETE FROM dishes WHERE dish_name LIKE '%Shrimp'
```

Для сопоставления с самими знаками **%** и **_** в операторе **LIKE** их следует предварить знаком обратной косой черты. Так, по запросу SQL, приведенному в примере 8.49, обнаруживаются все строки таблицы, где наименования блюд в столбце **dish_name** содержат фразу **50% off**.

Пример 8.49. Экранирование подстановочных символов

```
SELECT * FROM dishes WHERE dish_name LIKE '%50\% off%'
```

В отсутствие обратной косой черты по запросу SQL из примера 8.49 были бы сопоставлены строки таблицы, где наименования блюд в столбце **dish_name** содержат число **50**, пробел и слово **off** в любом сочетании, как, например, **Spicy 50 shrimp with shells off salad** (Салат из 50 креветок без раковин со специями) или **Famous 500 offer duck** (Утка, знаменитая ассортиментом из 500 разных блюд).

Чтобы предотвратить действие подстановочных символов в запросах SQL, придется отказаться от удобства и простоты замещающих знаков и положиться на следующие две функции: метод `quote()` из расширения PDO и функцию `strtr()`, встроенную в PHP. Сначала следует вызвать метод `quote()` для переданного на обработку значения.

Этот метод выполняет такую же операцию заключения в кавычки, как и замещающий знак. Например, данный метод преобразует строковое значение **General Tso's Chicken** в эквивалентное значение **'General Tso"s Chicken'**. Затем вызывается функция `strtr()`, чтобы экранировать знаком обратной косой черты подстановочные символы **%** и **_** в запросе SQL. Таким образом, заключенное в кавычки и экранированное строковое значение может быть безопасно использовано в запросе. В примере 8.50 показано, как с помощью метода `quote()` и функции `strtr()` переданное на обработку значение делается безопасным для применения в предложении `WHERE`.

Пример 8.50. Составление запроса SQL с командой SELECT без применения замещающих знаков

```
// Сначала заключишь переданное на обработку значение в кавычки
$dish = $db->quote($_POST['dish_search']);
// Затем экранировать знаками обратной косой черты знаки
// подчеркивания и процента
```

```
$dish = strtr($dish, array=>('_' => '\_', '%' => '\%'));
// После санобработки значение переменной $dish может
// быть вставлено в запрос SQL
$stmt = $db->query("SELECT dish_name, price FROM dishes
                    WHERE dish_name LIKE $dish");
```

В данном случае воспользоваться замещающим знаком нельзя, потому что экранирование подстановочных знаков SQL выполняется после обычного заключения в кавычки, при котором знак обратной косой черты ставится не только перед кавычками, но и перед самими знаками обратной косой черты. Так, если обработать переданное строковое значение **%chicken%** сначала с помощью функции `strtr()`, оно примет вид `\%chicken\%`. А после заключения в кавычки с помощью метода `quote()` или замещающего знака оно примет вид `'\\%chicken\\%'`. Такое строковое значение будет интерпретировано программой базы данных как знак обратной косой черты, после которого следует подстановочный символ, обозначающий сопоставление с любыми символами, затем слово **chicken**, далее еще один знак обратной косой черты и, наконец, еще один подстановочный символ для сопоставления с любыми символами. Но если вызвать сначала метод `quote()`, то переданное на обработку строковое значение **%chicken%** будет преобразовано в значение **'%chicken%'**. А в функции `strtr()` оно будет затем преобразовано в окончательный вид `'\%chicken\%'`. Такое строковое значение будет интерпретировано программой базы данных как знак процентов, после которого следует слово **chicken** и еще один знак процентов, т.е. именно в том виде, в каком это значение было введено пользователем.

Если не заключить подстановочные символы в кавычки, это будет иметь еще более серьезные последствия в предложении `UPDATE` или `DELETE`, применяемом в команде `UPDATE` или `DELETE`. В примере 8.51 демонстрируется неверное применение знаков замещения в запросе SQL с целью установить цену 1 доллар на блюда, определяемые значением, введенным пользователем.

Пример 8.51. Неверное применение знаков замещения в запросе SQL с командой `UPDATE`

```
$stmt = $db->prepare('UPDATE dishes SET price = 1 WHERE dish_name LIKE ?');
$stmt->execute(array($_POST['dish_name']));
```

Если для столбца `dish_name` таблицы передано наименование блюда **Fried Bean Curd**, то запрос SQL из примера 8.51 будет обработан как обычно: цена 1 доллар будет установлена только на указанное блюдо. Но если в значении элемента массива `$_POST['dish_name']` окажется знак **%**, то цена 1 доллар будет установлена на все блюда в таблице! Этот недостаток можно устранить с помощью метода `quote()` и функции `strtr()`. В примере 8.52 демонстрируется правильный способ обновления цены на указанное блюдо.

Пример 8.52. Правильное применение метода `quote()` и функции `strtr()` в команде `UPDATE`

```
// Сначала заключить переданное на обработку значение в кавычки
$dish = $db->quote($_POST['dish_name']);
// Затем предварить знаки подчеркивания и процентов знаками
// обратной косой черты
$dish = strtr($dish, array('_' => '\_', '%' => '\%'));
// После санобработки значение переменной $dish может
// быть вставлено в запрос SQL
$db->exec("UPDATE dishes SET price = 1 WHERE dish_name LIKE $dish");
```

Законченная форма для извлечения записей из базы данных

В примере 8.53 демонстрируется законченная программа для составления формы и извлечения записей из базы данных. Эта программа предоставляет форму поиска и выводит на экран HTML-таблицу, состоящую из всех строк в таблице `dishes` базы данных, удовлетворяющих заданным

критериям поиска. Как и программа из примера 8.28, данная программа основана на вспомогательном классе для составления форм, определенном в отдельном файле `FormHelper.php`.

Пример 8.53. Программа для поиска записей в таблице `dishes`

```
<?php

// загрузить вспомогательный класс для составления форм

require 'FormHelper.php';

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Can't connect: " . $e->getMessage();
    exit();
}
// установить исключения при ошибках в базе данных
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// установить режим извлечения строк таблицы в виде объектов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);

// задать варианты выбора из списка формы, определяющие
// наличие специй в блюде
$spicy_choices = array('no','yes','either');

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    // установить свои значения по умолчанию
    $defaults = array('min_price' => '5.00',
                      'max_price' => '25.00');

    // Создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper($defaults);
```

```
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'retrieve-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();

    // удалить любые начальные и конечные пробелы из переданного
    // на обработку наименования блюда
    $input['dish_name'] = trim($_POST['dish_name'] ?? '');

    // Минимальная цена на блюдо должна быть
    // достоверным числом с плавающей точкой
    $input['min_price'] = filter_input(INPUT_POST,'min_price',
                                       FILTER_VALIDATE_FLOAT);
    if ($input['min_price'] === null ||
                                $input['min_price'] === false) {
        $errors[] = 'Please enter a valid minimum price.';
    }

    // Максимальная цена на блюдо должна быть
    // достоверным числом с плавающей точкой
    $input['max_price'] = filter_input(INPUT_POST,'max_price',
                                       FILTER_VALIDATE_FLOAT);
    if ($input['max_price'] === null ||
                                $input['max_price'] === false) {
        $errors[] = 'Please enter a valid maximum price.';
    }

    // Минимальная цена на блюдо должна быть меньше
    // максимальной цены
    if ($input['min_price'] >= $input['max_price']) {
        $errors[] =
          'The minimum price must be less than the maximum price.';
    }

    $input['is_spicy'] = $_POST['is_spicy'] ?? '';
    if(! array_key_exists($input['is_spicy'],
                          $GLOBALS['spicy_choices'])) {
        $errors[] = 'Please choose a valid "spicy" option.';
    }
    return array($errors, $input);
}

function process_form($input) {
    // получить доступ к глобальной переменной $db
    // в теле данной функции
    global $db;

    // составить запрос к базе данных
```

```
    $sql = 'SELECT dish_name, price, is_spicy FROM dishes
                  WHERE price >= ? AND price <= ?';

    // Если наименование блюда передано, ввести его в
    // предложение WHERE. С помощью метода quote() и
    // функции strtr() предотвращается действие вводимых
    // пользователем подстановочных символов
    if (strlen($input['dish_name'])) {
        $dish = $db->quote($input['dish_name']);
        $dish = strtr($dish, array('_' => '\_', '%' => '\%'));
        $sql .= " AND dish_name LIKE $dish";
    }

    // Если в элементе ввода is_spicy установлено значение
    // 'yes' или 'no', ввести в запрос SQL соответствующее
    // логическое условие. (Если же установлено значение "either",
    // вводить логическое условие в предложение WHERE не нужно.)
    $spicy_choice = $GLOBALS['spicy_choices'][ $input['is_spicy'] ];
    if ($spicy_choice == 'yes') {
        $sql .= ' AND is_spicy = 1';
    } elseif ($spicy_choice == 'no') {
        $sql .= ' AND is_spicy = 0';
    }

    // отправить запрос программе базы данных и получить
    // в ответ все нужные строки из таблицы
    $stmt = $db->prepare($sql);
    $stmt->execute(array($input['min_price'], $input['max_price']));
    $dishes = $stmt->fetchAll();
    if (count($dishes) == 0) {
        print 'No dishes matched.';
    } else {
        print '<table>';
        print
          '<tr><th>Dish Name</th><th>Price</th><th>Spicy?</th></tr>';
        foreach ($dishes as $dish) {
           if ($dish->is_spicy == 1) {
               $spicy = 'Yes';
           } else {
               $spicy = 'No';
           }
           printf('<tr><td>%s</td><td>$%.02f</td><td>%s</td></tr>',
                  htmlentities($dish->dish_name),
                  $dish->price, $spicy);
        }
    }
}
?>
```

Исходный код из примера 8.53 во многом похож на исходный код из примера 8.28. В нем применяется стандартная структура для отображения, проверки достоверности и обработки формы с глобальным кодом для подключения и взаимодействия с базой данных в функции `process_form()`. А в функции `show_form()` отображается HTML-разметка формы, определенная в файле `retrieve-`

`form.php`. Содержимое этого файла приведено в примере 8.54.

Пример 8.54. Форма для извлечения информации о блюдах из базы данных

```
<form method="POST" action="<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
       </ul></td>
  <?php } ?>

  <tr>
    <td>Dish Name:</td>
    <td><?= $form->input('text', ['name' => 'dish_name']) ?></td>
  </tr>

  <tr>
    <td>Minimum Price:</td>
    <td><?= $form->input ('text',['name' => 'min_price']) ?></td>
  </tr>

  <tr>
    <td>Maximum Price:</td>
    <td><?= $form->input('text',['name' => 'max_price']) ?></td>
  </tr>

  <tr>
    <td>Spicy:</td>
    <td><?= $form->select($GLOBALS['spicy_choices'],
                          ['name' => 'is_spicy']) ?>
    </td>
  </tr>

  <tr>
    <td colspan="2" align="center">
        <?= $form->input('submit', ['name' => 'search',
                         'value' => 'Search']) ?></td>
  </tr>
</table>
</form>
```

Исходный код из примера 8.53 отличается дополнительной строкой при подключении к базе данных, где вызывается функция `setAttribute()`, устанавливающая режим извлечения записей. Этот режим очень важен для извлечения информации из базы данных в функции `process_form()`.

В функции `process_form()` составляется запрос SQL с командой `SELECT`, который затем отправляется в базу данных с помощью метода `execute()`, а полученные результаты извлекаются с помощью `fetchAll()` и выводятся на экран в виде HTML-таблицы. В предложении `WHERE`, применяемом в команде `SELECT`, указывается до четырех логических условий. Два первых условия

определяют минимальную и максимальную цену на блюдо. А поскольку они всегда присутствуют в запросе, то получают замещающие знаки в переменной `$sql`, где хранится запрос SQL.

Далее следует наименование блюда. И хотя это необязательно, но если оно все же передается на обработку, то включается в запрос. Тем не менее замещающий знак не вполне подходит для столбца `dish_name`, поскольку данные из переданной формы могут содержать подстановочные символы SQL. Вместо этого с помощью метода `quote()` и функции `strtr()` подготавливается прошедший "санобработку" вариант наименования блюда, который затем вводится непосредственно в предложение `WHERE`.

И последним в предложении `WHERE` указывается столбец `is_spicy`. Если на обработку передан вариант выбора `yes`, то в запрос SQL вводится логическое условие `AND is_spicy = 1`, чтобы из таблицы базы данных были извлечены только блюда со специями. Если же на обработку передан вариант выбора `no`, то в запрос SQL вводится логическое условие `AND is_spicy = 0`, чтобы в таблице базы данных были найдены только блюда без специй. А если на обработку передан вариант выбора `either`, то указывать столбец `is_spicy` в запросе не нужно, поскольку строки должны быть извлечены из таблицы независимо от того, приправлены ли они специями.

Как только запрос SQL будет полностью составлен в переменной `$sql`, он подготавливается с помощью функции prepare() и направляется программе базы данных с помощью метода execute(). В качестве второго аргумента метода execute() указывается массив, содержащий значения минимальной и максимальной цены, чтобы подставить их вместо замещающих знаков. Массив строк таблицы, возвращаемый функцией fetchAll(), сохраняется в переменной `$dishes`.

И, наконец, в функции `process_form()` выводятся результаты обработки запроса SQL. Если данные в переменной `$dishes` отсутствуют, то на экран выводится сообщение `"No dishes matched"` (Совпавшие блюда отсутствуют). В противном случае содержимое переменной `$dishes` выводится в цикле `foreach()` в виде строк HTML-таблицы с каждым блюдом в отдельности, причем для надлежащего форматирования цены на блюдо вызывается функция `printf()`, а для кодирования любых специальных символов в наименовании блюда — функция `htmlentities()`. В условном операторе `if()` удобные для базы данных значения **`1`** и **`0`** в столбце `is_spicy` преобразуются в более удобные для восприятия пользователем значения **`Yes`** и **`No`** соответственно.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Выяснение видов информации, которые можно хранить в базе данных.
- Представления об организации информации в базе данных.
- Установление соединения с базой данных.
- Создание таблицы в базе данных.
- Удаление таблицы из базы данных.
- Применение команды `INSERT` в запросах SQL.
- Ввод информации в базу данных с помощью метода `exec()`.
- Проверка ошибок при выполнении операций в базе данных путем обработки генерируемых исключений.
- Изменение режима выдачи ошибок с помощью метода `setAttribute()`.
- Применение команд `UPDATE` и `DELETE` в запросах SQL.
- Изменение или удаление информации в базе данных с помощью метода `exec()`.

- Подсчет количества строк таблицы, затрагиваемых запросом SQL.
- Применение замещающих знаков для безопасного ввода информации в запрос SQL.
- Применение команды `SELECT` в запросах SQL.
- Извлечение данных из базы данных с помощью методов `query()` и `fetch()`.
- Подсчет количества строк, извлеченных из таблицы, с помощью метода `query()`.
- Применение ключевых слов `ORDER BY` и `LIMIT` в запросах SQL.
- Извлечение строк в виде массивов, индексированных символьными строками, или в виде объектов.
- Применение подстановочных символов **`%`** и **`_`** вместе с оператором `LIKE` в запросах SQL.
- Экранирование подстановочных символов в командах `SELECT` запросов SQL.
- Сохранение параметров переданной на обработку формы в базе данных.
- Применение информации, извлекаемой из базы данных, в заполняемой форме.

Упражнения

В последующих упражнениях применяется таблица `dishes` базы данных, имеющая следующую структуру:

```
CREATE TABLE dishes (
    dish_id    INT,
    dish_name  VARCHAR(255),
    price      DECIMAL(4,2),
    is_spicy   INT
)
```

Ниже приведен образец данных, размещаемых в таблице `dishes`.

```
INSERT INTO dishes VALUES (1,'Walnut Bun',1.00,0)
INSERT INTO dishes VALUES (2,'Cashew Nuts and White Mushrooms',4.95,0)
INSERT INTO dishes VALUES (3,'Dried Mulberries',3.00,0)
INSERT INTO dishes VALUES (4,'Eggplant with Chili Sauce',6.50,1)
INSERT INTO dishes VALUES (5,'Red Bean Bun',1.00,0)
INSERT INTO dishes VALUES (6,'General Tso''s Chicken',5.50,1)
```

1. Напишите программу, в которой все блюда, находящиеся в таблице `dishes`, перечисляются отсортированными по цене.
2. Напишите программу, отображающую форму с запросом блюд по их цене. После передачи формы на обработку программа должна вывести наименования и цены тех блюд, которые стоят не меньше, чем указано в форме. Не извлекайте из таблицы базы данных строки или столбцы, которые не подлежат выводу на экран.
3. Напишите программу, отображающую форму со списком наименований блюд, размечаемым дескриптором `<select>`. Составьте такой список из наименований блюд, извлеченных из базы данных. После передачи формы на обработку программа должна вывести из таблицы всю информацию о выбранном блюде, в том числе идентификатор, наименование, цену и наличие специй в данном блюде.

4. Создайте новую таблицу для хранения информации о посетителе ресторана. В этой таблице должна храниться следующая информация о каждом посетителе ресторана: идентификатор посетителя, имя, номер телефона, а также идентификатор излюбленного блюда посетителя. Напишите программу, отображающую форму для ввода нового посетителя в таблицу. Часть формы, предназначенная для ввода излюбленного блюда посетителя, должна быть реализована в виде списка наименований блюд, размечаемого дескриптором `<select>`. Идентификатор посетителя должен формироваться вашей программой, а не вводиться в заполняемой форме.

ГЛАВА 9

Манипулирование файлами

Избранным местом для хранения информации в веб-приложениях служит база данных. Но это совсем не означает, что можно полностью отказаться от обычных файлов. Простые текстовые файлы по-прежнему остаются удобным и универсальным средством для обмена некоторыми видами информации.

Храня шаблоны HTML-разметки в текстовых файлах, можно заметно упростить специальную настройку веб-сайта. А когда настанет время для формирования специализированной страницы, достаточно загрузить соответствующий текстовый файл, подставить настоящие данные в элемент шаблона и вывести страницу на экран. В примере 9.2 показано, как это делается.

Файлы удобны также для обмена табличными данными между прикладной программой и электронной таблицей. В программах на PHP можно легко организовать чтение и запись информации в файлы формата CSV (значения, разделяемые запятыми), с которыми совместимы программы электронных таблиц.

В этой главе демонстрируется порядок манипулирования файлами в программах на PHP. В ней, в частности, рассматриваются *полномочия доступа к файлам*, применяемые на компьютере для соблюдения правил, разрешающих обращаться к файлам из прикладных программ; чтение и запись информации в файлы, а также обработка ошибок, которые могут возникнуть при выполнении операций с файлами.

Представление о полномочиях доступа к файлам

Чтобы прочитать или записать информацию в файл, используя любую из функций, рассматриваемых в этой главе, интерпретатор PHP должен получить соответствующие полномочия от операционной системы. Всякая программа, выполняющаяся на компьютере, включая и интерпретатор PHP, действует с привилегиями, определяемыми учетной записью пользователя. Большинство учетных записей назначаются для конкретных пользователей. Когда пользователь входит в систему и запускает на выполнение текстовый редактор, последний действует с привилегиями, определяемыми учетной записью данного пользователя. Такие привилегии могут разрешать чтение файлов для просмотра их содержимого, а также запись в файлы для изменения их содержимого.

Но некоторые учетные записи на компьютере рассчитаны не на пользователей, а на такие системные процессы, как веб-серверы. Когда интерпретатор PHP выполняется на веб-сервере, он получает те привилегии, которые предоставляет учетная запись веб-сервера. Так, если веб-серверу разрешается прочитать содержимое какого-нибудь файла или каталога, интерпретатор PHP, а следовательно, и программа на PHP может прочитать этот файл или каталог. А если веб-серверу разрешается изменить содержимое какого-нибудь файла или записывать информацию в новые файлы, создаваемые

в конкретном каталоге, это может также сделать интерпретатор PHP и программа, написанная на PHP.

Как правило, привилегии, распространяемые на учетную запись веб-сервера, более ограниченные, чем привилегии, которыми наделяется учетная запись настоящего пользователя. Веб-сервер (и интерпретатор PHP) должен быть в состоянии читать все исходные файлы программ на PHP, составляющие веб-сайт, но он не должен их изменять. На тот случай, если программная ошибка на веб-сервере или ненадежная программа на PHP позволит атакующему злоумышленнику проникнуть на веб-сайт, исходные файлы программ на PHP должны быть защищены от возможного изменения этим злоумышленником.

На практике это означает, что программы на PHP не должны испытывать особых затруднений при чтении большинства файлов, необходимых для нормальной работы веб-сайта. (Безусловно, всякая попытка прочитать личные файлы другого пользователя должна быть в корне пресечена!) Но доступ к файлам, которые разрешается изменять в программе на PHP, а также к каталогам, в которые она может записывать новые файлы, должен быть ограничен. Так, если в программах на PHP требуется создать немало новых файлов, следует обратиться за помощью к системному администратору, чтобы создать специальный каталог, в который можно записывать файлы, не нарушая безопасность системы. О том, как определять файлы и каталоги, которые разрешается читать и записывать в программе на PHP, речь пойдет далее, в разделе “Проверка полномочий доступа к файлам”.

Чтение и запись всего содержимого файлов

В этом разделе поясняется, как обрабатывать содержимое всего файла в целом, а не отдельных хранящихся в нем строк. Для чтения и записи сразу всего файла в PHP предоставляются специальные функции.

Чтение из файла

Чтобы прочитать содержимое файла в символьную строку, следует вызвать функцию `file_get_contents()`, передав ей имя файла. Эта функция возвратит символьную строку, содержащую все, что находится в данном файле. Так, в примере 9.2 содержимое файла из примера 9.1 сначала читается с помощью функции `file_get_contents()` и затем видоизменяется с помощью функции `str_replace()`, а полученный результат выводится на экран.

Пример 9.1. Содержимое файла `page-template.html`, используемого в примере 9.2

```
<html>
<head><title>{page_title}</title></head>
<body bgcolor="{color}">

<h1>Hello, {name}</h1>
</body>
</html>
```

Пример 9.2. Чтение, видоизменение и вывод шаблона страницы на экран

```
// загрузить файл шаблона из предыдущего примера
$page = file_get_contents('page-template.html');

// ввести заглавие страницы
$page = str_replace('{page_title}', 'Welcome', $page);
```

```
// окрасить страницу голубым цветом после полудня и
// зеленым цветом с утра
if (date('H' >= 12)) {
$page = str_replace('{color}', 'blue', $page);
} else {
$page = str_replace('{color}', 'green', $page);
}

// взять имя пользователя из переменной сохраненного
// предыдущего сеанса
$page = str_replace('{name}', $_SESSION['username'], $page);

// вывести полученные результаты на экран
print $page;
```

Всякий раз, когда вы пользуетесь функцией доступа к файлу, непременно проверяйте, не возникла ли при ее выполнении ошибка в связи с исчерпанием места на диске, нарушением прав доступа к файлу или по какой-нибудь другой причине. Более подробно проверка на возникновение ошибок рассматривается в разделе "Выявление ошибок" главы 9. В примерах, приведенных в ряде последующих разделов, отсутствует код проверки ошибок, что позволяет продемонстрировать действие функции доступа к файлу, не отвлекаясь на другой новый материал. А в реальных программах следует организовать проверку на возникновение ошибок после вызова функции доступа к файлу.

Если в элементе массива `$_SESSION['username']` установлено имя пользователя `Jacob`, то при выполнении кода из примера 9.2 на экран выводится следующий результат:

```
<html>
<head><title>Welcome</title></head>
<body bgcolor="green">

<h1>Hello, Jacob</h1>

</body>
</html>
```

Запись в файл

Запись символьной строки в файл является противоположной операции чтения содержимого файла в символьную строку. И для этой цели служит функция `file_put_contents()`, противоположная по своему действию функции `file_get_contents()`. Пример 9.3 расширяет пример 9.2 сохранением HTML-разметки в файле вместо ее вывода на экран.

Пример 9.3. Запись в файл с помощью функции `file_put_contents()`

```
// загрузить файл шаблона из предыдущего примера
$page = file_get_contents('page-template.html');

// ввести заглавие страницы
$page = str_replace('{page_title}', 'Welcome', $page);
```

```
// окрасить страницу голубым цветом после полудня и
// зеленым цветом с утра
if (date('H' >= 12)) {
    $page = str_replace('{color}', 'blue', $page);
} else {
    $page = str_replace('{color}', 'green', $page);
}

// взять имя пользователя из переменной сохраненного
// предыдущего сеанса
$page = str_replace('{name}', $_SESSION['username'], $page);

// записать полученные результаты в файл page.html
file_put_contents('page.html', $page);
```

В коде из примера 9.3 содержимое переменной `$page` (т.е. HTML-разметка страницы) записывается в файл `page.html`. В качестве первого аргумента функции `file_put_contents()` передается имя файла, в который производится запись, а в качестве второго — записываемое содержимое.

Частичное чтение и запись файлов

Функции `file_get_contents()` и `file_put_contents()` удобны для манипулирования сразу всем файлом. Но иногда требуется манипулировать лишь частью файла, и для этой цели служит функция `file()`, которая осуществляет доступ к отдельным строкам в файле. Так, в примере 9.4 демонстрируется чтение из файла, каждая строка которого содержит имя и адрес электронной почты, а затем эта информация выводится на экран списком в формате HTML.

Пример 9.4. Доступ к каждой строке в файле

```
foreach (file('people.txt') as $line) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto:' . $info[0] . '">'
                        . $info[1] ."</li>\n";
}
```

Допустим, что в файле `people.txt` содержится текст, приведенный в примере 9.5.

Пример 9.5. Содержимое файла `people.txt`, применяемого в примере 9.4

```
alice@example.com|Alice Liddell
bandersnatch@example.org|Bandersnatch Gardner
charles@milk.example.com|Charlie Tenniel
dodgson@turtle.example.com|Lewis Humbert
```

В таком случае при выполнении кода из примера 9.4 на экран выводится следующий результат:

```
<li><a href="mailto:alice@example.com">Alice Liddell</li>
<li><a href="mailto:bandersnatch@example.org">
    Bandersnatch Gardner</li>
<li><a href="mailto:charles@milk.example.com">
    Charlie Tenniel</li>
<li><a href="mailto:dodgson@turtle.example.com">
    Lewis Humbert</li>
```

Функция `file()` возвращает массив. Каждый элемент этого массива представляет собой символьную строку, содержащую одну строку из файла, включая знак перевода строки. Поэтому в цикле `foreach()` из примера 9.4 осуществляется перебор каждого элемента данного массива с сохранением его символьной строки в переменной `$line`. В частности, функция `trim()` удаляет конечный знак перевода строки, функция `explode()` разделяет строку на две части до и после знака `|`, а оператор `print` выводит элементы списка в формате HTML.

Несмотря на все удобства функции `file()`, она не совсем подходит для манипулирования крупными файлами. Ведь эта функция читает все содержимое файла для построения массива из прочитанных строк, и если файл состоит из большого количества строк, то для их чтения может потребоваться слишком много оперативной памяти. В таком случае содержимое крупного файла придется читать построчно, как показано в примере 9.6.

Пример 9.6. Чтение файла построчно

```
$fh = fopen('people.txt','rb');
while ((! feof($fh)) && ($line = fgets($fh))) {
    $line = trim($line);
    $info = explode('|', $line);
    print '<li><a href="mailto: ' . $info[0] . '">'
                        . $info[1] ."</li>\n";
}
fclose($fh);
```

В примере 9.6 применяются четыре функции доступа к файлу: `fopen()`, `fgets()`, `feof()` и `fclose()`. Совместно они действуют следующим образом.

- Функция `fopen()` открывает файл и возвращает переменную, используемую для последующего доступа к файлу в программе. (Принципиально это делается аналогично подключению к базе данных с помощью операции `new PDO()`, возвращающей объект, сохраняемый в переменной, как пояснялось в главе 8.)
- Функция `fgets()` читает строку из файла и возвращает ее в виде символьной строки.
- Интерпретатор PHP хранит закладку на текущую позицию в файле. Сначала эта закладка делается на начало файла, и поэтому при первом вызове функции `fgets()` из файла читается первая строка. После того как эта строка будет прочитана, закладка обновится, установившись на начале следующей строки.
- Функция `feof()` возвращает логическое значение `true`, если закладка указывает на позицию после конца файла (признак `"eof"` означает конец файла).
- Функция `fclose()` закрывает файл.

Цикл `while()` из примера 9.6 продолжает выполняться до тех пор, пока не произойдет одно из двух:

- функция `feof()` возвратит логическое значение `false`;
- переменной `$line` будет присвоено значение, возвращаемое в результате вызова функции `fgets($fh)` и вычисляемое как истинное (`true`).

Всякий раз, когда вызывается функция `fgets($fh)`, интерпретатор PHP извлекает строку из файла, продвигает свою закладку, иначе называемую дескриптором файла, и возвращает эту строку. Если закладка указывает на самую последнюю позицию в файле, функция `feof($fh)` по-прежнему

возвращает логическое значение `false`. Но в то же время функция `fgets($fh)` возвращает логическое значение `false`, поскольку она пытается прочитать строку и не может этого сделать. Таким образом, обе проверки этих возвращаемых значений необходимы для правильного завершения цикла.

В примере 9.6 применяется функция `trim()` для обработки значения переменной `$line`, поскольку символьная строка, возвращаемая функцией `fgets()`, содержит конечный знак перевода строки. Функция `trim()` удаляет этот знак, чтобы усовершенствовать выводимый результат.

В качестве первого аргумента функции `fopen()` передается имя файла, к которому требуется получить доступ. Как и при вызове остальных функций доступа к файлу, предоставляемых в PHP, в данном случае следует использовать знак косой черты (/) вместо знака обратной косой черты (\) — даже в Windows. В примере 9.7 демонстрируется, как открывается файл из каталога в ОС Windows.

Пример 9.7. Открытие файла в Windows

```
$fh = fopen('c:/windows/system32/settings.txt','rb');
```

Знаки обратной косой черты имеют особое назначение, экранируя специальные символы в строках, как пояснялось в разделе “Определение символьных строк текста” в главе 2. Поэтому в путях к файлам проще пользоваться знаками косой черты. Они правильно распознаются интерпретатором PHP при загрузке файлов в Windows.

В качестве второго аргумента функции `fopen()` передается *режим доступа к файлу*, определяющий операции, которые разрешается выполнять над файлом: чтение, запись и т.д. Кроме того, режим доступа к файлу определяет начальную позицию в файле, на которой интерпретатор PHP делает закладку, порядок очистки содержимого файла при его открытии и способ реагирования интерпретатора PHP на отсутствие файла. В табл. 9.1 перечислены различные режимы доступа к файлу, которые распознает функция `fopen()`.

Таблица 9.1. Режимы доступа к файлу, пригодные для функции `fopen()`

Режим	Допустимые действия	Начальная позиция закладки	Очистка содержимого	Что делать, если файл отсутствует
rb	Чтение	Начало файла	Нет	Выдать предупреждение, возвратить логическое значение **false**
rb+	Чтение, запись	Начало файла	Нет	Выдать предупреждение, возвратить логическое значение **false**
wb	Запись	Начало файла	Да	Попытаться создать файл
wb+	Чтение, запись	Начало файла	Да	Попытаться создать файл
ab	Запись	Конец файла	Нет	Попытаться создать файл
ab+	Чтение, запись	Конец файла	Нет	Попытаться создать файл
xb	Запись	Начало файла	Нет	Попытаться создать файл. Если файл существует, выдать предупреждение и возвратить логическое значение **false**
xb+	Чтение, запись	Начало файла	Нет	Попытаться создать файл. Если файл существует, выдать предупреждение и возвратить логическое значение **false**
cb	Запись	Начало файла	Нет	Попытаться создать файл
cb+	Чтение, запись	Начало файла	Нет	Попытаться создать файл

Открыв файл в режиме доступа, разрешающем запись, можно воспользоваться функцией `fwrite()`, чтобы записать какие-нибудь данные в файл. Так, в примере 9.8 используется режим доступа **`wb`** вместе с функциями `fopen()` и `fwrite()` для записи информации, извлекаемой из таблицы базы данных, в текстовый файл `dishes.txt`.

Пример 9.8. Запись данных в файл

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    print "Couldn't connect to database: " . $e->getMessage();
    exit();
}

// открыть файл dishes.txt для записи
$fh = fopen('dishes.txt','wb');
$q = $db->query("SELECT dish_name, price FROM dishes");
while($row = $q->fetch()) {
    // записать каждую строку (с завершающим знаком перевода
    // строки) в файл dishes.txt
    fwrite($fh, "The price of $row[0] is $row[1] \n");
}
fclose($fh);
```

Функция `fwrite()` не добавляет автоматически знак перевода строки в конце записываемой в файл символьной строки, а просто записывает именно то, что ей передается в качестве второго аргумента. Если же требуется дополнить знаком перевода строки каждую записываемую в файл строку, как это делается в примере 9.8, в конце символьной строки, передаваемой функции `fwrite()`, следует добавить знаки `\n`, обозначающие переход на новую строку.

Манипулирование файлами формата CSV

Особой интерпретации в PHP подлежат текстовые файлы формата CSV. Такие файлы не могут содержать графики или диаграммы, но в то же время они позволяют обмениваться табличными данными между разными программами. Чтобы прочитать строку из файла формата CSV, следует вызвать функцию `fgetcsv()` вместо `fgets()`. Эта функция читает строку из файла формата CSV и возвращает массив, содержащий каждое поле в данной строке. Так, в примере 9.9 приведен файл формата CSV, содержащий ресторанное меню, а в пример 9.10 демонстрируется применение функции `fgetcsv()` для чтения файла и ввода полученной информации в таблицу базы данных, упоминавшуюся в главе 8.

Пример 9.9. Содержимое файла `dishes.csv`

```
"Fish Ball with Vegetables",4.25,0
"Spicy Salt Baked Prawns",5.50,1
"Steamed Rock Cod",11.95,0
"Sauteed String Beans",3.15,1
"Confucius ""Chicken""",4.75,0
```

Пример 9.10. Ввод данных формата CSV в таблицу базы данных

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    print "Couldn't connect to database: " . $e->getMessage();
    exit();
}
$fh = fopen('dishes.csv','rb');
$stmt = $db->prepare('INSERT INTO dishes (dish_name,
                                          price, is_spicy)
                                  VALUES (?,?,?)');
while ((! feof($fh)) && ($info = fgetcsv($fh))) {
    // Элемент массива $info[0] содержит наименование блюда
    // из первого поля в строке, считанной из файла dishes.csv.
    // Элемент массива $info[1] содержит цену на блюдо
    // из второго поля в считанной строке.
    // Элемент массива $info[2] содержит состояние, обозначающее
    // наличие специй в блюде, из третьего поля в считанной строке.
    // Ввести упомянутое содержимое массива $info отдельной строкой
    // в таблицу базы данных
    $stmt->execute($info);
    print "Inserted $info[0]\n";
}
// закрыть файл
fclose($fh);
```

При выполнении кода из примера 9.10 на экран выводится следующий результат:

```
Inserted Fish Ball with Vegetables
Inserted Spicy Salt Baked Prawns
Inserted Steamed Rock Cod
Inserted Sauteed String Beans
Inserted Confucius "Chicken"
```

Запись строки в файл формата CSV выполняется аналогично чтению из строки из такого файла. С этой целью вызывается функция `fputcsv()`, принимающая в качестве аргументов дескриптор файла и массив значений, заранее подготовленных в формате CSV для записи в файл. Так, в примере 9.11 демонстрируется применение функций `fputcsv()` и `fopen()` для записи в файл формата CSV информации, извлекаемой из таблицы базы данных.

Пример 9.11. Запись данных в файл формата CSV

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    print "Couldn't connect to database: " . $e->getMessage();
    exit();
}
// открыть файл формата CSV для записи
$fh = fopen('dish-list.csv','wb');
$dishes = $db->query('SELECT dish_name, price, is_spicy FROM dishes');
while ($row = $dishes->fetch(PDO::FETCH_NUM)) {
```

```
    // записать в массив $row данные в виде строки
    // формата CSV. Функция fputcsv() добавляет
    // знак перевода строки в конце записываемой строки
    fputcsv($fh, $row);
}
fclose($fh);
```

Чтобы отправить веб-клиенту обратно страницу, состоящую только из данных в формате CSV, функции `fputcsv()` нужно дать команду направить данные в стандартный для PHP поток вывода вместо записи в файл. Необходимо также вызвать встроенную в PHP функцию `header()`, чтобы уведомить веб-клиента, что ему предполагается передать документ формата CSV, а не HTML. В примере 9.12 показано, каким образом функция `header()` вызывается с соответствующими аргументами.

Пример 9.12. Смена типа страницы на CSV

```
// уведомить веб-клиента, что ему предполагается передать
// файл формата CSV
header('Content-Type: text/csv');
// уведомить веб-клиента, что содержимое файла формата CSV
// следует просматривать в отдельной программе
header('Content-Disposition: attachment; filename="dishes.csv"');
```

В примере 9.13 показана законченная программа, посылающая правильный заголовок в формате CSV, извлекающая строки из таблицы базы данных и выводящая их на экран. Результат, выводимый из этой программы, может быть непосредственно загружен в программу электронных таблиц из веб-браузера пользователя.

Пример 9.13. Отправка файла формата CSV браузеру

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    print "Couldn't connect to database: " . $e->getMessage();
   exit();
}

// уведомить веб-клиента, что ему передается файл формата CSV
// под названием dishes.csv
header('Content-Type: text/csv');
header('Content-Disposition: attachment; filename="dishes.csv"');

// открыть файл с дескриптором потока вывода
$fh = fopen('php://output'wb');

// извлечь информацию из таблицы базы данных и
// вывести ее на экран
$dishes = $db->query('SELECT dish_name, price,
                             is_spicy FROM dishes');
while ($row = $dishes->fetch(PDO::FETCH_NUM)) {
    fputcsv($fh, $row);
}
```

В коде из примера 9.13 в качестве аргумента функции `fputcsv()` передается **`php://output`** — встроенный дескриптор файла, специально предназначенный для отправки данных туда же, куда они выводятся по команде `print`. Чтобы сформировать более сложные электронные таблицы, включающие в себя формулы, изображения и средства форматирования, следует воспользоваться пакетом PHPExcel (`https://packagist.org/packages/phpoffice/phpexcel`) из библиотечного набора PHPOffice.

Подробнее о том, как пользоваться пакетами, речь пойдет в главе 16.

Проверка полномочий доступа к файлам

Как упоминалось в начале этой главы, в программах на PHP можно организовать только чтение и запись файлов, если интерпретатор PHP разрешает это делать. Но при этом нельзя слепо полагаться на сообщения об ошибках, чтобы выяснить полномочия на доступ к файлу. В языке PHP предоставляются функции, с помощью которых можно определить полномочия отдельных программ.

Чтобы проверить, существует ли файл или каталог, следует вызвать функцию `file_exists()`. В примере 9.14 эта функция применяется для вывода на экран сообщения о том, был ли создан индексный файл в каталоге.

Пример 9.14. Проверка существования файла

```
if (file_exists('/usr/local/htdocs/index.html')) {
    print "Index file is there.";
} else {
    print "No index file in /usr/local/htdocs.";
}
```

Чтобы определить, разрешено ли программе читать или записывать конкретный файл, следует вызвать функцию `s_readable()` или `is_writeable()`. Так, в примере 9.15 проверяется, можно ли прочитать файл, прежде чем извлечь его содержимое с помощью функции `file_get_contents()`.

Пример 9.15. Проверка на полномочия читать файл

```
$template_file = 'page-template.html';
if (is_readable($template_file)) {
    $template = file_get_contents($template_file);
} else {
    print "Can't read template file.";
}
```

В примере 9.16 с помощью функций `fopen()` и `fwrite()` проверяется, доступен ли файл для чтения, прежде чем присоединить к нему строку.

Пример 9.16. Проверка на полномочия записывать файл

```
$log_file = '/var/log/users.log';
if (is_writeable($log_file)) {
    $fh = fopen($log_file,'ab');
    fwrite($fh, $_SESSION['username'] . ' at '
           . strftime('%c') . "\n");
    fclose($fh);
} else {
    print "Cant write to log file.";
}
```

Выявление ошибок

В примерах, приведенных до сих пор в этой главе, демонстрировалось манипулирование файлами без выявления ошибок. Это способствовало краткости приведенных ранее примеров, которые были сосредоточены на таких функциях манипулирования файлами, как `file_get_contents()`, `fopen()` и `fgetcsv()`. Но в то же время это делает приведенные выше примеры неполными. Аналогично взаимодействию с программой базы данных, манипулирование файлами означает взаимодействие с ресурсами, внешними по отношению к прикладной программе. Таким образом, необходимо тщательно проанализировать причины, которые могут вызвать появление ошибок в программе, в том числе нарушение прав доступа к файлам или исчерпание свободного места на диске, а затем принять соответствующие меры для надлежащей обработки подобных ошибок.

На практике для написания надежного кода манипулирования файлами необходимо организовать проверку значения, возвращаемого каждой функцией, связанной с манипулированием файлами. Каждая из таких функций формирует предупреждающее сообщение и возвращает логическое значение `false`, если возникнет ошибка. Так, если отслеживание ошибок активизировано с помощью директивы `track_errors` в файле конфигурации PHP, то текст сообщения об ошибке будет доступным в глобальной переменной `php_errormsg`. В примере 9.17 показано, каким образом выявляются ошибки при выполнении функции `fopen()` или `fclose()`.

Пример 9.17. Выявление ошибок при выполнении функции `fopen()` или `fclose()`

```
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    print "Couldn't connect to database: " . $e->getMessage();
    exit();
}

// открыть файл dishes.txt для записи
$fh = fopen('/usr/local/dishes.txt','wb');
if (! $fh) {
    print "Error opening dishes.txt: $php_errormsg";
} else {
    $q = $db->query("SELECT dish_name, price FROM dishes");
    while ($row = $q->fetch()) {
        // записать каждую строку с оконечным знаком
        // перевода строки в файл dishes.txt
        fwrite($fh, "The price of $row[0] is $row[1] \n");
    }
    if (! fclose($fh)) {
```

```
        print "Error closing dishes.txt: $php_errormsg";
    }
}
```

Если у программы нет разрешения на запись в каталоге `/usr/local`, то функция `fopen()` возвратит логическое значение `false`, а при выполнении кода из примера 9.17 на экран будет выведено следующее сообщение об ошибке:

```
Error opening dishes.txt: failed to open stream: Permission denied

[ Ошибка при открытии файла dishes.txt:
  не удалось открыть поток ввода-вывода:
  В разрешении на доступ к файлу отказано ]
```

Кроме того, будет сгенерировано предупреждающее сообщение такого содержания:

```
Warning: fopen(/usr/local/dishes.txt): failed to open stream:
Permission denied in dishes.php on line 5

[ Предупреждение: fopen(/usr/local/dishes.txt):
  не удалось открыть поток ввода-вывода:
  В разрешении на доступ к файлу dishes.php
  отказано в строке кода 5 ]
```

О том, как организовать вывод сообщений, предупреждающих об ошибках, речь пойдет в разделе “Управление выводом сообщений об ошибках” главы 12.

То же самое происходит и с функцией `fclose()`. Если эта функция возвратит логическое значение `false`, то на экран будет выведено сообщение `"Error closing dishes.txt"` (Ошибка при закрытии файла `dishes.txt`). Иногда операционные системы буферизуют данные, записываемые в файл с помощью функции `fwrite()`, и поэтому данные фактически не сохраняются в файле до тех пор, пока не будет вызвана функция `fclose()`. Если же на диске не окажется свободного места для записываемых данных, ошибка может возникнуть при вызове функции `fclose()`, а не функции `fwrite()`.

Несколько сложнее выявить ошибки при выполнении других функций манипулирования файлами, например, `fgets()`, `fwrite()`, `fgetcsv()`, `file_get_contents()` или `file_put_contents()`. Ведь в этом случае приходится принимать специальные меры, чтобы отличить значения, возвращаемые подобными функциями при возникновении ошибок, от данных, которые они возвращают при удачном исходе.

При неудачном исходе функции `fgets()`, `file_get_contents()` или `fgetcsv()` возвращают логическое значение `false`. Но вполне возможно, что эти функции могут завершиться удачно и возвратить значение, которое вычисляется при сравнении как ложное (`false`). Так, если функция `file_get_contents()` выполняет чтение файла, состоящего только из одного символа **0**, она возвратит строку, содержащую единственный символ **0**. Но, как пояснялось в разделе “Общее представление об истинности или ложности” главы 3, такая символьная строка считается ложной.

Чтобы правильно оценить возвращаемое значение, следует воспользоваться операцией тождественности. Таким образом, сравнив возвращаемое значение с логическим значением `false`, можно выяснить, что ошибка произошла лишь в том случае, если функция возвратит логическое значение `false`, а не символьную строку, которая вычисляется как ложная (`false`). В примере 9.18 показано применение операции тождественности для выявления ошибки при выполнении функции `file_get_contents()`.

Пример 9.18. Выявление ошибки при выполнении функции `file_get_contents()`

```
$page = file_get_contents('page-template.html');
// Обратите внимание на три знака равенства в следующем
// проверочном выражении
if ($page === false) {
    print "Couldn't load template: $php_errormsg";
} else {
    // ... здесь следует обработка выявленной ошибки
}
```

Таким же способом можно воспользоваться и при вызове функции `fgets()` или `fgetcsv()`. В примере 9.19 показано, как правильно выявлять ошибки при выполнении функций `fopen()`, `fgets()` и `fclose()`.

Пример 9.19. Выявление ошибок при выполнении функций `fopen()`, `fgets()` и `fclose()`

```
$fh = fopen('people.txt','rb');
if (! $fh) {
    print "Error opening people.txt: $php_errormsg";
} else {
    while (! feof($fh)) {
        $line = fgets($fh);
        if ($line !== false) {
            $line = trim($line);
            $info = explode('|', $line);
            print '<li><a href="mailto:' . $info[0] . '">'
                                . $info[1] ."</li>\n";
        }
    }
    if (! fclose($fh)) {
        print "Error closing people.txt: $php_errormsg";
    }
}
```

При удачном исходе функции `fwrite()`, `fputcsv()` и `file_put_contents()` возвращают количество записанных ими байт. А при неудачном исходе функция `fwrite()` или `fputcsv()` возвращает логическое значение `false`, и в этом случае можно воспользоваться операцией тождественности таким же образом, как и для анализа значения, возвращаемого функцией `fgets()`. Несколько иначе дело обстоит с функцией `file_put_contents()`. В зависимости от характера ошибки она возвратит логическое значение **`false`** или **`-1`**, а следовательно, придется проверить возможность возврата обоих этих значений. В примере 9.20 показано, как выявлять и обрабатывать ошибки, возвращаемые функцией `file_put_contents()`.

Пример 9.20. Выявление и обработка ошибок, возвращаемых функцией `file_put_contents()`

```
// загрузить файл из примера 9.1
$page = file_get_contents('page-template.html');

// ввести заглавие страницы
$page = str_replace('{page_title}', 'Welcome', $page);

// окрасить страницу голубым цветом после полудня и
```

```
// зеленым цветом с утра
if (date('H' >= 12)) {
    $page = str_replace('{color}', 'blue', $page);
} else {
    $page = str_replace('{color}', 'green', $page);
}

// взять имя пользователя из переменной сохраненного
// предыдущего сеанса работы
$page = str_replace('{name}', $_SESSION['username'], $page);

$result = file_put_contentsCpage.html', $page);
// непременно проверить, возвращает ли функция file_put_contents()
// логическое значение false или же значение -1
if (($result === false) || ($result == -1)) {
    print "Couldn't save HTML to page.html";
}
```

Санобработка предоставляемых извне путей к файлам

Данные, передаваемые в форме или URL, могут вызвать такие же осложнения при их отображении (например, атаки типа межсайтового выполнения сценариев) или вводе в запрос SQL (в том числе атаки умышленного внесения запросов SQL), как и в том случае, когда они представляются в виде пути к файлу или части этого пути. У этого вида атак нет специального названия, но они могут оказаться такими же опустошительными.

У подобных осложнений одна и та же причина: наличие специальных символов, которые должны быть непременно экранированы, чтобы утратить свое особое назначение. В качестве специальных символов в путях к файлам зачастую употребляются знаки косой черты (/), разделяющие отдельные части пути к файлу, а также две точки (..), обозначающие переход к каталогу, находящемуся на один уровень выше в иерархии файловой системы.

Например, замысловатый путь к файлу `/usr/local/data/../../../etc/passwd` указывает не на местонахождение файла в каталоге `/usr/local/data`, а на местонахождение файла `/etc/passwd`, где в большинстве систем Unix хранится список учетных записей пользователей. В частности, путь к файлу `/usr/local/data/../../../etc/passwd` означает переход из каталога `/usr/local/data` на один уровень вверх к каталогу `/usr/local`, затем на еще один уровень вверх к каталогу `/usr` и еще на один уровень вверх к каталогу `/`, находящемуся на вершине иерархии файловой системы, а далее — на один уровень вниз к каталогу `/etc` и, наконец, к файлу `passwd`.

Какие трудности это может представлять для программ на PHP? Если в пути к файлу используются данные из формы, то возникает уязвимость к атакам, способным предоставить злоумышленнику доступ к тем частям файловой системы, которые желательно сделать недоступными для посторонних. И злоумышленник может получить такой доступ, если не подвергнуть предварительной очистке данные из переданной на обработку формы. В примере 9.21 демонстрируется способ удаления всех знаков косой черты (/) и двух точек (..) из параметра переданной на обработку формы перед внедрением этого параметра в путь к файлу.

Пример 9.21. Очистка параметра, вставляемого из переданной на обработку формы в путь к файлу

```
// удалить знаки косой черты (/) из имени пользователя,
// введенного в форме
$user = str_replace('/', '', $_POST['user']);
```

```
// удалить последовательности из двух точек (..) из
// имени пользователя, введенного в форме
$user = str_replace('..', '', $user);

if (is_readable("/usr/local/data/$user")) {
    print 'User profile for ' . htmlentities($user) .': <br/>';
    print file_get_contents("/usr/local/data/$user");
}
```

Если злонамеренный пользователь введет путь к файлу `../../../etc/passwd` в поле параметра `user` заполняемой формы, то в коде из примера 9.21 этот путь будет преобразован в имя файла `etcpasswd` перед вставкой в путь к файлу, передаваемый функции `file_get_contents()`.

Еще один полезный способ избавиться от всякой мерзости, вводимой пользователями, состоит в применении функции `realpath()`. Эта функция преобразует запутанный путь к файлу, содержащий последовательный ряд двух точек (`..`), в очищенный от них путь, точнее указывающий местонахождение файла. Так, в результате вызова `realpath('/usr/local/data/../../../etc/passwd')` возвращается символьная строка, содержащая путь `/etc/passwd`. В примере 9.22 демонстрируется применение функции `realpath()` с целью проверить достоверность пути к файлу после вставки в него данных из переданной на обработку формы.

Пример 9.22. Очистка пути к файлу с помощью функции `realpath()`

```
$filename = realpath("/usr/local/data/$_POST[user]");
// убедиться в том, что файл указан в переменной $filename
// по пути /usr/local/data
if (('/usr/local/data/' == substr($filename, 0, 16)) &&
                           is_readable($filename)) {
    print 'User profile for ' . htmlentities($_POST['user']) .': <br/>';
    print file_get_contents($filename);
} else {
    print "Invalid user entered.";
}
```

Если элемент массива `$_POST['user']` в примере 9.22 содержит строковое значение `james`, то в переменной `$filename` устанавливается путь к файлу `/usr/local/data/james` и выполняется блок кода в условном операторе `if()`. Но если элемент массива `$_POST['user']` содержит что-нибудь подозрительное, например, путь к файлу `../secrets.txt`, то в переменной `$filename` устанавливается путь `/usr/local/secrets.txt` и проверка в условном операторе `if()` не проходит, а на экран выводится сообщение `"Invalid user entered."` (Введено неверное имя пользователя).

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о происхождении полномочий на доступ к файлам в интерпретаторе PHP.
- Чтение всего файла с помощью функции `file_get_contents()`.
- Запись всего содержимого в файл с помощью функции `file_put_contents()`.
- Построчное чтение содержимого файла с помощью функции `file()`.
- Открытие и закрытие файлов с помощью функций `fopen()` и `fclose()`.

- Чтение отдельных строк из файла с помощью функции `fgets()`.
- Построчное чтение содержимого файла в цикле `while()` с применением функции `feof()`.
- Применение знаков косой черты в путях к файлам во всех операционных системах.
- Задание разных режимов доступа к файлу при вызове функции `fopen()`.
- Запись данных в файл с помощью функции `fwrite()`.
- Чтение отдельных строк из файла формата CSV с помощью функции `fgetcsv()`.
- Запись отдельных строк в файл формата CSV с помощью функции `fputcsv()`.
- Применение стандартного потока вывода `php://output` для отображения выводимых результатов.
- Выявление факта существования файла с помощью функции `file_exists()`.
- Проверка прав доступа к файлам с помощью функций `s_readable()` и `is_writeable()`.
- Выявление ошибок, возвращаемых функциями доступа к файлам.
- Представление о том, когда именно следует проверять возвращаемое значение с помощью операции тождественности.
- Удаление потенциально опасных частей из предоставляемых извне путей к файлам.

Упражнения

1. Создайте вне интерпретатора PHP новый HTML-файл шаблона по образцу, приведенному в примере 9.1. Напишите программу, в которой функции `file_get_contents()` и `file_put_contents()` применяются для чтения HTML-файла шаблона, вместо переменных в шаблон подставляются соответствующие значения, а новая страница сохраняется в отдельном файле.

2. Создайте вне интерпретатора PHP новый файл, содержащий ряд адресов электронной почты в отдельных строках, причем некоторые адреса должны присутствовать в файле неоднократно. Присвойте новому файлу имя `addresses.txt`. Затем напишите на PHP программу, читающую каждую строку из файла `addresses.txt` и подсчитывающую, сколько раз каждый адрес упоминается в данном файле. Для каждого адреса из файла `addresses.txt` эта программа должна записывать отдельную строку в другой файл, называемый `addresses-count.txt`. Каждая строка из файла `addresses-count.txt` должна состоять из числа, обозначающего, сколько раз данный адрес упоминается в файле `addresses.txt`, запятой и самого адреса электронной почты. Такие строки должны записываться в файл `addresses-count.txt` в отсортированном порядке, начиная с адреса, чаще всего упоминаемого в файле `addresses.txt`, и кончая адресом, реже всего упоминаемым в файле `addresses.txt`.

3. Отобразите файл формата CSV в виде HTML-таблицы. Если у вас нет под рукой такого файла (или программы электронных таблиц), воспользуйтесь данными из примера 9.9.

4. Напишите на PHP программу, отображающую форму, в которой пользователю предлагается указать имя файла в корневом каталоге документов на вебсервере. Если такой файл существует на веб-сервере, доступен для чтения и находится в корневом каталоге документов, то программа должна отобразить его содержимое. Так, если пользователь введет имя файла `article.html`, программа должна отобразить содержимое файла `article.html`, находящегося в корневом каталоге документа. А если пользователь введет путь к файлу

`catalog/show.php`, программа должна отобразить содержимое файла `show.php` из каталога `catalog`, входящего в корневой каталог документа. О том, как находить корневой каталог документа на веб-сервере, см. в табл. 7.1.

5. Видоизмените программу из предыдущего упражнения, чтобы она отображала только файлы с расширением **`.html`**. Ведь разрешать пользователям просматривать исходный код PHP любой страницы веб-сайта опасно, если она содержит уязвимую информацию (например, имена пользователей и их пароли).

ГЛАВА 10

Сохранение сведений о пользователях в cookie-файлах и сеансах

Веб-сервер очень похож на продавца в магазине деликатесов, которого одолевают своими запросами назойливые покупатели. Одному требуется полфунта солонины, а другому — фунт тонко нарезанной копченой говядины. Продавец проворно нарезает деликатес и упаковывает его, удовлетворяя запросы по очереди. Аналогичные запросы в электронном виде посылают веб-клиенты, например, предоставить файл `/catalog/yak.php` или принять форму, передаваемую на обработку. А сервер формирует ответы с помощью интерпретатора PHP, чтобы удовлетворить принятые запросы.

Но преимущество продавца из магазина деликатесов над веб-сервером заключается в памяти. Он естественным образом связывает вместе все запросы, поступающие от конкретного покупателя. Интерпретатор PHP и веб-сервер неспособны на такое, если не принять дополнительные меры. И здесь на помощь приходят cookie-файлы.

С помощью cookie-файла отдельный веб-клиент идентифицируется веб-сервером и интерпретатором PHP. Всякий раз, когда веб-клиент делает запрос, он посылает вместе с ним cookie-файл. Интерпретатор PHP читает этот файл и выясняет, что конкретный запрос поступил от того же самого веб-клиента, который посылал предыдущие запросы, которым сопутствовал тот же cookie-файл.

Если бы покупателям в магазине деликатесов пришлось иметь дело с продавцом, обладающим слабой памятью, им довелось бы принять аналогичную стратегию. Их запросы на обслуживание выглядели бы следующим образом.

“Я покупатель №56, мне нужно полфунта солонины”.

“Я покупатель №29, мне нужно три книша”.

“Я покупатель №56, мне нужно два фунта копченой говядины”.

“Я покупатель №77, я возвращаю эту буханку ржаного хлеба — он черствый”.

“Я покупатель №29, мне нужно салями”.

Подобные запросы составляют cookie-файл. Он дает продавцу нить, чтобы связать вместе запросы отдельного покупателя.

В cookie-файле содержится имя (например, “покупатель”) и значение (в частности, “77” или “ronald”). В следующем разделе поясняется, как манипулировать отдельными cookie-файлами в программах, т.е. устанавливать, читать и удалять их.

Один cookie-файл лучше всего подходит для отслеживания какого-то одного фрагмента информации. Но зачастую требуется отслеживать больше информации о пользователе (например, содержи-

мое корзины для покупок пользователя), и пользоваться для этой цели несколькими cookie-файлами неудобно. В качестве выхода из этого положения в PHP предоставляется *сеанс*.

В сеансе cookie-файлы служат для различения пользователей, упрощая ведение временного массива данных для каждого пользователя на сервере. Этот массив данных сохраняется в промежутках между отдельными запросами. По одному запросу можно ввести переменную в сеанс пользователя (например, разместить какой-нибудь товар в корзине для покупок), а по другому запросу извлечь содержимое сеанса (в частности, на странице выписки счета, когда потребуется указать все товары в корзине для покупок). В разделе "Активизация сеансов" далее в этой главе поясняется, как приступить к работе с сеансами, а в разделе "Сохранение и извлечение информации" описаны особенности работы с сеансами.

Манипулирование cookie-файлами

Чтобы установить cookie-файл, следует воспользоваться функцией `setcookie()`. Эта функция дает веб-клиенту команду запомнить имя cookie-файла и сохранить в нем значение, а при последующих запросах — отправлять их обратно на сервер. Так, в примере 10.1 в cookie-файле `userid` устанавливается значение `ralph`.

Пример 10.1. Установка cookie-файла

```
setcookie('userid','ralph');
```

Чтобы прочитать установленный ранее cookie-файл из программы на PHP, следует воспользоваться автоглобальным массивом `$_COOKIE`. В примере 10.2 демонстрируется вывод значения из cookie-файла на экран.

Пример 10.2. Вывод значения из cookie-файла на экран

```
print 'Hello, ' . $_COOKIE['userid'];
```

Значение, предоставляемое функции `setcookie()` для запоминания в cookie-файле, может быть строковым или числовым. Оно не может быть массивом или еще более сложной структурой данных.

Функция `setcookie()` кодирует в формате URL значение, сохраняемое в cookie-файле, прежде чем отправить его веб-клиенту. Это означает, что пробел преобразуется в знак **+**, а все остальные символы, кроме букв, цифр, знаков подчеркивания, дефисов и точек, преобразуются в знак процента и шестнадцатеричное значение символа в коде ASCII. Если же не требуется манипулировать значением в cookie-файле средствами PHP, вместо функции `setcookie()` следует воспользоваться функцией `setrawcookie()`. Но в этом случае значение в cookie-файле не может содержать знаки **=**, **,**, **;** и любые пробелы.

Когда вызывается функция `setcookie()`, в ответ, формируемый интерпретатором PHP для отправки обратно веб-клиенту, включается специальный заголовок, сообщающий веб-клиенту о новом cookie-файле. А при последующих запросах веб- клиент отправляет имя данного cookie-файла и хранящееся в нем значение обратно на веб-сервер. Такой двухэтапный диалог наглядно показан на рис. 10.1.

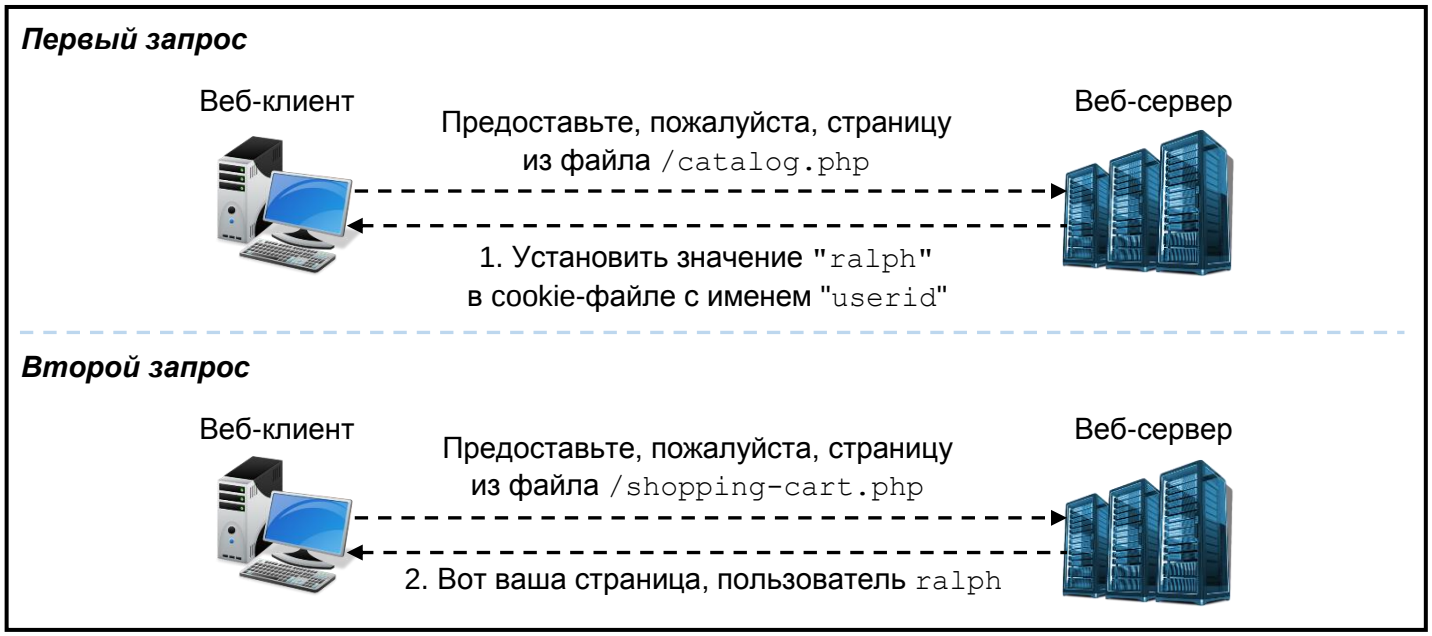


Рис. 10.1: Взаимодействие клиента и сервера при установке cookie-файла

Как правило, функцию `setcookie()` следует вызывать, прежде чем на странице будет сформирован какой-нибудь выводимый на экран результат. Это означает, что вызов функции `setcookie()` должен предшествовать любым операторам `print`. Но это также означает, что на странице *нельзя* вводить никакого текста перед начальным дескриптором `<?php`, предшествующим вызову функции `setcookie()`. В разделе “Причины для размещения вызовов функций `setcookie()` и `session_start()` вначале страницы” далее в этой главе поясняется, почему такое требование существует и как его обойти в некоторых случаях. В примере 10.3 показано, как правильно размещать вызов функции `setcookie()` вначале страницы.

Пример 10.3. Правильное размещение вызова функции `setcookie()` вначале страницы

```
<?php
setcookie('userid','ralph');
?>
<html><head><title>Page with cookies</title><head>
<body>
This page sets a cookie properly, because the PHP block
with setcookie() in it comes before all of the HTML.
</body></html>
```

Установленные cookie-файлы появляются в массиве `$_COOKIE` лишь в том случае, если веб-клиент отправляет их вместе с запросом. Это означает, что имя cookie-файла и значение в нем не появляются в массиве `$_COOKIE` сразу же после вызова функции `setcookie()`. И только после того, как веб-клиент воспримет ответ об установке cookie-файла, он будет знать о cookie-файле. А сведения о cookie-файле появятся в массиве `$_COOKIE` лишь после того, как веб-клиент отправит cookie-файл на сервер вместе с последующим запросом.

По умолчанию срок действия cookie-файла соответствует сроку действия вебклиента. При выходе из браузера Safari или Firefox соответствующий cookie-файл удаляется. Чтобы продлить срок действия cookie-файла (или, наоборот, прекратить его действие раньше срока), следует передать необязательно устанавливаемое истечение срока действия cookie-файла в качестве третьего аргумента функции `setcookie()`. В примере 10.4 показано, как устанавливается разное истечение срока действия cookie-файлов.

Пример 10.4. Установка истечения срока действия cookie-файлов

```
// Срок действия cookie-файла истекает через час
setcookie('short-userid'ralph',time() + 60*60);
// Срок действия cookie-файла истекает через день
setcookie('longer-userid','ralph',time() + 60*60*24);
// Срок действия cookie-файла истекает в полден 1 октября 2019 г.
$d = new DateTime("2019-10-01 12:00:00");
setcookie('much-longer-userid','ralph', $d->format('U'));
```

Истечение срока действия cookie-файла необходимо указывать функции `setcookie()` в виде количества секунд, отсчитываемых от 1 января 1970 г. Для указания подходящего истечения срока действия cookie-файла используется функция `time()` и символ форматирования **U**, указываемый при вызове метода `DateTime::format()`.[1] Функция `time()` возвращает текущее количество секунд, прошедших после 1 января 1970 г. (т.е. с момента, называемого в ОС Unix "эпохой"). Так, если требуется установить истечение срока действия cookie-файла в виде определенного количества секунд, отсчитываемых от текущего момента, это значение следует сложить со значением, возвращаемым функцией `time()`. В одном часе содержится 60 минут, а в каждой минуте — 60 секунд, поэтому произведение `60*60` дает количество секунд в одном часе. Таким образом, сумма `time() + 60*60` равна количеству секунд, истекающих через час, начиная с текущего момента. Аналогично произведение `60*60*24` дает количество секунд в одном дне, и поэтому сумма `time() + 60*60*24` равна количеству секунд, истекающих через день, начиная с текущего момента.

Символ форматирования **U**, указываемый при вызове метода `DateTime::format()`, обозначает количество секунд, истекающих с момента времени, представленного объектом класса `DateTime`. Установка cookie-файла с конкретным временем истечения срока его действия продлевает этот срок, даже если веб-клиент завершает свою работу и перезапускается снова. Помимо времени истечения срока действия, имеются и другие параметры cookie-файла, которые полезно настраивать, в том числе путь к данному файлу, домен и два параметра, имеющих непосредственное отношение к безопасности.

Как правило, cookie-файлы посылаются обратно на сервер только вместе с запросами страниц, находящихся в том же самом (или расположенном ниже) каталоге, где располагается страница, на которой устанавливается cookie-файл. Так, cookie- файл, устанавливаемый на веб-странице по адресу `http://www.example.com/buy.php`, отправляется вместе со всеми запросами обратно на сервер `www.example.com`, поскольку файл `buy.php` находится в каталоге верхнего уровня на веб-сервере. А cookie-файл, устанавливаемый на веб-странице по адресу `http://www.example.com/catalog/list.php`, отправляется обратно на сервер вместе с другими запросами из каталога `catalog`, например, по адресу `http://www.example.com/catalog/search.php`. Он также посылается обратно на сервер вместе с запросами страниц из подкаталогов, входящих в каталог `catalog`, например, по адресу `http://www.example.com/catalog/detailed/search.php`. Но в то же время он не посылается обратно на сервер с запросами страниц, находящихся выше или вне каталога `catalog`, например, по адресу `http://www.example.com/sell.php` или `http://www.example.com/users/profile.php`.

Часть, указываемая в URL после имени хоста (например, `/buy.php`, `/catalog/list.php` или `/users/profile.php`), называется *путем*. Чтобы сообщить веб-клиенту придерживаться другого пути, когда требуется выяснить, следует ли отправить cookie-файл на сервер, данный путь следует указать в качестве четвертого аргумента функции `setcookie()`. Удобнее всего предоставить путь **/**, который означает следующее: отправить данный cookie-файл со всеми запросами обратно на сервер. Так, в примере 10.5 устанавливается cookie-файл по заданному пути **/**.

[1] Подробнее о классе `DateTime` и его методе `format()` речь пойдет в главе 15.

Пример 10.5. Установка cookie-файла по заданному пути

```
setcookie('short-userid','ralph',0,'/');
```

В примере 10.5 аргумент функции `setcookie()`, определяющий время истечения срока действия cookie-файла, устанавливается равным нулю. Этим функции `setcookie()` сообщается порядок применения устанавливаемого по умолчанию (т.е. по завершении работы веб-клиента) времени истечения срока действия cookie-файла. Если же функции `setcookie()` передается путь, то в качестве другого аргумента необходимо также указать время истечения срока действия, устанавливаемого для cookie-файла. Это может быть конкретная величина времени (например, сумма `time() + 60*60`) или нулевое значение, употребляемое в качестве времени истечения срока действия, устанавливаемого по умолчанию для cookie-файла.

Задавать путь, отличающийся от пути **/**, целесообразно в том случае, если вебсайт размещается на коллективно используемом сервере, а все его страницы — в конкретном каталоге. Так, если имеется веб-пространство, доступное по адресу `http://students.example.edu/ alice/`, то при установке cookie-файла следует задать путь `/ alice/`, как показано в примере 10.6.

Пример 10.6. Установка cookie-файла вместе с путем для размещения в конкретном каталоге

```
setcookie('snort-userid','ralph',0,'/~alice/');
```

При заданном пути `/~alice/` к cookie-файлу `short-userid` он посылается вместе с запросом по адресу `http://students.example.edu/~alice/search.php`, но не с запросами веб-страниц других студентов, например, по адресу `http://students.example.edu/~bob/sneaky.php` или `http://students.example.edu/~charlie/search.php`.

Следующий аргумент, определяющий, какие именно запросы веб-клиент решит посылать, обозначает конкретный домен. По умолчанию cookie-файлы посылаются только с запросами на тот же самый хост, где они установлены. Так, если установить cookie-файл по адресу `http://www.example.com/login.php`, он будет отправлен вместе с другими запросами обратно на сервер `www.example.com`, но не с запросами на сервер `shop.example.com`, `www.yahoo.com` или `www.example.org`.

Такое поведение можно немного изменить, дав веб-сайту команду посылать cookie-файл с запросами на хост, имя которого имеет окончание, совпадающее с пятым аргументом функции `setcookie()`. Чаще всего такая возможность используется для установки домена вроде **`.example.com`**, где начальная точка важна для более старых веб-клиентов. Этим веб-сайту сообщается, что cookie-файл следует отправлять вместе с последующими запросами на сервер `www.example.com`, `shop.example.com`, `testing.development.example.com` или любой другой сервер, имя которого оканчивается на **`.example.com`**. В примере 10.7 демонстрируется именно такой способ установки cookie-файла.

Пример 10.7. Установка cookie-файла с заданным доменом

```
setcookie('short-userid','ralph',0,'/','.example.com');
```

Срок действия cookie-файла из примера 10.7 истекает, когда веб-клиент завершает свою работу. Этот cookie-файл посылается вместе с запросами страниц в любом каталоге (поскольку указан путь **/**) на любом сервере, имя которого оканчивается на **`.example.com`**.

Домен, предоставляемый функции `setcookie()`, должен совпадать с окончанием имени сервера. Так, если написанные на PHP программы размещаются на сервере `students.example.edu`, то в качестве домена для cookie-файла *нельзя* указать **`.yahoo.com`**, чтобы отправить cookie-файл обратно на все серверы, находящиеся в домене `yahoo.com`. Но в качестве домена для cookie-файла можно указать **`.example.edu`**, чтобы отправить cookie-файл вместе со всеми запросами обратно на любой сервер, находящийся в домене `example.edu`.

Два последних необязательных аргумента функции `setcookie()` оказывают влияние на настройки параметров безопасности в cookie-файле. Так, если указать логическое значение `true` в

качестве шестого аргумента функции `setcookie()`, вебклиент должен возвращать cookie-файл только по защищенному соединению, которое устанавливается по URL с префиксом **`https`**. Хотя подобные вызовы функции `setcookie()` следует делать лишь в том случае, если запрос страницы, где вызывается функция `setcookie()`, выполняется по защищенному соединению. Но этим вебклиенту предписывается также не отправлять cookie-файл вместе с последующими запросами обратно на сервер по незащищенному соединению.

И, наконец, если указать логическое значение `true` в качестве седьмого аргумента функции `setcookie()`, веб-клиент должен трактовать данный cookie-файл как доступный *только* по сетевому протоколу Http (HttpOnly). Клиент и сервер обмениваются таким cookie-файлом, как обычно, но он недоступен для сценария JavaScript на стороне клиента. Благодаря этому обеспечивается определенный уровень защиты от атак типа межсайтового выполнения сценариев, описанных в разделе "HTML и JavaScript" главы 7. Как показано в примере 10.8, срок действия устанавливаемого cookie-файла истекает через 24 часа, на него не накладывается никаких ограничений в отношении пути или домена, он должен отправляться обратно на сервер по защищенному соединению и недоступен для сценариев JavaScript на стороне клиента.

Пример 10.8. Установка cookie-файла с параметрами безопасности

```
// Пустое значение null, указываемое в качестве пути и домена,
// означает, что интерпретатору PHP не следует вообще размещать
// какой-нибудь путь или домен в cookie-файле
setcookie('short-userid','ralph',0, null, null, true, true);
```

Чтобы удалить cookie-файл, достаточно вызвать функцию `setcookie()` с именем удаляемого cookie-файла и пустой символьной строкой в качестве его значения (пример 10.9).

Пример 10.9. Удаление cookie-файла

```
setcookie('short-userid','');
```

Если cookie-файл установлен с нестандартными значениями параметров времени истечения срока действия, пути или домена, то те же самые значения параметров придется указать снова при удалении данного cookie-файла, чтобы эта операция была выполнена правильно. Чаще всего cookie-файлы достаточно устанавливать со стандартными значениями параметров времени истечения срока действия, пути или домена. Тем не менее ясное представление о порядке изменения этих значений помогает лучше понять, как лучше настроить сеанс PHP на специальное поведение.

Активизация сеансов

По умолчанию в сеансах применяется cookie-файл, называемый `PHPSESSID`. Если начать сеанс работы на странице, интерпретатор PHP проверит наличие этого cookie-файла, а если он отсутствует, то установит его. В качестве значения в cookie-файле `PHPSESSID` хранится произвольная буквенно-цифровая символьная строка. Каждому веб-клиенту присваивается идентификатор сеанса, обозначающий в cookie-файле `PHPSESSID` однозначный характер веб-клиента для сервера. Благодаря этому интерпретатор PHP может поддерживать отдельные массивы данных для каждого веб-клиента. Диалог между веб-клиентом и сервером вначале сеанса наглядно показан на рис. 10.2.

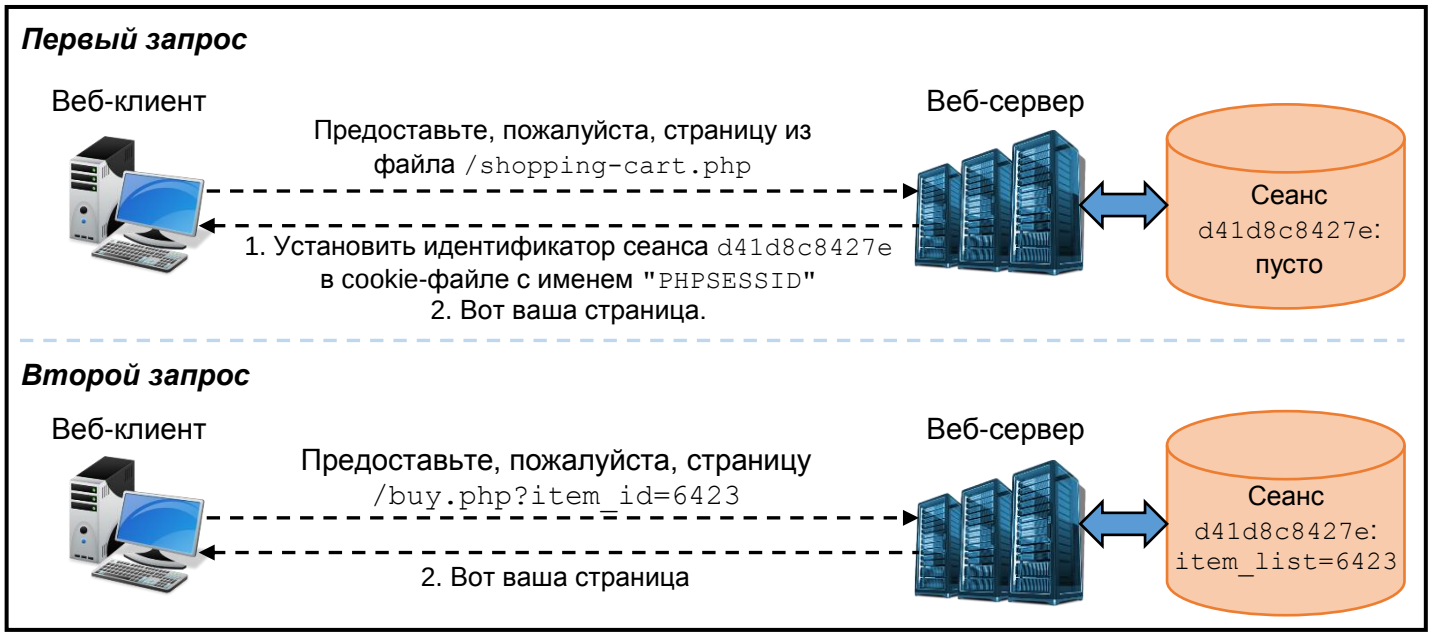


Рис. 10.2: Взаимодействие клиента и сервера вначале сеанса

Чтобы воспользоваться сеансом работы на странице, следует вызвать функцию `session_start()` в самом начале сценария на PHP. Как и функцию `setcookie()`, функцию `session_start()` следует вызывать перед выводом любых результатов на экран. Если же требуется воспользоваться сеансами работы на всех страницах, с этой целью следует установить значение On в директиве `session.auto_start` конфигурации сервера. Как только это будет сделано, вызывать функцию `session_start()` на каждой странице больше не придется. В приложении А к данной книге поясняется, как изменять параметры конфигурации PHP.

Сохранение и извлечение информации

Данные сеанса хранятся в автоглобальном массиве `$_SESSION`. Для манипулирования данными сеанса достаточно прочитать и внести изменения в содержимое элементов массива `$_SESSION`. В примере 10.10 демонстрируется счетчик страниц, в котором массив `$_SESSION` применяется для того, чтобы следить, сколько раз пользователь просматривал страницу.

Пример 10.10. Подсчет количества просмотров страницы с помощью сеанса

```
session_start();
if (isset($_SESSION['count'])) {
    $_SESSION['count'] = $_SESSION['count'] + 1;
} else {
    $_SESSION['count'] = 1;
}
print "You've looked at this page " . $_SESSION['count'] . ' times.';
```

Когда пользователь посещает страницу из примера 10.10 в первый раз, веб-клиент не отправляет cookie-файл `PHPSESSID` на сервер. В функции `session_start()` создается новый сеанс для пользователя, а на сервер посылается cookie-файл `PHPSESSID` с новым идентификатором сеанса.

При создании сеанса массив `$_SESSION` еще пуст. Поэтому в коде из примера 10.10 проверяется наличие ключа `count` в массиве `$_SESSION`. Если такой ключ есть, то подсчет посещений страницы инкрементируется. В противном случае устанавливается значение **1**, обозначающее первое посещение страницы, а на экран выводится приведенное ниже сообщение. По окончании обработки

запроса данные из массива $_SESSION сохраняются в файле на веб-сервере с идентификатором соответствующего сеанса.

```
You've looked at this page 1 times.
```

При последующем посещении страницы веб-клиент отправит cookie-файл PHPSESSID. Функция session_start() обнаружит идентификатор сеанса в этом cookie-файле и загрузит файл, содержащий информацию о сеансе, сохраненную по данному идентификатору. В данном случае сохраненная информация просто сообщает, что в элементе массива $_SESSION['count'] содержится значение 1. Далее значение в элементе массива $_SESSION['count'] увеличивается до 2, и на экран выводится сообщение "You've looked at this page 2 times."(Вы просматривали эту страницу 2 раза). По окончании обработки запроса содержимое массива $_SESSION (теперь это значение 2 в элементе массива $_SESSION['count']) снова сохраняется в упомянутом выше файле.

Интерпретатор PHP отслеживает содержимое массива $_SESSION отдельно по каждому идентификатору сеанса. При выполнении программы на PHP массив $_SESSION содержит сохраненные данные только из одного сеанса, который является активным и соответствует идентификатору, отправленному на сервер в cookie-файле PHPSESSID. Для каждого пользователя в cookie-файле PHPSESSID хранится отдельное значение.

Если функция session_start() вызывается в самом начале страницы (или же если в директиве session.auto_start конфигурации сервера установлено значение **On**), данные о сеансе работы пользователя становятся доступными на данной странице. Массив $_SESSION служит для обмена информацией между страницами. В примере 10.11 демонстрируется законченная программа, отображающая форму, в которой пользователь выбирает блюдо в нужном количестве. Выбранное блюдо и его количество вводятся в переменную сеанса order.

Пример 10.11. Сохранение данных из формы в сеансе

```
require 'FormHelper.php';

session_start();

$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' =>
                              'Abalone with Marrow and Duck Feet');

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        process_form($input);
    }
} else {
    show_form();
}

function show_form(terrors = array()) {
    // Собственные значения, устанавливаемые по умолчанию,
```

```
    // отсутствуют, поэтому и нечего передавать конструктору
    // класса FormHelper
    $form = new FormHelper();

    // построить HTML-таблицу из сообщений об ошибках для
    // последующего применения
    if ($errors) {
        $errorHtml = '<ul><li>';
        $errorHtml .= implode('</li><li>', $errors);
        $errorHtml .= '</li></ul>';
    } else {
        $errorHtml = '';
    }

    // Это небольшая форма, поэтому ниже выводятся ее составляющие
    print <<<_FORM_
      <form method="POST" action=
                          "{$ form->encode($_SERVER['PHP_SELF'])}">
        $errorHtml
        Dish: {$form->select($GLOBALS['main_dishes'],
                                    ['name' => 'dish'])} <br/>
        Quantity: {$form->input('text',
                                ['name' => 'quantity'])} <br/>
        {$form->input('submit',['value' => 'Order'])}
      </form>
      _FORM_;
}

function validate_form() {
    $input = array();
    $errors = array();
    // Блюдо, выбранное из списка, должно быть достоверным
    $input['dish'] = $_POST['dish'] ?? '';
    if (! array_key_exists($input['dish'],
                           $GLOBALS['main_dishes'])) {
        $errors[] = 'Please select a valid dish.';
    }

    $input['quantity'] = filter_input(INPUT_POST, 'quantity',
                                      FILTER_VALIDATE_INT,
                                      array('options' =>
                                         array('min_range' => 1)));
    if (($input['quantity'] === false) ||
                ($input['quantity'] === null)) {
        $errors[] = 'Please enter a quantity.';
    }
    return array($errors, $input);
}

function process_form($input) {
         $_SESSION['order'][] = array('dish' => $input['dish'],
                                      'quantity' => $input['quantity']);
```

```
    print 'Thank you for your order.';
}
```

Большая часть кода обработки формы из примера 10.11 должна быть вам уже знакома. Как и в примерах 8.28 и 8.53, вспомогательный класс для вывода элементов заполняемой формы загружается из исходного файла `FormHelper.php`, а функции `show_form()`, `validate_form()` и `process_form()` служат для отображения, проверки достоверности и обработки данных из переданной формы.

Но в функции `process_form()` из примера 10.11 выгодно используются сеансы. Всякий раз, когда форма передается на обработку с достоверными данными, вводится элемент массива `$_SESSION['order']`. Данные сеанса не ограничиваются символьными строками и числами, как в cookie-файлах. Массив `$_SESSION` можно рассматривать таким же образом, как и любой другой. Синтаксис `$_SESSION['order'][]` означает, что элемент массива `$_SESSION['order']` следует трактовать как отдельный массив, в конце которого вводится новый элемент. В данном случае в конце массива `$_SESSION['order']` вводится двухэлементный массив, содержащий информацию о блюде и его количестве из переданной на обработку формы.

В программе, демонстрируемой в примере 10.12, на экран выводится список заказанных блюд, составленный в результате доступа к информации, сохраненной в сеансе программой из примера 10.11.

Пример 10.12. Вывод данных сеанса на экран

```
session_start();
$main_dishes = array('cuke' => 'Braised Sea Cucumber',
                     'stomach' => "Sauteed Pig's Stomach",
                     'tripe' => 'Sauteed Tripe with Wine Sauce',
                     'taro' => 'Stewed Pork with Taro',
                     'giblets' => 'Baked Giblets with Salt',
                     'abalone' =>
                         'Abalone with Marrow and Duck Feet');
if (isset($_SESSION['order']) &&
          (count($_SESSION['order']) > 0)) {
    print '<ul>';
    foreach ($_SESSION['order'] as $order) {
        $dish_name = $main_dishes[ $order['dish'] ];
        print "<li> $order[quantity] of $dish_name </li>";
    }
    print "</ul>";
} else {
    print "You haven't ordered anything.";
}
```

В программе из примера 10.12 осуществляется доступ к информации, сохраненной в сеансе программой из примера 10.11. В этой программе элемент массива `$_SESSION['order']` трактуется как отдельный массив. Если в этом массиве имеются элементы, о чем свидетельствует возврат положительного числового значения из функции `count()`, то он перебирается с цикле `foreach()`, где выводится также элемент списка для каждого заказанного блюда.

Конфигурирование сеансов

Сеансы вполне пригодны для дополнительной настройки. Достаточно активизировать сеанс с помощью функции `session_start()` или директивы `session.auto_start` конфигурации сервера, чтобы получить массив `$_SESSION` в свое распоряжение. Но если требуется уточнить порядок

функционирования сеансов, то для этой цели имеется ряд полезных параметров настройки, которые можно изменить.

Данные сеанса сохраняются, если доступ к ним производится хотя бы один раз в течение каждых 24 минут, что вполне приемлемо для большинства приложений. Сеансы не предназначены в качестве постоянных хранилищ пользовательской информации, поскольку для этого имеются базы данных. Напротив, сеансы служат для отслеживания последних действий пользователя с целью сделать более удобным просмотр содержимого веб-страниц.

Но в некоторых случаях требуется сократить продолжительность сеансов. Так, если разрабатывается финансовое приложение, то время его простоя не должно превышать 5-10 минут, чтобы необслуживаемым компьютером не смогло воспользоваться неуполномоченное лицо. А если приложение не предназначено для обработки ответственных данных, то можно продлить существование сеансов свыше 24 минут, чтобы не потерять его пользователей.

Директива `session.gc_maxlifetime` конфигурации определяет допустимое время простоя между запросами, в течение которого сеанс сохраняется активным. По умолчанию в ней устанавливается значение **`1440`**, поскольку в 24 минутах содержится 1440 секунд. Чтобы изменить это время, можно внести соответствующие коррективы в директиву `session.gc_maxlifetime` конфигурации или вызвать функцию `ini_set()` из программы на PHP. Так, если решено воспользоваться функцией `ini_set()`, ее следует вызвать прежде функции `session_start()`. В примере 10.13 показано, как воспользоваться функцией `ini_set()`, чтобы изменить на 10 минут допустимое время простоя для сеансов.

Пример 10.13. Изменение допустимого времени простоя для сеансов

```
ini_set('session.gc_maxlifetime',600); // 600 секунд == 10 минут
session_start();
```

Сеансы фактически стираются по истечении 24 минут, и происходит это следующим образом. Вначале обработки любого запроса, в котором используются сеансы (вследствие вызова функции `session_start()` на странице или установки значения **`On`** в директиве конфигурации `session.auto_start`), существует лишь 1 из 100 шансов, что интерпретатор PHP просмотрит все сеансы на сервере и удалит из них истекшие. Такая вероятность данного события свидетельствует о его явной непредсказуемости для компьютерной программы. Тем не менее она повышает эффективность вычислений. Ведь на поиски истекших сеансов с целью удалить их вначале обработки каждого запроса потребовалось бы слишком много вычислительных ресурсов сервера.

Величина 1% вероятности (т.е. 1 из 100 шансов) не годится, если истекшие сеансы требуется удалить быстрее вначале обработки запроса. Эту величину можно изменить с помощью директивы `session.gc_probability`. В частности, чтобы это происходило при каждом запросе, в данной директиве следует установить значение **`100`**. С другой стороны, чтобы изменить значение в директиве `session.gc_probability`, можно воспользоваться функцией `ini_set()`, которую следует вызвать прежде функции `session_start()`. В примере 10.14 показано, как изменить значение в директиве конфигурации `session.gc_probability` с помощью `ini_set()`.

Пример 10.14. Изменение вероятности очистки истекших сеансов

```
ini_set('session.gc_probability',100); // значение 100% означает
          // полную очистку истекших сеансов при каждом запросе
session_start();
```

Если сеансы активизируются с помощью директивы `session.auto_start` конфигурации сервера и требуется изменить значение в директиве `session.gc_maxlifetime` или `session.gc_probability`, то сделать это с помощью функции `ini_set()` нельзя. Для этого придется внести коррективы непосредственно в конфигурацию сервера.

У cookie-файла, применяемого для хранения идентификатора пользовательского сеанса, имеются свои свойства, корректируемые в том числе и с помощью параметров конфигурации. Корректируемые

свойства отражают настройки, которые можно производить в обычном cookie-файле посредством различных аргументов функции `setcookie()`, за исключением, разумеется, значений, хранящихся в cookie-файле. В табл. 10.1 перечислены различные параметры конфигурации сеансов в cookie-файле.

Таблица 10.1. Параметры конфигурации сеансов в cookie-файле

Параметр конфигурации	Значение по умолчанию	Описание
`session.name`	`PHPSESSID`	Имя cookie-файла, где допускается указывать буквы и цифры, причем не меньше одной буквы
`session.cookie_lifetime`	`0`	Количество секунд, которые должны пройти после 1 января 1970 года для истечения срока действия cookie-файла, где нулевое значение обозначает срок действия cookie-файла до тех пор, пока существует браузер
`session.cookie_path`	`/`	Путевой префикс в URL, пригодный для отправки cookie-файла
`session.cookie_domain`	Отсутствует	Суффикс домена, пригодный для отправки cookie-файла. Отсутствие значения означает, что cookie-файл отправляется обратно только по полностью указанному имени хоста
`session.cookie_secure`	`Off`	Чтобы отправить cookie-файл по URL через соединение по сетевому протоколу HTTPS, следует установить значение `On` данного параметра
`session.cookie_httponly`	`Off`	Чтобы дать браузеру команду предотвратить чтение cookie-файла из сценария JavaScript, следует установить значение `On` данного параметра

Регистрация и идентификация пользователей

Сеанс устанавливает анонимное отношение с конкретным пользователем. Требование для пользователей регистрироваться на веб-сайте позволяет им идентифицировать себя. Процесс регистрации, как правило, требует от пользователей предоставить следующие два фрагмента информации: один — для их идентификации (имя пользователя или адрес электронной почты), а другой — для подтверждения пользователями своей личности (секретный пароль). Как только пользователь будет зарегистрирован, он может получить доступ к закрытым данным, передать сообщение для публикации на форуме или сделать еще что-нибудь такое, что не разрешено широкому кругу лиц.

Добавление данных регистрации пользователя к сеансам происходит в течение следующих пяти этапов.

1. Отображение формы, в которой запрашивается имя пользователя и пароль.
2. Проверка переданной на обработку формы.
3. Ввод имени пользователя в сеанс, если передан правильный пароль.
4. Поиск имени пользователя в сеансе для выполнения задач, характерных для данного пользователя.
5. Удаление имени пользователя из сеанса при снятии пользователя с регистрации.

На первых трех этапах обработка данных происходит в контексте обработки обычной формы. Функция `validate_form()` берет на себя ответственность за проверку достоверности предоставляемого имени пользователя и пароля, а функция `process_form()` вводит имя пользователя в сеанс. Так, в примере 10.15 демонстрируется отображение формы регистрации и ввод имени пользователя, если регистрация пройдет успешно.

Пример 10.15. Отображение формы, регистрации

```
require 'FormHelper.php';

session_start();
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        process_form($input);
    }
} else {
    show_form();
}

function show_form($errors = array()) {
    // Собственные значения, устанавливаемые по умолчанию,
    // отсутствуют, поэтому и нечего передавать конструктору
    // класса FormHelper
    $form = new FormHelper();

    // построить HTML-таблицу из сообщений об ошибках для
    // последующего применения
    if ($errors) {
        $errorHtml = '<ul><li>';
        $errorHtml .= implode('</li><li>',$errors);
        $errorHtml .= '</li></ul>';
    } else {
        $errorHtml = '';
    }

    // Это небольшая форма, поэтому ниже выводятся ее составляющие
    print <<<_FORM_
      <form method="POST" action=
                  "{$form->encode($_SERVER['PHP_SELF'])}">
       $errorHtml
       Username: {$form->input('text',
                  ['name' => 'username'])} <br/>
       Password: {$form->input('password',
                  ['name' => 'password'])} <br/>
       {$form->input('submit', ['value' => 'Log In'])}
     </form>
     _FORM_;
}

function validate_form() {
```

```
    $input = array();
    $errors = array();
    // Некоторые образцы имен пользователей и паролей
   $users = array('alice' => 'dog123',
                  'bob' => 'my^pwd',
                  'charlie' => '**fun**');

    // убедиться в достоверности имени пользователя
    $input['username'] = $_POST['username'] ?? '';
    if (! array_key_exists($input['username'], $users)) {
        $errors[] = 'Please enter a valid username and password.';
    }
    // Оператор else означает, что проверка пароля пропускается,
    // если введено недостоверное имя пользователя
    else {
        // проверить правильность введенного пароля
        $saved_password = $users[ $input['username'] ];
        $submitted_password = $_POST['password'] ?? '';
        if ($saved_password != $submitted_password) {
            $errors[] =
                 'Please enter a valid username and password.';
        }
    }
    return array($errors, $input);
}

function process_form($input) {
    // ввести имя пользователя в сеанс
    $_SESSION['username'] = $input['username'];
    print "Welcome, $_SESSION[username]";
}
?>
```

На рис. 10.3 приведена форма, отображаемая в коде из примера 10.15, на рис. 10.4 показано, что происходит, если введен неверный пароль, а на рис. 10.5 — если введен правильный пароль.

Рис. 10.3: Форма регистрации

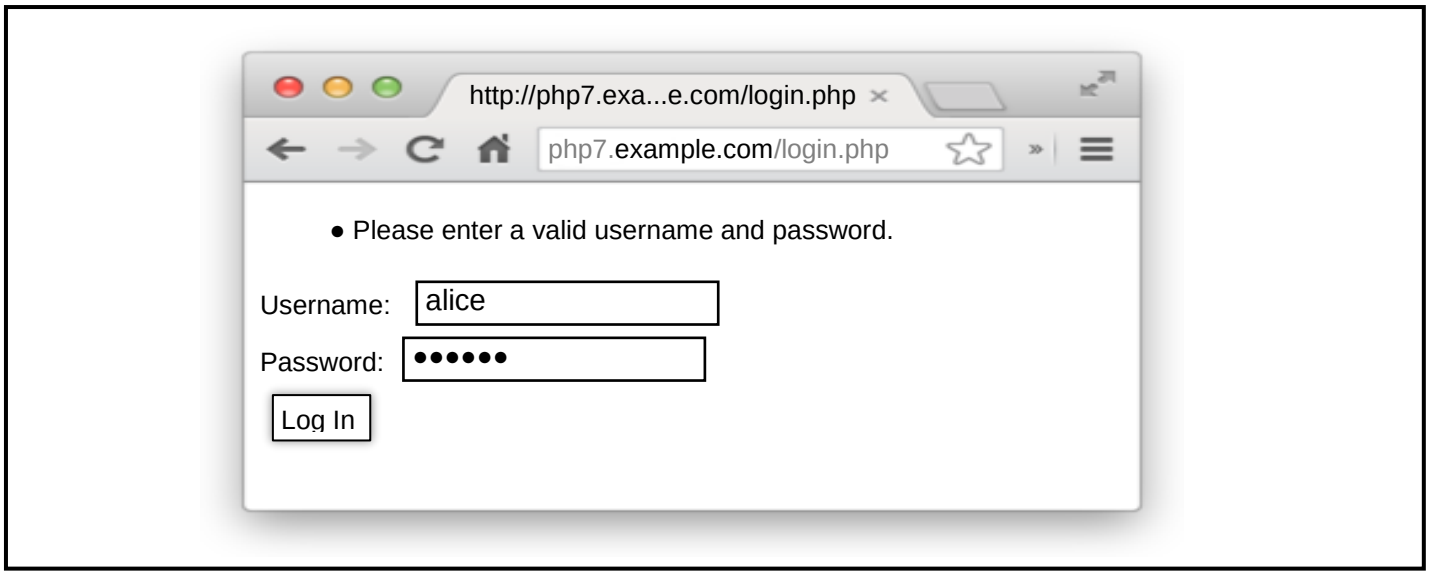


Рис. 10.4: Неудачная регистрация

Рис. 10.5: Удачная регистрация

В функции `validate_form()` из примера 10.15 проверяется достоверность введенного имени пользователя и правильность предъявленного им пароля. Но в любом случае в массив `$errors` вводится одно и то же сообщение об ошибке. Если же выбрать для неверного имени пользователя и неправильного пароля разные сообщения об ошибках (например, “Имя пользователя не найдено” и “Пароль не подходит”), то в них могут найти полезную информацию те, кто пытается разгадать имя пользователя и пароль. Как только злоумышленник, атакующий веб-сайт, споткнется о достоверное имя пользователя и получит сообщение “Пароль не подходит” вместо сообщения “Имя пользователя не найдено”, он будет знать, что оперирует настоящим именем пользователя, и ему остается лишь разгадать пароль. А если сообщения об ошибках одинаковы в обоих случаях, то атакующий злоумышленник не будет знать, что именно неверно в выбранной им комбинации имени пользователя и пароля.

Если предъявлены достоверное имя пользователя и правильный пароль, функция `validate_form()` не возвратит ошибки. И тогда вызывается функция `process_form()`, которая вводит в сеанс предъявленное имя пользователя из элемента массива `($input['username'])`, а также выводит на экран приветственное сообщение для пользователя. Благодаря этому имя пользователя оказывается доступным для применения на других страницах в текущем сеансе. В примере 10.16 демонстрируется порядок проверки имени пользователя на другой странице в текущем сеансе.

Пример 10.16. Выполнение специальных действий для зарегистрированного пользователя

```
<?php
session_start();
if (array_key_exists('username', $_SESSION)) {
    print "Hello, $_SESSION['username'].
} else {
    print 'Howdy, stranger.';
}
?>
```

Элемент username может быть введен в массив `$_SESSION` только программным способом. И если он присутствует, значит, пользователь успешно зарегистрировался.

В функции `validate_form()` из примера 10.15 используется образцовый массив имен пользователей и паролей под названием `$users`. Но хранить пароли в нехешированном виде не рекомендуется. Если список нехешированных паролей окажется раскрытым, атакующий злоумышленник может зарегистрироваться как любой пользователь. А если хранить пароли в хешированном виде, то атакующий злоумышленник не сможет раскрыть настоящие пароли, даже если он доберется до

списка хешированных паролей, поскольку ему вряд ли удастся восстановить хешированный пароль для последующей регистрации. Именно такой способ применяется в тех операционных системах, в которые требуется входить с паролем.

Более совершенный вариант функции `validate_form()` демонстрируется в примере 10.17. Массив `$users` в этом варианте содержит пароли, хешированные с помощью встроенной в PHP функции `password_hash()`. А поскольку пароли хранятся в виде хешированных символьных строк, то их нельзя сравнивать непосредственно с паролем, введенным пользователем в незашифрованном виде. Вместо этого предъявленный пароль, содержащийся в элементе массива `$input ['password']`, проверяется с помощью функции `password_verify()`. В этой функции используется информация из сохраненного в хешированном виде пароля для получения такого же хеш-кода предъявленного пароля. Если оба хеш-кода совпадают, значит, пользователь предъявил правильный пароль, и тогда функция `password_verify()` возвратит логическое значение `true`.

Пример 10.17. Применение хешированных паролей

```
function validate_form() {
    $input = array();
    $errors = array();

    // Образец имен пользователей с хешированными паролями
    $users = array('alice' =>
      '$2y$10$N47IXmT8C.sKUFXs1EBS9uJRuVV8bWxwqubcvNqYP9vcFmlSWEAbq',
                   'bob' =>
      '$2y$10$qCczYRc7S011VRESMqUkGeWQT4V4OQ2qkSyhnxO0c.fk.LulKwUwW',
                   'Charlie' =>
      '$2y$10$nKfkdviOBONrzZkRq5pAgOCbaTFiFI6O2xFka9yzXpEBRAXMW5mYi');

    // проверить достоверность имени пользователя
    if (! array_key_exists($_POST['username'], $users)) {
        $errors[ ] = 'Please enter a valid username and password.';
    }
    else {
        // проверить правильность пароля
        $saved_password = $users[ $input['username'] ];
        $submitted_password = $_POST['password'] ?? '';
        if (! password_verify($submitted_password,
                              $saved_password)) {
            $errors[ ] =
                   'Please enter a valid username and password.';
        }
    }

    return array($errors, $input);
}
```

Применение функций `password_hash()` и `password_verify()` гарантирует, что пароли хешируются достаточно надежным способом, и дает возможность усилить хеширование в дальнейшем по мере надобности. Если вы хотите узнать подробнее о том, как действуют эти функции, обратитесь за справкой к страницам документации на них по адресу `http://www.php.net/password_hash` и `http://www.php.net/password_verify` соответственно в оперативно доступном руководстве PHP или к рецепту 18.7, который дается в книге *PHP Cookbook* Дэвида Скляра (David Sklar) и Адама Трахтенберга (Adam Trachtenberg; издательство O'Reilly; 3-е издание этой книги в русском переводе вышло под названием *PHP. Рецепты программирования* в издательстве "Питер", 2015 г.).

Функции `password_hash()` и `password_verify()` появились в версии PHP 5.5.0. Если вы пользуетесь более ранней версией PHP, обратитесь к библиотеке `password_compat` (по адресу `https://github.com/ircmaxell/password_compat`), в которой предоставляются аналоги этих функций.

Благодаря размещению имен пользователей и паролей в функции `validate_form()` программы в приведенных выше примерах оказываются автономными. Но зачастую имена пользователей и пароли хранятся в таблице базы данных. Так, в примере 10.18 демонстрируется версия функции `validate_form()`, где имя пользователя и хешированный пароль извлекаются из базы данных. При этом предполагается, что подключение к базе данных уже осуществлено вне данной функции и доступно в глобальной переменной `$db`.

Пример 10.18. Извлечение имени пользователя и пароля из базы данных

```
function validate_form() {
    global $db;
    $input = array();
    $errors = array();

    // В этой переменной устанавливается логическое значение true
    // только в том случае, если предъявленный пароль подходит
    $password_ok = false;

    $input['username'] = $_POST['username'] ?? '';
    $submitted_password = $_POST['password'] ?? '';

    $stmt = $db->prepare('SELECT password FROM users
                          WHERE username = ?');

    $stmt->execute($input['username']);
    $row = $stmt->fetch();
    // Если в таблице отсутствует искомая строка, имя
    // пользователя не найдено ни в одной из строк таблицы
    if ($row) {
        $password_ok = password_verify($submitted_password,
                                       $row[0]);
    }
    if (! $password_ok) {
        $errors[] = 'Please enter a valid username and password.';
    }

    return array($errors, $input);
}
```

По запросу, составляемому функцией `prepare()` и посылаемому функцией `execute()` в базу данных, возвращается хешированный пароль пользователя с идентифицированным именем в элементе массива `$input['username']`. Если предъявленное имя пользователя не найдено ни в одной из строк таблицы базы данных, то в переменной `$row` устанавливается логическое значение `false`. А если строка с таким именем пользователя найдена, то в функции `password_verify()` проверяется предъявленный пароль относительно хешированного пароля, извлеченного из базы данных. И

только если в возвращаемой строке таблицы содержится правильный хешированный пароль, в переменной `$password_ok` устанавливается логическое значение true. В противном случае сообщение об ошибке вводится в массив `$errors`.

Чтобы удалить ключ и значение из массива `$_SESSION`, как из любого другого массива, следует воспользоваться функцией `unset()`. Именно таким образом пользователь снимается с регистрации. В примере 10.19 демонстрируется снятие с регистрации на странице.

Пример 10.19. Снятие с регистрации

```
session_start();
unset($_SESSION['username']);

print 'Bye-bye.';
```

В конце обработки запроса, когда вызывается функция `unset()` и сохраняется массив `$_SESSION`, элемент `username` не включается в сохраняемые данные. А при последующей загрузке данных сеанса в массив `$_SESSION` элемент `username` в нем отсутствует, и пользователь снова становится анонимным.

Причины для размещения вызовов функций `setcookie()` `session_start()` вначале страницы

Когда веб-сервер посылает ответ веб-клиенту, большую часть этого ответа составляет HTML-документ, воспроизводимый браузером на веб-странице, отображаемой на экране и состоящей из совокупности дескрипторов и фрагментов текста, которые браузер Safari или Firefox форматирует в таблицы, изменяет их цвет или размер. Но в таком ответе HTML-документу предшествуют *заголовки*, которые не отображаются на экране, но состоят из команд или информации, передаваемой из сервера веб-клиенту. В этих заголовках сообщается, в частности, о том, что данная страница была сформирована в такой-то момент, ее не следует кешировать, а если она подходит, то нужно запомнить cookie-файл под именем `userid` и со значением `ralph`.

Все заголовки должны непременно располагаться вначале ответа, посылаемого из сервера веб-клиенту, предшествуя *телу ответа*, которое составляет HTML-разметка, управляющая тем, что браузер фактически отображает. Как только тело ответа будет отправлено (даже в виде единственной строки), никаких заголовков больше отправлять нельзя.

Заголовки вводятся в ответ с помощью функций `setcookie()` и `session_start()`. Чтобы заголовки были отправлены надлежащим образом, они должны быть введены в ответ прежде любой выводимой информации. Именно поэтому упомянутые выше функции следует вызывать прежде любых операторов `print` или HTML-разметки, оказывающейся за пределами дескрипторов `<?php ?>` разметки кода PHP.

Если попытаться отправить любые выводимые результаты прежде вызова функции `setcookie()` или `session_start()`, то интерпретатор PHP выдаст сообщение об ошибке, аналогичное следующему:

```
Warning: Cannot modify header information - headers already sent by
(output started at /www/htdocs/catalog.php:2)
in /www/htdocs/catalog.php on line 4

[ Предупреждение: видоизменить информацию в заголовке нельзя —
  заголовки уже отправлены
  (вывод начат по адресу /www/htdocs/catalog.php:2)
  в строке кода 4 из исходного файла /www/htdocs/catalog.php ]
```

Это означает, что в строке 4 кода из исходного файла `catalog.php` вызвана функция, посылающая заголовок. Но прежде в строке 2 кода из того же самого исходного файла уже была выведена какая-то информация.

Если появится сообщение об ошибке, где извещается, что заголовки уже отправлены, тщательно проверьте исходный код своей программы на наличие ошибочного вывода, убедившись в отсутствии операторов `print` перед вызовом функции `setcookie()` или `session_start()`. В частности, ни перед начальным дескриптором `<?php` на странице, ни за пределами дескрипторов `<?php` и `?>` во включаемых или обязательных файлах не должно быть ничего — даже пробелов.

Альтернативой выявлению неуместных пустых строк в исходных файлах может служить *буферизация вывода*, предписывающая интерпретатору PHP задержать отправку любой выводимой информации до тех пор, пока не завершится обработка всего запроса. И только после этого посылаются любые заданные заголовки, а вслед за ними — все обычно выводимые результаты. Чтобы активизировать буферизацию вывода, следует установить значение On в директиве `output_buffering` конфигурации. Веб-клиентам придется подождать несколько дополнительных миллисекунд, чтобы получить содержимое страницы из сервера, но в то же время удастся сэкономить немало времени на выявление в исходном коде всего вывода, происходящего прежде вызова функции `setcookie()` или `session_start()`.

Если буферизация вывода активизирована, операторы `print`, функции манипулирования cookie-файлами и сеансами, HTML-разметку за пределами дескрипторов `<?php` и `?>` и обычный код PHP можно употреблять в любом сочетании и где угодно на странице, не боясь получить сообщение об ошибке, где извещается, что заголовки уже отправлены. Так, программа из примера 10.20 будет работоспособной лишь в том случае, если активизирована буферизация вывода. В противном случае HTML- документ выводится в ней прежде начального дескриптора `<?php`, запускающего отправку заголовков, что помешает надлежащему выполнению функции `setcookie()`.

Пример 10.20. Программа, в которой требуется, чтобы действовала буферизация вывода

```
<html>
<head>Choose Your Site Version</head>
<body>
<?php
setcookie('seen_intro', 1);
?>
<a href="/basic.php">Basic</a>
or
<a href="/advanced.php">Advanced</a>
</body>
</html>
```

Резюме

В этой главе были рассмотрены следующие вопросы.

- Выяснение причин, по которым cookie-файлы необходимы для идентификации конкретного веб-браузера или веб-сервера.
- Установка cookie-файла в программе на PHP.
- Чтение значения из cookie-файла в программе на PHP.
- Видоизменение таких параметров cookie-файла, как время истечения срока действия, путь и домен.

- Удаление cookie-файла в программе на PHP.
- Активизация сеансов из программы на PHP или в конфигурации интерпретатора PHP.
- Сохранение информации в сеансе.
- Чтение информации из сеанса.
- Сохранение данных из переданной на обработку формы в сеансе.
- Удаление данных из сеанса.
- Настройка срока действия и очистки сеанса.
- Отображение, проверка достоверности и обработка формы регистрации.
- Применение хешированных паролей.
- Выяснение причин, по которым функции `setcookie()` и `session_start()` должны вызываться прежде вывода любых результатов.

Упражнения

1. Создайте веб-страницу, на которой cookie-файл используется для слежения за количеством ее просмотров отдельным пользователем. Когда отдельный пользователь просматривает страницу в первый раз, на странице должно быть выведено сообщение вроде `"Number of views: 1"` (Количество просмотров: 1), а когда данный пользователь просматривает страницу во второй раз — сообщение `"Number of views: 1"` ит.д.
2. Видоизмените веб-страницу из первого упражнения таким образом, чтобы на ней выводилось специальное сообщение при 5-м, 10-м и 15-м просмотре страницы, а после 20-го просмотра должен быть удален cookie-файл и подсчет просмотров страницы начат с самого начала.
3. Напишите на PHP программу, отображающую форму, где пользователь может выбрать излюбленный цвет из списка. Создайте еще одну страницу с цветом фона, выбранным пользователем в данной форме. Сохраните значение этого цвета в массиве `$_SESSION`, чтобы сделать его доступным на обеих страницах.
4. Напишите на PHP программу, отображающую форму заказа, где перечисляются шесть товаров. Рядом с названием каждого товара должно находиться текстовое поле, в котором пользователь может ввести количество заказываемого товара. После передачи формы на обработку данные из нее должны быть сохранены в сеансе. Создайте еще одну страницу, отображающую содержимое сохраненного заказа, обратную ссылку на страницу с формой заказа, а также кнопку Check Out (Оформить заказ). Если щелкнуть на обратной ссылке, должна появиться страница с формой заказа, в текстовых полях которой должны быть указаны сохраненные количества заказанных товаров. А если щелкнуть на кнопке Check Out, то заказ должен быть удален из текущего сеанса.

ГЛАВА 11

Взаимодействие с другими веб-сайтами и веб-службами

В предыдущих главах обсуждались такие внешние источники данных, как базы данных и файлы. А в этой главе речь пойдет о еще одном важном внешнем источнике данных — других веб-сайтах. Программы на PHP зачастую служат в качестве клиентов для других сайтов или прикладных программных интерфейсов API, предоставляющих нужные данные. Один веб-сайт может и сам поставлять данные, которые требуются другому сайту. И в этой главе будет показано, как извлекать данные из внешних источников по заданным URL и получать доступ к прикладным программным интерфейсам API. В ней также поясняется, что именно требуется для обслуживания запросов прикладных программных интерфейсов API, поступающих из других сайтов.

В первой части этой главы речь пойдет о том, как пользоваться встроенными в PHP функциями для доступа к файлам, указывая URL вместо имен файлов. Это удобно для быстрого и простого доступа к удаленным веб-ресурсам по заданному URL. А для еще большего удобства и эффективности служит расширение cURL языка PHP, осуждаемое в разделе "Универсальный доступ по URL с помощью расширения cURL". Функции из расширения cURL позволяют управлять многими свойствами выполняемых запросов.

Обслуживанию запросов прикладных программных интерфейсов API, вместо веб-страниц в программе на PHP, посвящен раздел "Обслуживание запросов API". Ответы на подобные запросы аналогичны стандартным HTML-страницам, но у них имеется ряд важных отличий.

Простой доступ по URL с помощью функций манипулирования файлами

Функции доступа к файлам вроде `file_get_contents()` особенно удобны тем, что они распознают не только имена файлов, но и URL. Чтобы, например, извлечь содержимое из удаленного веб-ресурса и разместить его в символьной строке, достаточно передать URL этого веб-ресурса функции `file_get_contents()`. В примере 11.1 демонстрируется применение функции `file_get_contents()` для отображения интересного факта, связанного с датой 27 сентября, из веб-сайта по адресу `numbersapi.com`.

Пример 11.1. Извлечение веб-содержимого по заданному URL с помощью функции ***`file_get_contents()`***

```
Did you know that
        <?= file_get_contents('http://numbersapi.com/09/27') ?>
```

В прикладном программном интерфейсе Numbers API содержится немало интересных фактов на каждый день года, но в примере 11.1 на экран выводится такой интересный факт:

```
Did you know that September 27th is the day in 1961 that
Sierra Leone joins the United Nations.

[ Знаете ли вы, что 27 сентября 1961 года Сьерра-Леона вступила в ООН ]
```

Функция `http_build_query()` оказывается удобной при построении API URL, содержащего параметры строки запроса. Так, если предоставить функции `http_build_query()` ассоциативный массив имен и значений параметров, она возвратит символьную строку, состоящую из пар "ключ-значение", соединяемых знаком **&** и закодированных надлежащим образом, т.е. именно то, что и требуется для построения URL.

На веб-сайте Министерства земледелия США имеется изящный прикладной программный интерфейс API, надстроенный над национальной базой данных продуктов питания (NDB). Прикладной программный интерфейс NDB API свободно доступен и прост в употреблении.

Прикладной программный интерфейс NDB API, применяемый в примерах кода из этой главы, требует, чтобы запросы содержали параметр `api_key`, значением которого является отдельный ключ, получаемый при подписке на данный интерфейс. Чтобы получить свой ключ API, перейдите по адресу `https://api.data.gov/signup/`. Этот ключ предоставляется бесплатно и быстро, для чего достаточно указать минимум личных сведений: Ф.И.О. и адрес электронной почты.

В примерах кода из этой главы употребляется константа `NDB_API_KEY` вместо конкретного ключа NDB API. Чтобы выполнить код из этих примеров, подставьте вместо константы `NDB_API_KEY` символьную строку, содержащую ваш ключ NDB API, или же вызовите функцию `define()`, устанавливающую значение константы `NDB_API_KEY` равным значению вашего ключа NDB API. Так, если у вас имеется ключ **273bqhebrfkhuebf**, введите следующую строку в самом начале своего исходного кода:

```
define('NDB_API_KEY','273bqhebrfkhuebf');
```

В примере 11.2 демонстрируется применение прикладного программного интерфейса API для поиска в базе данных NDB сведений о черном перце. Этот прикладной программный интерфейс API возвращает из базы данных NDB сведения о каждом продукте питания, наименование которого совпадает с тем, что указано в параметре строки запроса `q`.

Пример 11.2. Ввод параметров строки запроса в API URL

```
$params = array('api_key' => NDB_API_KEY,
                'q' => 'black pepper',
                'format' => 'json');

$url = "http://api.nal.usda.gov/ndb/search?" . http_build_query($params);
```

Переменной `$url` в коде из примера 11.2 присваивается значение, аналогичное приведенному ниже. Конкретное ее значение зависит от указанного ключа NDB API.

```
http://api.nal.usda.gov/ndb/search?.api_key=j724nbefuy72n4&q=black+pepper&format=json
```

Каждый ключ из массива `$params` соединен в строке запроса со своим значением с помощью знаков **=** и **&**, а специальные символы (например, пробел в наименовании продукта `black pepper`) закодированы. Если передать сформированный таким образом URL функции `file_get_contents()`, то произойдет обращение к прикладному программному интерфейсу NDB API. В данном случае интерфейс NDB API возвратит данные в формате JSON, а следовательно, функция `file_get_contents()` возвратит следующую символьную строку:

```
{
    "list": {
        "q": "black pepper",
        "sr": "27",
        "start": 0,
        "end": 1,
        "total": 1,
        "group": "",
        "sort": "r",
        "item": [
            {
                "offset": 0,
                "group": "Spices and Herbs",
                "name": "Spices, pepper, black",
                "ndbno": "02030"
            }
        ]
    }
}
```

В связи с тем что функция `file_get_contents()` возвращает информацию, излеченную по заданному URL из базы данных NDB, в виде символьной строки, эту строку проще всего передать другим функциям для последующего преобразования. Так, если передать приведенный выше ответ из NDB API функции `json_decode()`, он будет преобразован из формата JSON в структуру данных, которой можно манипулировать в коде PHP. В примере 11.3 демонстрируется вывод на экран идентификатора каждого продукта питания, обнаруженного в базе данных NDB.

Пример 11.3. Декодирование ответа из NDB API

```
$params = array('api_key' => NDB_API_KEY,
                'q' => 'black pepper',
                'format' => 'json');

$url = "http://api.nal.usda.gov/ndb/search?". http_build_query($params);
$response = file_get_contents($url);
$info = json_decode($response);

foreach ($info->list->item as $item) {
    print "The ndbno for {$item->name} is {$item->ndbno}.\n";
}
```

Функция `json_decode()` преобразует объекты и массивы из формата JSON в формат PHP. Элементом верхнего уровня в ответе является объект. Именно он и возвращается функцией `json_decode()`, а затем присваивается переменной `$info`. Этот объект содержит свойство `list`, которое представляет другой объект. К объекту `list` можно обращаться как к свойству по ссылке `$info->list`. В свою очередь, объект `list` содержит свойство массива под названием `item`, а

элементы данного массива — подробные сведения о найденных продуктах питания. Таким образом, в цикле `foreach()` перебирается массив `$info->list->item`. На каждом шаге этого цикла переменной `$item` присваивается один объект из данного массива. При выполнении кода из примера 11.3 на экран выводится следующий результат:

```
The ndbno for Spices, pepper, black is 02030.
```

В приведенных до сих пор примерах при обращении к прикладному программному интерфейсу NDB API данные возвращались в формате JSON, поскольку в URL был указан параметр строки запроса `format=json`. Но прикладному интерфейсу NDB API можно также указать формат ответа, послав заголовок `Content-Type`.[1] Так, если указать в этом заголовке значение **`application/json`**, сервер возвратит ответ в формате JSON.

Чтобы ввести заголовки в HTTP-запрос с помощью функции `file_get_contents()`, следует создать *потоковый контекст*. Для ввода и вывода данных из программ в механизме PHP предусмотрены базовые средства, называемые потоками. В качестве потока может служить локальный файл, удаленный веб-ресурс, доступный по заданному URL, или другое необычное место, поставляющее и потребляющее данные. При вызове функции `file_get_contents()` в качестве первого аргумента ей передается цель потока: файл или URL для чтения или записи данных. А дополнительные сведения об операции чтения или записи выражаются через *потоковый контекст*, для создания которого достаточно передать ассоциативный массив с дополнительными сведениями функции `stream_context_create()`.

В различных видах потоков поддерживаются разные параметры их контекста. Так, для потока `http` предусмотрен параметр `header`, получающий строковое значение, содержащее имена и значения из любых заголовков, посылаемых в HTTP- запросе. В примере 11.4 демонстрируется порядок создания потокового контекста, включая HTTP-заголовок, а также его применения в функции `file_get_contents()`.

Пример 11.4. Отправка HTTP-заголовков с потоковым контекстом

```
// В следующей строке запроса указан только ключ и параметр
// запроса, но никакого формата
$params = array('api_key' => NDB_API_KEY,
                'q' => 'black pepper');
$url = "http://api.nal.usda.gov/ndb/search?"
        . http_build_query($params);

// Параметры для установки заголовка Content-Type в HTTP-запросе
$options = array('header' => 'Content-Type: application/json');
// Создать контекст для потока 'http'
$context = stream_context_create(array('http' => $options));

// передать потоковый контекст в качестве аргумента
// функции file_get_contents()
print file_get_contents($url, false, $context);
```

В коде из примера 11.4 массив `$options` содержит пары "ключ-значение" устанавливаемых параметров. Функции `stream_context_create()` требуется указать вид потока, для которого она должна создать контекст, и поэтому в качестве аргумента данной функции передается массив, где ключом является тип потока (`http`), а значением — устанавливаемые параметры.

В качестве второго аргумента функции `file_get_contents()` указывается, должна ли она обращать внимание на путь, включаемый интерпретатором PHP, при поиске файла. А поскольку

[1] В спецификации сетевого протокола HTTP для этой цели предусмотрен заголовок `Accept`, но он не пригоден для работы с данным прикладным интерфейсом API.

это не соответствует сетевому протоколу HTTP, то в качестве второго аргумента данной функции следует указать логическое значение `false`. И, наконец, в качестве третьего аргумента функции `file_get_contents()` передается потоковый контекст.

Потоковый контекст определяет также порядок отправки запроса по методу POST с помощью функции `file_get_contents()`. Параметр контекста `method` определяет метод отправки запроса, а параметр контекста `content` — содержимое тела любого посылаемого запроса. Это содержимое должно быть отформатировано в соответствии с типом содержимого, указанным в качестве заголовка.

Пользоваться методом `POST` для отправки запросов прикладному программному интерфейсу NDB API нельзя, потому что он сообщает, пользуясь методом `GET`, лишь сведения о запрашиваемом продукте питания, но не разрешает посылать ему новые данные. В примере 11.5 демонстрируется применение функции `file_get_contents()` для отправки запроса методом `POST` по URL, указанному ради примера. Такой запрос действует аналогично передаче на обработку формы с двумя переменными: `name` и `smell`.

Пример 11.5. Отправка запроса методом POST с помощью функции `file_get_contents()`

```
$url = 'http://php7.example.com/post-server.php';

// Две переменные запроса, посылаемого методом POST
$form_data = array('name' => 'black pepper',
'smell' => 'good');

// задать метод, тип содержимого и само содержимое
$options = array('method' => 'POST',
                 'header' =>'Content-Type:
                             application/x-www-form-urlencoded',
                 'content' => http_build_query($form_data));

// создать контекст для потока типа 'http'
$context = stream_context_create(array('http' => $options));

// передать контекст в качестве третьего аргумента
// функции file_get_contents()
print file_get_contents($url, false, $context);
```

В примере 11.5 параметр потокового контекста `method` гарантирует, что запрос отправляется методом `POST`. Его значение используется буквально при выполнении запроса в интерпретаторе PHP, и поэтому его следует указывать прописными буквами. Для заголовка `Content-Type` указывается стандартное значение, применяемое веб-браузерами для представления данных из обычной формы. Оно соответствует данным, отформатированным подобно параметрам в строке запроса, но посылается в теле запроса. И это удобно, поскольку дает возможность воспользоваться функцией `http_build_query()` для составления отформатированного надлежащим образом тела запроса.

В примере 11.5, как, впрочем, и в других примерах из этого раздела, употребляется вымышленное имя хоста `php7.example.com`. Чтобы сделать код из этого и других примеров работоспособным, данное имя хоста следует заменить настоящим (желательно, адресом своего веб-сервера).

Если в запросе методом POST требуется отправить данные другого типа, достаточно изменить значение в заголовке Content-Type и порядок форматирования содержимого запроса. Например, чтобы отправить данные в формате JSON, следует изменить значение параметра header на **Content-Type:application/json**, а значение параметра content — на **json_encode ($form_data)**. Подробнее о различных типах потоков, предусмотренных в интерпретаторе PHP, а также о других поддерживаемых параметрах потокового контекста см. по адресу http://www.php.net/context.

Несмотря на всю простоту извлечения веб-содержимого по заданным URL с помощью встроенных в PHP функций, эти функции все же не упрощают дело, если при выполнении запроса возникает ошибка. В таком случае функция file_get_contents() возвращает логическое значение false, а интерпретатор PHP формирует сообщение об ошибке вроде следующего:

```
failed to open stream: HTTP request failed! HTTP/1.1 404 Not Found

[ не удалось открыть поток: выполнить HTTP-запрос не удалось!
 HTTP/1.1 Ошибка 404 "Документ не найден" ]
```

Возможность точнее контролировать ситуацию, когда выполнить запрос не удается, служит одним из веских оснований для обращения к расширению cURL вместо встроенных в PHP функций.

Универсальный доступ по URL с помощью расширения cURL

Функция file_get_contents(), особенно в сочетании с параметрами потокового контекста, позволяет выполнять самые разные HTTP-запросы. Но когда действительно требуется точно контролировать HTTP-запросы и ответы, придется обратиться к функциям из расширения cURL языка PHP. Опираясь на эффективную библиотеку libcurl, эти функции предоставляют доступ ко всем свойствам HTTP-запросов и ответов.

Извлечение данных по заданному URL методом GET

Доступ к веб-ресурсу по заданному URL средствами cURL начинается с передачи URL этого веб-ресурса функции curl_init(). Эта функция выполняется не сразу, извлекая данные по указанному URL. Она возвращает *дескриптор* в виде переменной, которая передается другим функциям для установки параметров и настройки режима работы расширения cURL. В разных переменных может одновременно храниться несколько дескрипторов, причем каждый дескриптор управляет отдельным запросом.

Функция curl_setopt() определяет поведение интерпретатора PHP при извлечении данных по указанному URL, а функция curl_exec() фактически выполняет запрос на их извлечение. В примере 11.6 демонстрируется применение расширения cURL для извлечения данных по URL веб-ресурса numbersapi.com из примера 11.1.

Пример 11.6. Извлечение данных по URL средствами cURL

```
<?php

$c = curl_init('http://numbersapi.com/09/27');
// Дать расширению cURL команду возвратить содержимое ответа
// в виде символьной строки, а не вывести его сразу на экран
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// Выполнить запрос $fact = curl_exec($c);

?>
Did you know that <?= $fact ?>
```

При вызове функции curl_setopt() из примера 11.6 задается параметр CURLOPT_RETURNTRANSFER. Этим расширение cURL извещается о том, что при выполнении HTTP-запроса следует возвратить ответ в виде символьной строки. В противном случае ответ выводится по мере получения. Функция curl_exec() выполняет запрос и возвращает результат его обработки. А другие параметры cURL позволяют устанавливать заголовки. В примере 11.7 демонстрируется применение функций cURL для выполнения запроса из примера 11.4.

Пример 11.7. Применение функций cURL с параметрами из строки запроса и заголовками

```
// В следующей строке запроса указан только ключ и параметр
// запроса, но никакого формата
$params = array('api_key' => NDB_API_KEY,
                'q' => 'black pepper');
$url = "http://api.nal.usda.gov/ndb/search?"
       . http_build_query($params);

$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_HTTPHEADER,
            array('Content-Type: application/json'));
print curl_exec($c);
```

В примере 11.7 URL составляется таким же образом, как и с помощью функции http_build_query(). Параметры из строки запроса являются составной частью URL, поэтому они входят в строку URL, передаваемую функции curl_init(). Новый параметр CURLOPT_HTTP_HEADER задает HTTP-заголовок, посылаемый вместе с запросом. Если же имеется несколько заголовков, их следует разместить в массиве.

Имеются две категории ошибок, которые приходится обрабатывать при выполнении запросов средствами cURL. К первой категории относятся ошибки в самом URL, в том числе отсутствие имени хоста или неспособность установить соединение с удаленным сервером. Если происходят ошибки такого рода, функция curl_exec() возвращает логическое значение false, а функция curl_errno() — код ошибки. Функция curl_error() возвращает сообщение об ошибке, соответствующее ее коду.

Ко второй категории относятся ошибки, поступающие из удаленного сервера. Они происходят в том случае, если веб-ресурс по заданному URL не найден или же если сервер испытывает трудности при формировании ответа на полученный запрос. Тем не менее в расширении cURL такой запрос считается выполненным успешно, поскольку сервер возвратил хотя бы что-нибудь. Поэтому необходимо проверить код HTTP-ответа, чтобы выяснить, содержит ли он ошибку. Функция curl_getinfo() возвращает массив со сведениями о запросе. Один из элементов данного массива содержит код HTTP-ответа. В примере 11.8 демонстрируется код выполнения запроса средствами cURL с обработкой двух рассматриваемых здесь категорий ошибок.

Пример 11.8. Обработка ошибок средствами cURL

```
// Несуществующая на самом деле точка доступа к прикладному
// программному интерфейсу API
$c = curl_init('http://api.example.com');
curl_setopt($c, CORLOPT_RETURNTRANSFER, true);
$result = curl_exec($c);
// получить все сведения о соединении независимо от того,
// было ли оно установлено или не установлено
$info = curl_getinfo($c);
```

```
// установить соединение по какой-то причине не удалось
if ($result === false) {
    print "Error #" . curl_errno($c) . "\n";
    print "Uh-oh! cURL says: " . curl_error($c) . "\n";
}
// Коды HTTP-ответа в пределах от 400 до 500 обозначают ошибки
else if ($info['http_code'] >= 400) {
    print "The server says HTTP error {$info['http_code']}.\n";
}
else {
    print "A successful result!\n";
}
// В сведения о запросе включены также временные
// статистические данные
print "By the way, this request took
       {$info['total_time']} seconds.\n";
```

Код из примера 11.8 начинается со стандартного для cURL запроса. После выполнения запроса сведения о нем, возвращаемые из функции `curl_getinfo()`, сохраняются в переменной `$info`. Функции `curl_getinfo()` необходимо передать дескриптор cURL, которым она должна оперировать, аналогично функции `curl_errno()` или `curl_error()`. Это необходимо для того, чтобы возвратить информацию о правильном запросе.

Хост `api.example.com` на самом деле не существует, и поэтому расширение cURL не может установить с ним соединение, чтобы выполнить запрос. Следовательно, функция `curl_exec()` возвратит логическое значение `false`. При выполнении кода из примера 11.8 на экран выводится приведенный ниже результат. На странице документации на функцию `curl_errno()`, оперативно доступной по адресу `http://www.php.net/curl_errno`, перечисляются все коды ошибок, генерируемых средствами cURL.

```
Error #6
Dh-oh! cURL says: Could not resolve host: api.example.com
By the way, this request took 0.000146 seconds.

[ Ошибка #6
 Ой-ой! cURL сообщает: не удалось разрешить имя хоста:api.example.com
 Кстати, обработка этого запроса отняла 0,000146 с ]
```

Если запрос направляется на сервер, но тот возвращает ошибку, переменная `$result` содержит не логическое значение `false`, а любой ответ, отправленный обратно сервером. Код этого ответа хранится в элементе `http_code` массива `$info`. Так, если при выполнении кода из примера 11.8 возникнет ошибка HTTP 404, которая означает, что серверу не удалось найти указанную в запросе страницу, то на экран будет выведен следующий результат:

```
The server says HTTP error 404.
By the way, this request took 0.00567 seconds.

[ Сервер сообщает об ошибке HTTP 404.
  Кстати, обработка этого запроса отняла 0,00567 с ]
```

В оба результата выполнения кода из примера 11.8 включено общее время выполнения запроса. Это еще один полезный фрагмент данных из массива `$info`. На странице документации на функцию `curl_getinfo()`, оперативно доступной по адресу `http://www.php.net/curl_getinfo`, перечисляются все элементы данного массива.

Извлечение данных по заданному URL методом POST

Чтобы воспользоваться методом POST вместе с расширением cURL, следует настроить параметры с целью изменить метод запроса и снабдить данными тело самого запроса. В частности, параметр CURLOPT_POST сообщает расширению cURL, что требуется выполнить запрос методом POST, а параметр CURLOPT_POSTFIELDS содержит отправляемые данные. В примере 11.9 демонстрируется выполнение запроса методом POST и средствами cURL.

Пример 11.9. Выполнение запроса методом POST и средствами cURL

```
$url = 'http://php7.example.com/post-server.php';

// Две переменные, отправляемые по запросу методом POST
$form_data = array('name' => 'black pepper',
                   'smell' => 'good');

$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// Это должен быть запрос, выполняемый методом POST
curl_setopt($c, CURLOPT_POST, true);
// А это отправляемые данные
curl_setopt($c, CURLOPT_POSTFIELDS, $form_data);

print curl_exec($c);
```

В примере 11.9 не нужно устанавливать заголовок Content-Type или формат отправляемых данных, поскольку расширение cURL делает это автоматически. Но если требуется отправить содержимое другого типа, а не данные из обычной формы, то для этого придется приложить чуть больше усилий. Так, в примере 11.10 показано, как отправить данные формата JSON по запросу, выполняемому методом POST и средствами cURL.

Пример 11.10. Отправка данных формата JSON по запросу, выполняемому методом POST и средствами cURL

```
$url = 'http://php7.example.com/post-server.php';

// Две переменные, отправляемые в формате JSON
// по запросу методом POST
$form_data = array('name' => 'black pepper',
                   'smell' => 'good');

$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// Это должен быть запрос, выполняемый методом POST
curl_setopt($c, CURLOPT_POST, true);
// Это сам запрос, содержащий данные формата JSON
curl_setopt($c, CURLOPT_HTTPHEADER,
            array('Content-Type: application/json'));
// А это соответственно отформатированные отправляемые данные
curl_setopt($c, CURLOPT_POSTFIELDS, json_encode($form_data));

print curl_exec($c);
```

В примере 11.10 параметр CURLOPT_HTTPHEADER сообщает серверу, что тело запроса содержит данные в формате JSON, а не данные обычной формы. Кроме того, параметр CURLOPT_POSTFIELDS принимает значение `json_encode($form_data)`, чтобы тело запроса действительно содержало данные в формате JSON.

Применение cookie-файлов

Если ответ на запрос, выполняемый средствами cURL, включает в себя заголовок, устанавливающий cookie-файл, расширение cURL не делает ничего особенного с этим заголовком по умолчанию. Но в то же время расширение cURL предоставляет ряд параметров настройки, позволяющих отслеживать cookie-файлы, — даже в разных программах на PHP или разных экземплярах выполнения одной и той же программы.

В примере 11.11 показана простая страница, поддерживающая cookie-файл `c`. Всякий раз, когда запрашивается данная страница, в ответ включается cookie-файл `c`, значение в котором на единицу больше, чем значение, предоставляемое для cookie-файла `c` в запросе. Если же cookie-файл `c` не посылается, то при ответе в cookie-файле `c` устанавливается значение **`1`**.

Пример 11.11. Простой способ установки cookie-файла на сервере

```
// использовать значение, отправляемое в cookie-файле,
// или нулевое значение, если cookie-файл не предоставляется
$value = $_COOKIE['c'] ?? 0;
// увеличить значение на 1
$value++;
// установить новый cookie-файл в ответе
setcookie('c', $value);
// сообщить пользователю об обнаруженных cookie-файлах
print "Cookies: " . count($_COOKIE) . "\n";
foreach ($_COOKIE as $k => $v) {
    print "$k: $v\n";
}
```

В отсутствие дополнительной настройки расширение cURL не отслеживает cookie-файл, посылаемый обратно в коде из примера 11.11. А в коде из примера 11.12 функция `curl_exec()` вызывается дважды по одному и тому же дескриптору. Но cookie-файл, посылаемый в ответ на первый запрос, не посылается по второму запросу.

Пример 11.12. Поведение расширения с URL по умолчанию при обработке cookie-файлов

```
// извлечь страницу с cookie-файлами на сервере,
// не посылая обратно никаких cookie-файлов
$c = curl_init('http://php7.example.com/cookie-server.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// В первый раз cookie-файлы отсутствуют
$res = curl_exec($c);
print $res;

// Во второй раз cookie-файлы по-прежнему отсутствуют
$res = curl_exec($c);
print $res;
```

В примере 11.12, как, впрочем, и в остальных примерах, демонстрирующих взаимодействие клиента и сервера в этом разделе, страница с программой на PHP доступна по вымышленному адресу `http://php7.example.com/cookieserver.php`. Чтобы выполнить код из данного и других примеров в этом разделе, укажите в URL свой PHP-сервер.

При выполнении кода из примера 11.12 на экран выводится следующий результат:

```
Cookies: 0
Cookies: 0
```

На оба запроса получается ответ в виде строки `"Cookies: 0"`, поскольку расширение cURL не отправило заголовок `Cookie` вместе с запросом. Если же активизировать *хранилище cookie-файлов* в cURL, это расширение будет отслеживать cookie-файлы. А для того чтобы отслеживать cookie-файлы в течение срока действия конкретного дескриптора cURL, следует установить логическое значение `true` в параметре `CURLOPT_COOKIEJAR` (пример 11.13).

Пример 11.13. Активизация хранилища cookie-файлов в расширении cURL

```
// извлечь страницу с cookie-файлами на сервере,
// не посылая обратно никаких cookie-файлов
$c = curl_init('http://php7.example.com/cookie-server.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// активизировать хранилище cookie-файлов
curl_setopt($c, CURLOPT_COOKIEJAR, true);

// В первый раз cookie-файлы отсутствуют
$res = curl_exec($c);
print $res;

// Во второй раз присутствуют cookie-файлы
// из первого запроса
$res = curl_exec($c);
print $res;
```

При выполнении кода из примера 11.13 на экран выводится следующий результат:

```
Cookies: 0
Cookies: 1
c: 1
```

В примере 11.13 расширение cURL отслеживает cookie-файлы в ответ на запрос, при условии, что дескриптор для данного запроса cURL существует в программе. При вызове функции `curl_exec()` с дескриптором `$c` во второй раз используется cookie-файл, установленный при первом запросе.

В данном режиме cookie-файлы отслеживаются хранилищем cookie-файлов только в пределах дескриптора. Если изменить значение параметра `CURLOPT_COOKIEJAR` на имя файла, расширение cURL должно записывать значения из cookie-файлов в указанный файл. Имя этого файла может быть также предоставлено в качестве значения параметра `CURLOPT_COOKIEFILE`. Прежде чем посылать запрос, расширение cURL читает любые cookie-значения из файла, указанного в параметре `CURLOPT_COOKIEFILE`, используя их в последующих запросах. В примере 11.14 демонстрируется применение хранилища cookie-файлов и файла cookie-значений для их отслеживания.

Пример 11.14. Отслеживание cookie-файлов по запросам

```
// извлечь страницу с cookie-файлами на сервере
$c = curl_init('http://php7.example.com/cookie-server.php');
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
// сохранить cookie-значения в файле 'saved.cookies',
// размещаемом в том же самом каталоге, где и данная программа
curl_setopt ($c, CURLOPT_COOKIEJAR, __DIR__ . '/saved.cookies');
// загрузить cookie-значения из файла 'saved.cookies', находящегося
// в том же самом каталоге, если они вообще были в нем сохранены
curl_setopt($c, CURLOPT_COOKIEFILE, __DIR__ . '/saved.cookies');

// Этот запрос включает в себя cookie-значения из заданного файла,
// если таковой вообще существует
$res = curl_exec($c);
print $res;
```

При выполнении кода из примера 11.14 в первый раз на экран выводится следующий результат:

```
Cookies: 0
```

При выполнении кода из примера 11.14 во второй раз на экран выводится такой результат:

```
Cookies: 1
c: 1
```

При выполнении кода из примера 11.14 в третий раз на экран выводится приведенный ниже результат.

```
Cookies: 1
c: 2
```

И так далее. Всякий раз, когда выполняется программа из данного примера, в ней осуществляется поиск файла **saved.cookies**, загружаются любые cookie-значения, хранящиеся в данном файле, а далее они используются для запроса. После запроса любые cookie-значения сохраняются снова в том же самом файле, поскольку значения параметров `CURLOPT_COOKIEFILE` и `CURLOPT_COOKIEJAR` совпадают. Благодаря этому файл `saved.cookies` обновляется новым значением, чтобы подготовить его к выполнению данной программы в следующий раз.

Если требуется написать программу, имитирующую процесс регистрации пользователя в веб-браузере и последующего выполнения запросов, хранилище cookie-файлов оказывается очень удобным для того, чтобы снабдить отправляемыми cookie-файлами все запросы, выполняемые средствами cURL.

Извлечение данных по HTTPS URL

Чтобы извлечь данные по URL, в которых применяется сетевой протокол HTTPS, достаточно выполнить те же самые действия, пользуясь средствами cURL, что и при извлечении данных по URL, в которых применяется обычный сетевой протокол HTTP. Но помимо этого, необходимо правильно установить некоторые параметры безопасности, чтобы добиться нужного результата. Как правило, для этого достаточно устанавливаемых по умолчанию значений этих параметров. В этом разделе поясняются значения параметров безопасности, чтобы стало понятно, почему их не стоит изменять.

Когда веб-клиент извлекает данные по HTTPS URL, необходимая безопасность обеспечивается двумя средствами. Одним из них является проверка идентичности, в ходе которой сервер подтверждает свою подлинность и способность обрабатывать URL, опираясь на имя хоста. А вторым средством является защита от перехвата информации, когда всякий, перехватывающий обмен данными

между клиентом и сервером, получает в итоге бессмысленный набор символов вместо настоящего запроса и ответа.

Параметры CURLOPT_SSL_VERIFYPEER и CURLOPT_SSL_VERIFYHOST определяют строгость проверки идентичности средствами cURL. Так, если установлено логическое значение false параметра CURLOPT_SSL_VERIFYPEER или же любое значение, кроме **2**, параметра CURLOPT_SSL_VERIFYHOST, расширение пропустит существенные стадии проверки подлинности сервера.

Начиная с версии 7.10, параметр CURLOPT_SSL_VERIFYPEER устанавливается в расширении cURL по умолчанию, а с версии 7.28.1 в расширении cURL не допускается изменять значение 2 параметра CURLOPT_SSL_VERIFYHOST на какое-нибудь другое.

Чтобы выяснить, какая именно версия cURL установлена вместе с PHP, достаточно вызвать функцию curl_version(). Эта функция возвращает ассоциативный массив со сведениями об установленной версии cURL и ее возможностях. В частности, элемент version данного массива содержит версию cURL, на которую опирается интерпретатор PHP.

Имеются также разные версии безопасного сетевого протокола, которыми пользуются веб-клиенты и серверы для реализации HTTPS URL. Выбор конкретной версии для применения в cURL определяется параметром CURLOPT_SSLVERSION. Изменять значение этого параметра не нужно, поскольку значение, устанавливаемое в нем по умолчанию (или явным образом с помощью константы CURL_SSLVERSION_DEFAULT), определяет выбор самой последней и безопасной версии сетевого протокола, доступной для имеющейся версии cURL.

Обслуживание запросов API

В программе на PHP можно также обслуживать запросы, направляемые вебклиентами прикладному программному интерфейсу API. Вместо того чтобы формировать обыкновенную HTML-страницу, достаточно сформировать те данные, которые пригодны для обращения к прикладному программному интерфейсу API. Кроме того, может возникнуть потребность манипулировать кодом HTTP-ответа и заголовками, посылаемыми в ответах на запросы.

Чтобы отправить HTTP-заголовок вместе с ответом, следует воспользоваться функцией header(). В примере 11.15 демонстрируется крошечный прикладной программный интерфейс API, написанный на PHP. Он обслуживает запросы текущего времени в формате JSON.

Пример 11.15. Обслуживание ответа на запрос текущего времени в формате JSON

```
$response_data = array('now' => timed);
header('Content-Type: application/json');
print json_encode($response_data);
```

При вызове функции header() в коде из примера 11.15 в HTTP-ответ, формируемый интерпретатором PHP, вводится строка заголовка, в которой можно передать содержимое заголовка любого типа. А далее в коде из примера 11.15 вызывается функция json_encode(), чтобы сформировать ответ в формате JSON. Действие этой функции противоположно действию упоминавшейся ранее функции json_decode(). Ей передается аргумент одного из типов данных PHP (символьная строка, число, объект, массив и т.д.), а возвращает она символьную строку, содержащую представление в формате JSON тех данных, которые были переданы ей в качестве аргумента. HTTP-ответ, формируемый в коде из примера 11.15, выглядит следующим образом:

```
HTTP/1.1 200 OK
Host: www.example.com
Connection: close
Content-Type: application/json

{"now":1962258300}
```

В первых четырех из приведенных выше строк выводятся заголовки. Лишь некоторые из них автоматически добавляются веб-сервером, а строка `Content-Type: application/json` получается в результате вызова функции `header()`. После пустой строки следует тело запроса. Этот результат можно наблюдать в окне веб-браузера, а если он представлен в формате HTML, то отображается в окне веб-браузера с учетом HTML-разметки. В данном случае тело запроса составляет объект в формате JSON с одним свойством `now`, значением которого является отметка текущего времени. Результат, который вы получите при выполнении кода из рассматриваемого здесь примера, скорее всего, будет иным, поскольку этот код будет выполняться в другое время, отличное от указанного выше. Подробнее о том, как манипулировать временем в PHP, речь пойдет в главе 15.

В первой строке заголовков из представленного выше ответа на запрос текущего времени содержится код ответа. В данном случае — это код состояния **200**, который в спецификации сетевого протокола HTTP означает "хорошо", т.е. все прошло нормально. Чтобы отправить другой код ответа, следует воспользоваться функцией `http_response_code()`. Пример 11.16 похож на пример 11.15, за исключением того, что в ответ на запрос посылается код состояния **403** ("запрещено") и тело ответа с ошибкой, если только в параметре `key` из строки запроса не указано значение `pineapple`.

Пример 11.16. Изменение кода ответа

```
if (! (isset($_GET['key']) && ($_GET['key'] == 'pineapple'))) {
    http_response_code(403);
    $response_data = array('error' => 'bad key');
}
else {
    $response_data = array('now' => time());
}
header('Content-Type: application/json');
print json_encode($response_data);
```

Если не предоставить надлежащую строку запроса, в коде из примера 11.16 будет сформирован следующий ответ:

```
HTTP/1.1 403 Forbidden
Host: www.example.com
Connection: close
Content-Type: application/json

{"error":"bad key"}
```

В коде из примера 11.4 используется заголовок `Content-Type`, чтобы дать прикладному программному интерфейсу NDB API команду отправить обратно ответ в формате JSON. Чтобы получить доступ к заголовкам запросов, поступающих из программы на PHP, следует обратиться в массиву `$_SERVER`, где хранятся все заголовки из поступившего запроса. Ключ к этому массиву представлен в следующем формате: префикс **`HTTP_`**, имя заголовка, набранного прописными буквами со всеми дефисами (**`-`**), преобразованными в знаки подчеркивания (**`_`**). Например, значение заголовка `Content-Type` из поступившего запроса может храниться в элементе массива `$_SERVER['HTTP_CONTENT_TYPE']`. Так, в примере 11.17 значение заголовка `Accept` из поступившего запроса проверяется с целью выяснить порядок форматирования выводимых данных.

Пример 11.17. Проверка содержимого заголовка из поступившего запроса

```
<?php
```

```
// Форматы, которые требуется поддерживать
$formats = array('application/json','text/html','text/plain');
// Формат ответа, если он не указан
$default_format = 'application/json';

// Был ли предоставлен формат ответа?
if (isset($_SERVER['HTTP_ACCEPT'])) {
    // Если поддерживаемый формат предоставлен,
    // использовать его
    if (in_array($_SERVER['HTTP_ACCEPT'], $formats)) {
        $format = $_SERVER['HTTP_ACCEPT'];
    }
    // Предоставлен неподдерживаемый формат,
    // возвратить ошибку
    else {
        // Код состояния 406 означает "неприемлемо", т.е.
        // ответ требуется отправить в недоступном формате
        http_response_code(406);
        // выйти из программы без тела ответа, что вполне допустимо
        exit();
    }
} else {
    $format = $default_format;
}

// выяснить текущее время
$response_data = array('now' => timed);
// сообщить клиенту, какого типа содержимое ему посылается
header("Content-Type; $format");
// вывести время в надлежащем формате
if ($format == 'application/json') {
    print json_encode($response_data);
}
else if ($format == 'text/html') { ?>
    <!doctype html>
      <html>
        <head><title>Clock</title></head>
        <body><time><?= date('c', $response_data['now']) ?>
              </time></body>
      </html>
    <?php
} else if ($format == 'text/plain') {
    print $response_data['now'];
}
```

Если заголовок `Accept` из поступившего запроса содержит значение **application/json**, **text/html** или **text/plain**, в переменной `$format` устанавливается формат, подходящий для употребления. Это значение вводится в заголовок `Content-Type` посылаемого ответа и служит для формирования выводимых данных в надлежащем формате. Если же заголовок `Accept` не предоставлен, то используется устанавливаемое по умолчанию значение **application/json**. А если в заголовке `Accept` предоставлено какое-нибудь другое значение, то программа возвращает пустое тело ответа с кодом состояния ошибки **406**. Этим веб-клиент извещается, что данные были запро-

шены в неверном формате[2].

В массиве `$_SERVER` следует также искать ответ на вопрос, является ли текущий запрос безопасным, т.е. был ли он сделан по сетевому протоколу HTTPS. Если текущий запрос является безопасным, то в элементе массива `$_SERVER['HTTPS']` установлено значение **`on`**. Так, в коде из примера 11.18 проверяется, был ли текущий запрос сделан по сетевому протоколу HTTPS. Если это не так, то безопасный вариант текущего запроса перенаправляется по HTTPS URL.

Пример 11.18. Проверка безопасности текущего запроса

```
$is_https = (isset($_SERVER['HTTPS']) && ($_SERVER['HTTPS'] = 'on'));
if (! $is_https) {
    $newUrl = 'https://' . $_SERVER['HTTP_HOST']
                . $_SERVER['REQUEST_URI'];
    header("Location: $newUrl");
    exit();
}
print "You accessed this page over HTTPS. Yay!";
```

В первой строке кода из примера 11.18 определяется, был ли текущий запрос сделан по сетевому протоколу HTTPS. С этой целью проверяется, установлено ли значение в элементе массива `$_SERVER['HTTPS']` и равно ли оно **`on`**. Если оба эти условия оказываются истинными, то составляется безопасный вариант HTTPS текущего URL соединением правильного префикса сетевого протокола (**`https://`**) с именем хоста (т.е. значением элемента `$_SERVER['HTTP_HOST']`) и путем (т.е. значением `$_SERVER['REQUEST_URI']`) для текущего запроса. Если же запрос включает в себя любые параметры из строки запроса, то они дополнительно присоединяются из элемента массива `$_SERVER['REQUEST_URI']`. Заголовок `Location`, отправляемый функцией `header()`, переадресует веб-клиента по новому URL.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Простое извлечение данных по заданному URL с помощью функции `file_get_contents()`.
- Извлечение данных по заданному URL с включением параметров в строку запроса.
- Декодирование HTTP-ответа, присылаемого в формате JSON.
- Представление о потоковых контекстах в PHP.
- Включение дополнительных заголовков в запросы для излечения данных по заданному URL.
- Излечение данных по заданному URL методом `POST` с помощью функции `file_get_contents()`.
- Излечение данных по заданному URL средствами расширения cURL.
- Применение параметров из строки запроса в расширении cURL.
- Ввод заголовков в запрос средствами cURL.
- Обработка ошибок при выполнении запросов средствами cURL.

[2] Синтаксический анализ настоящих заголовков `Accept` несколько сложнее, потому что веб-клиентам разрешается отправлять запросы в нескольких форматах, указывая наиболее предпочтительные из них. Законченную реализацию данного процесса, называемого *согласованием содержимого*, можно найти по адресу `https://github.com/willdurand/Negotiation`.

- Излечение данных по заданному URL методом `POST` средствами cURL.
- Отслеживание cookie-файлов по сетевому протоколу HTTP средствами cURL.
- Безопасное применение расширения cURL по сетевому протоколу HTTPS.
- Обслуживание ответов на запросы без HTML-разметки.
- Изменение кода HTTP-ответа.
- Применение значений из заголовков HTTP-запросов.
- Проверка, был ли запрос сделан по сетевому протоколу HTTPS.

Упражнения

1. По адресу `http://php.net/releases/?json` находится лента подачи информации в формате JSON о последних выпусках версий языка PHP. Напишите программу, в которой функция `file_get_contents()` используется для извлечения из этой ленты и последующего вывода на экран информации о последней выпущенной версии языка PHP.
2. Видоизмените программу из предыдущего упражнения, чтобы воспользоваться расширением cURL вместо функции `file_get_contents()`.
3. Напишите программу для формирования веб-страницы, на которой cookie-файл используется с целью извещать пользователя о том, когда он в последний раз посещал данную веб-страницу (для этой цели могут пригодиться функции манипулирования датой и временем, описываемые в главе 15).
4. *Обменный фонд* GitHub (GitHub gist) — это фрагмент текста или кода, которым легко обмениваться с другими. Прикладной программный интерфейс GitHub API позволяет создавать такой обменный фонд без дополнительной регистрации. Напишите программу для создания обменного фонда, содержащего ее исходный код. Имейте в виду, что прикладной программный интерфейс GitHub API потребует от вас установить заголовок `User-Agent` в HTTP-запросах к нему. Для установки этого заголовка можете воспользоваться параметром `CURLOPT_USERAGENT`.

ГЛАВА 12

Отладка кода

Программы редко работают правильно с первого же раза, когда они запускаются на выполнение. В этой главе представлен ряд методик выявления и устранения ошибок в программах на PHP. Когда вы только начинаете осваивать язык PHP, ваши программы оказываются проще, чем у опытных программирующих на PHP. Но в целом устранять возникающие в них ошибки не намного проще, а для выявления и исправления этих ошибок приходится применять те же инструментальные средства и методики, которыми обычно пользуются опытные программирующие на PHP.

Управление выводом сообщений об ошибках

В вашей программе может произойти немало такого, что способно привести к генерированию интерпретатором PHP сообщения об ошибке. У вас имеется возможность выбрать место для вывода сообщений об ошибках. Такие сообщения можно, например, посылать вместе с другими выводимыми результатами выполнения программы веб-браузеру или же в журнал регистрации ошибок на веб-сервере.

Отображение сообщений об ошибках удобно сначала настроить на вывод их на экран во время разработки программы на PHP, а по окончании разработки и передачи программы в эксплуатацию — посылать сообщения об ошибках в журнал регистрации. В процессе разработки программы удобно сразу же выявлять, например, синтаксические ошибки в отдельных строках кода. Но как только программа будет признана работоспособной и пригодной для эксплуатации потребителями, появление сообщений о подобных ошибках могут их смутить.

Чтобы отображать сообщения об ошибках в окне браузера, достаточно установить значение **`On`** в директиве конфигурации `display_errors`. Если же в ней установлено значение **`Off`**, сообщения об ошибках не будут направляться браузеру для отображения в его окне. А для того чтобы ошибки в конечном итоге записывались в журнале регистрации на веб-сервере, следует установить значение On в директиве конфигурации `log_errors`.

Сообщение об ошибке, формируемое интерпретатором PHP, может быть отнесено к одной из следующих категорий.

Синтаксические ошибки. Это такие ошибки синтаксического характера в программе, как, например, отсутствие точки с запятой в конце оператора. Механизм PHP прерывает выполнение программы, когда обнаруживает синтаксическую ошибку.

Неисправимые ошибки. Это такие серьезные ошибки в программе, как, например, вызов неопределенной функции. Интерпретатор PHP прерывает выполнение программы, когда обнаруживает

неисправимую ошибку.

Предупреждения. Это рекомендации, которые интерпретатор PHP дает по поводу какого-нибудь подозрительного поведения программы, хотя ее выполнение не прерывается. Предупреждение может, например, появиться при вызове функции с неверным количеством аргументов.

Замечания. Это рекомендации, которые интерпретатор PHP дает по поводу какого-нибудь нарушения принятых норм программирования. Замечание может, например, появиться в том случае, если значение переменной выводится на экран без предварительной ее инициализации.

Строгие замечания и предупреждения об употреблении устаревших языковых средств. Это предостережения интерпретатора PHP по поводу выбранного стиля программирования или применения языковых средств, которые выйдут из употребления в последующих версиях PHP.

Совсем не обязательно, что вы будете извещены об ошибках всех перечисленных выше категорий. Директива конфигурации `error_reporting` определяет категории ошибок, о которых извещает интерпретатор PHP. По умолчанию в ней устанавливается значение `E_ALL & ~E_NOTICE & ~E_DEPRECATED`, которое предписывает интерпретатору PHP извещать обо всех категориях ошибок, кроме замечаний и предупреждений об употреблении устаревших языковых средств. В приложении А к данной книге поясняется, что обозначают знаки `&` и `~` в значениях директив конфигурации.

В языке PHP определен ряд констант для установки такого значения в директиве конфигурации `error_reporting`, чтобы интерпретатор PHP извещал об ошибках только определенных категорий. Эти константы перечислены ниже.

- `E_ALL` (все ошибки)
- `E_PARSE` (синтаксические ошибки)
- `E_ERROR` (неисправимые ошибки)
- `E_WARNING` (предупреждения)
- `E_NOTICE` (замечания)
- `E_STRICT` (строгие замечания, до версии PHP 7.0.0)

Строгие замечания появились в версии PHP 5 и поэтому не включены в язык PHP до версии 5.4.0. Чтобы интерпретатор PHP прежних версий посчитал нечто обнаруженное в программе возможной ошибкой и соответственно отреагировал, следует установить значение **`E_ALL | E_STRICT`** в директиве конфигурации `error_reporting`.

Устранение синтаксических ошибок

На самом деле интерпретатор PHP весьма разборчив и не очень многословен. Если вы не завершите оператор точкой с запятой или начнете символьную строку одиночной кавычкой, а завершите ее двойной кавычкой, интерпретатор PHP не выполнит такой код. Вместо этого он выдаст сообщение о синтаксической ошибке, оставляя вас перед малоприятной перспективой заниматься отладкой написанного вами исходного кода.

Весь исходный код программы должен быть набран таким образом, чтобы интерпретатор PHP нормально воспринял его. И это может оказаться самой неприятной обязанностью для начинающего программировать. Упростить этот процесс помогает написание программ в редакторе, способном анализировать исходный код PHP. Если уведомить такой редактор, что в нем редактируется программа на PHP, он активизирует специальные средства, упрощающие программирование.

К числу таких специальных средств относится *выделение синтаксических конструкций*. Оно изменяет цвет отдельных частей исходного кода программы в зависимости от назначения. Например, символьные строки выделяются розовым цветом, ключевые слова вроде `if` и `while` — голубым, комментарии — серым, а переменные — черным. Выделение синтаксических конструкций упрощает обнаружение, например, отсутствующих кавычек, закрывающих символьные строки, которые продолжаются до конца файла (или кавычки, следующей далее в исходном коде программы).

Еще одним удобным средством является *автоматическое закрытие кавычек и скобок*, помогающее правильно уравновесить кавычки и скобки. Когда вы вводите закрывающую фигурную скобку `}`, редактор автоматически выделяет соответствующую открывающую фигурную скобку `{`. В разных редакторах это делается по-разному, но, как правило, курсор на позиции открывающей фигурной скобке (`{`) начинает мигать или обе фигурные скобки (`{` и `}`) выделяются полужирным. Такое поведение удобно для ввода парных знаков препинания, в том числе одиночных и двойных кавычек, ограничивающих символьные строки, а также круглых, квадратных и фигурных скобок.

Такие редакторы отображают номера строк в исходных файлах программ. Когда вы получаете от интерпретатора PHP сообщение о синтаксической ошибке, например, в строке кода 35 вашей программы, то знаете, где искать ошибку. В табл. 12.1 перечислены текстовые редакторы, поддерживающие написание программ на PHP. Цены на них указаны в долларах США на момент написания данной книги.

Таблица 12.1. Текстовые редакторы, поддерживающие написание программ на PHP

Наименование	URL	Цена в долларах
PhpStorm	**https://www.jetbrains.com/phpstorm**	89
NetBeans	**https://netbeans.org**	Бесплатно
Zend Studio	**http://www.zend.com/en/products/studio**	89
Eclipse + PDT	**http://www.eclipse.org/pdt**	Бесплатно
Sublime Text	**http://www.sublimetext.com**	70
Emacs	**http://ergoemacs.org/emacs/which_emacs.html**	Бесплатно
Vim	**http://vim.wikia.com/wiki/Where_to_download_Vim**	Бесплатно

Программные продукты PhpStorm, NetBeans, Zend Studio и Eclipse + PDT в большей степени соответствуют традиционным интегрированным средам разработки (ИСР), тогда как редакторы Sublime Text, Emacs и Vim — традиционным текстовым редакторам, хотя их нетрудно настроить с помощью подключаемых модулей, помогающих распознавать исходный код PHP. Наиболее пригодными для написания программ на PHP считаются редакторы PhpStorm и Zend Studio, тогда как остальные редакторы позволяют программировать на многих других языках. Все коммерчески доступные редакторы, перечисленные в табл. 12.1, имеют бесплатные периоды оценивания, поэтому можете опробовать их, чтобы выбрать наиболее подходящий для вас.

Синтаксические ошибки возникают в тех случаях, когда интерпретатор PHP обнаруживает в программе нечто неожиданное. Обратимся к примеру 12.1, где демонстрируется неверно написанная программа.

Пример 12.1. Синтаксическая ошибка

```
<?php
if $logged_in) {
    print "Welcome, user.";
}
?>
```

Если попытаться выполнить код из примера 12.1, интерпретатор PHP выдаст следующее сообщение об ошибке:

```
PHP Parse error: syntax error, unexpected '$logged_in' (T_VARIABLE),
expecting '(' in welcome.php on line 2

[ Ошибка синтаксического анализа кода PHP: синтаксическая ошибка,
 неожиданная переменная '$logged_in' (I_VARIABLE) вместо ожидавшейся
 круглой скобки '(' в строке кода 2 исходного файла welcome.php ]
```

Приведенное выше сообщение об ошибке означает, что в строке 2 указанного исходного файла интерпретатор PHP предполагал обнаружить открывающую круглую скобку, а выявил переменную `$logged_in`, обозначаемую как *лексема* `T_VARIABLE`. С помощью лексем в интерпретаторе PHP выражаются различные основополагающие части программ. Когда интерпретатор PHP читает исходный текст программы, он преобразует его в перечень лексем. И там, где в исходном тексте программы введена переменная, интерпретатор PHP добавляет лексему `T_VARIABLE` в список.

Таким образом, интерпретатор PHP сообщает, что при чтении строки кода 2 была обнаружена переменная `$logged_in` там, где предполагалась открывающая скобка. Глядя на строку кода 2 из примера 12.1, нетрудно выявить причину подобной ошибки: отсутствие открывающей скобки, с которой должно начинаться проверочное выражение в условном операторе `if()`. Обнаружив ключевое слово **`if`**, интерпретатор PHP предполагал обнаружить далее открывающую скобку (**`(`**), с которой начинается проверочное выражение, а вместо нее он выявил переменную `$logged_in`.

Список всех лексем, употребляемых интерпретатором PHP, можно найти в оперативно доступном руководстве по языку PHP (`http://www.php.net/tokens`). Эти лексемы могут появляться в сообщениях об ошибках, выдаваемых интерпретатором PHP.

Но коварство синтаксических ошибок состоит в том, что номер строки кода, упоминаемый в сообщении об ошибке, зачастую не соответствует тому месту, где фактически возникает ошибка. Именно такой случай возникновения ошибки в коде и приведен в примере 12.2.

Пример 12.2. Труднообъяснимая ошибка

```
<?php
$first_name = "David';
if ($logged_in) {
    print "Welcome, $first_name";
} else {
    print "Howdy, Stranger.";
}
?>
```

При попытке выполнить код из примера 12.2 интерпретатор PHP выдаст следующее сообщение об ошибке:

```
PHP Parse error: syntax error, unexpected 'Welcome' (T_STRING)
in trickier.php on line 4

[ Ошибка синтаксического анализа кода PHP: синтаксическая ошибка,
 неожиданная символьная строка 'Welcome' (T_STRING)
 в строке кода 4 исходного файла trickier.php ]
```

В этом сообщении ошибка указана там, где ее на самом деле нет. Как бы тщательно вы ни пытались обнаружить указанную ошибку (наличие символьной строки `'Welcome'`) в строке кода 4, вам так и не удастся этого сделать. Строка кода `print "Welcome, $first_name";` составлена совершенно правильно, т.е. указанная в ней символьная строка заключена в двойные кавычки, а завершается эта строка кода точкой с запятой.

На самом деле искомая ошибка присутствует в строке кода 2 из примера 12.2. В частности, символьная строка, присваиваемая переменной `$first_name`, начинается с двойной кавычки, а

оканчивается одиночной кавычкой. Читая строку кода 2, интерпретатор PHP обнаруживает двойную кавычку и считает, что далее следует символьная строка. И поэтому он воспринимает весь остальной исходный код как содержимое символьной строки до тех пор, пока не встретится следующая (неэкранированная) двойная кавычка. Следовательно, интерпретатор PHP пропускает одиночную кавычку в строке кода 2, продолжая читать исходный код до тех пор, пока не встретится первая же двойная кавычка в строке 4. Обнаружив эту двойную кавычку, он интерпретирует ее как завершение символьной строки. Таким образом, все, что следует после этой двойной кавычки, он посчитает новой командой или оператором. Но ведь после этой двойной кавычки следует исходный код `Welcome, $first_name";`, который не имеет никакого смысла для интерпретатора PHP. Ведь он ожидает встретить точку с запятой сразу в конце оператора или же точку для сцепления только что определенной символьной строки со следующей символьной строкой. Но исходный код `Welcome, $first_name";` похож на неограниченную символьную строку, находящуюся не на своем месте, и поэтому интерпретатор PHP выдает сообщение о синтаксической ошибке в строке кода 4.

Представьте, что вы несетесь по улицам Манхеттена со сверхзвуковой скоростью. Тротуар на 35-й улице имеет выбоины, о которые вы спотыкаетесь. Но поскольку вы движетесь так быстро, то приземляетесь на 39-й улице, разбившись до крови о мостовую и выпустив на нее свои кишки. Подойдя к вам, полицейский из службы безопасности дорожного движения воскликнет: “Эй! Происшествие на 39-й улице! Кто- то испачкал тротуар своими внутренностями!”

Аналогичным образом поступает в данном случае интерпретатор PHP. Номер строки кода, в которой он обнаруживает нечто неожиданное, не всегда отвечает именно той строке, где фактически возникает ошибка.

Получив от интерпретатора PHP сообщение о синтаксической ошибке, проанализируйте сначала строку кода, указанную в данном сообщении. Проверьте соблюдение самых основных правил синтаксиса, в том числе наличие точки с запятой в конце оператора. Если эта строка кода написана правильно, проанализируйте исходный код программы на несколько строк вперед и назад в поисках фактической ошибки. Обратите особое внимание на те знаки препинания, которые должны следовать парами: одиночные или двойные кавычки, в которые заключаются символьные строки; круглые скобки в вызовах функций или проверочных выражениях; квадратные скобки в элементах массивов; а также фигурные скобки в блоках кода. При этом количество открывающих (`(`, `[` и `{`) и закрывающих (`)`, `]` и `}`) знаков препинания должно совпадать.

В подобных случаях настоящую помощь оказывает редактор, поддерживающий написание программ на PHP. Выделяя синтаксис и автоматически закрывая кавычки и скобки, такой редактор может подсказать, где искать ошибки в набираемом исходном коде программы, облегчая их выявление.

Проверка данных в программе

Избавившись от синтаксических ошибок, возможно, придется потрудиться еще немного, прежде чем завершить отладку исходного кода программы. Ведь программа может быть написана синтаксически безупречно, но логически неверно. Как и предложение, составленное грамматически верно, но не имеющее никакого логического смысла, программа, в которой интерпретатор PHP не выявит никаких ошибок, не делает то, что он нее требуется.

Выявление и исправление логических ошибок в тех частях программы, которые ведут себя не так, как предполагалось, является немалой долей программирования в целом. Конкретные подходы к диагностике и анализу отдельных ситуаций сильно зависят от характера логических ошибок, которые требуется устранить. В этом разделе демонстрируются две методики исследования происходящего в программе на PHP. Первая из них более простая и состоит в добавлении операторов вывода отладочной информации, хотя для этого придется видоизменить исходный код программы, что неприемлемо для условий эксплуатации, где обычные пользователи не должны видеть результаты отладки программы. А вторая методика состоит в применении отладчика, который требует дополнительных затрат труда для своей настройки, но в то же время обеспечивает большую гиб-

кость для проверки программы во время ее выполнения.

Добавление операторов вывода отладочной информации

Если программа ведет себя странно, в нее можно ввести контрольные точки для вывода значений переменных на экран. Это даст возможность выяснить места, где поведение программы отличается от предполагаемого. В примере 12.3 демонстрируется программа, неправильно пытающаяся вычислить общую стоимость нескольких товаров.

Пример 12.3. Логически неверно написанная программа

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    $total_price = $price * $tax_rate;
}

printf('Total price (with tax): $%.2f', $total_price);
```

Код программы из примера 12.3 действует неверно, выводя на экран следующий результат:

```
Total price (with tax): $13.50
```

Общая стоимость товаров должна быть не менее 20 долларов. Что же не так в программе из примера 12.3? Чтобы выяснить это, можно, например, ввести в цикл `foreach()` строки кода, где значение переменной `$total_price` выводится до и после его изменения. Это даст возможность выяснить причину ошибки в математических расчетах. В примере 12.4 исходный код из примера 12.3 аннотирован рядом диагностических операторов `print`.

Пример 12.4. Логически неверно написанная программа с выводом отладочной информации на экран

```
$prices = array(5.95, 3.00, 12.50);
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    print "[before: $total_price]";
    $total_price = $price * $tax_rate;
    print "[after: $total_price]";
}

printf('Total price (with tax): $%.2f', $total_price);
```

При выполнении кода из примера 12.4 на экран выводится следующий результат:

```
[before: 0][after: 6.426][before: 6.426][after: 3.24][before: 3.24]
[after: 13.5]Total price (with tax): $13.50
```

Проанализировав результаты вывода отладочной информации в программе из примера 12.4, можно обнаружить, что значение переменной `$total_price` не увеличивается на каждом шаге цикла `foreach()`. Более тщательный анализ кода этой программы позволяет сделать вывод, что строку кода

```
$total_price = $price * $tax_rate;
```

следует заменить на такую:

```
$total_price += $price * $tax_rate;
```

Это означает, что вместо операции присваивания (**=**) в данной строке кода следует использовать операцию инкрементирования с присваиванием (**+=**).

Чтобы включить выводимую отладочную информацию в массив, достаточно воспользоваться функцией `var_dump()`, которая выводит на экран все элементы данного массива. Результат, выводимый из функции `var_dump()`, следует заключить в дескрипторы `<pre>` и `</pre>`, чтобы правильно представить его в формате HTML для отображения в окне браузера. Так, в примере 12.5 содержимое всех параметров переданной на обработку форму выводится с помощью функции `var_dump()`.

Редактирование нужного исходного файла

Если вы вносите изменения в программу во время ее отладки, но не видите, как эти изменения отражаются на ее поведении при повторной загрузке страницы в веб-браузер, отредактируйте нужный исходный файл. Поработав с локальной копией программы, но собираясь загрузить ее в браузер из удаленного сервера, скопируйте измененный исходный файл на этот сервер, прежде чем перезагружать страницу.

Чтобы согласовать редактируемый исходный файл со страницей, отображаемой в окне браузера, можно, например, временно ввести в самом начале исходного кода программы следующую строку кода с вызовом функции **`die()`**:

```
die('This is: ' . __FILE__);
```

Специальная константа `__FILE__` содержит имя исполняемого файла. Таким образом, при загрузке PHP-страницы, в самом начале которой находится приведенная выше строка кода, по URL вроде следующего:

```
http://www.example.com/catalog.php
```

в окне браузера появится лишь такая строка:

```
This is: /usr/local/htdocs/catalog.php
```

По результатам выполнения функции **`die()`** в окне браузера можно выяснить, какой именно исходный файл редактируется в настоящий момент. По окончании редактирования удалите вызов функции **`die()`** из исходного файла программы и продолжите ее отладку.

Пример 12.5. Вывод всех параметров переданной на обработку формы с помощью функции `var_dump()`

```
print '<pre>';
var_dump($_POST);
print '</pre>';
```

Отладочные сообщения, безусловно, информативны, но они могут нарушать и отвлекать от нормального вывода информации на обычной странице. Чтобы направить отладочные сообщения в журнал регистрации ошибок на сервере, следует воспользоваться функцией `error_log()` вместо оператора `print`. В примере 12.6 приведена та же самая программа, что и в примере 12.4, но в ней для отправки диагностических сообщений в журнал регистрации ошибок на сервере применяется

функция error_log().

Пример 12.6. Логически неверно написанная программа с выводом отладочной информации в журнал регистрации ошибок

```
$total_price = 0;
$tax_rate = 1.08; // налог 8%

foreach ($prices as $price) {
    error_log("[before: $total_price]");
    $total_price = $price * $tax_rate;
    error_log("[after: $total_price]");
}

printf('Total price (with tax): $%.2f', $total_price);
```

При выполнении кода из примера 12.6 на экран выводится только общая стоимость товаров, как показано ниже.

```
Total price (with tax): $13.50
```

В то же время следующие строки с отладочной информацией направляются в журнал регистрации ошибок на сервере:

```
[before: 0]
[after: 6.426]
[before: 6.426]
[after: 3.24]
[before: 3.24]
[after: 13.5]
```

Конкретное местоположение журнала регистрации ошибок на веб-сервере зависит от его конфигурации. Так, если применяется веб-сервер Apache, местоположение журнала регистрации ошибок определяется в директиве ErrorLog конфигурации этого веб-сервера

Функция var_dump() сама выводит информацию, поэтому следует принять дополнительные меры, чтобы направить выводимую отладочную информацию в журнал регистрации ошибок аналогично буферизации вывода, обсуждавшейся в конце раздела “Причины для размещения вызовов функций setcookie() и session_start() вначале страницы” главы 10. Функцию var_dump() следует вызывать в промежутке между вызовами функций, временно приостанавливающих и возобновляющих вывод информации, как показано в примере 12.7.

Пример 12.7. Отправка всех параметров из переданной на обработку формы в журнал регистрации ошибок с помощью функции var_dump()

```
// задержать вывод информации
ob_start();
// вызвать функцию var_dump(), как обычно
var_dump($_POST);
// сохранить в переменной $output всю выводимую информацию,
// сформированную с момента вызова функции ob_start()
$output = ob_get_contents();
// вернуться к обычному выводу информации
ob_end_clean();
// отправить содержимое переменной $output
// в журнал регистрации ошибок
error_log($output);
```

Функции `ob_start()`, `ob_get_contents()` и `ob_end_clean()` определяют в примере 12.7 порядок формирования интерпретатором PHP выводимой информации. В частности, функция `ob_start()` дает интерпретатору PHP команду не выводить пока что никакой информации, а накапливать во внутреннем буфере все, что предназначается для вывода. А когда вызывается функция `var_dump()`, интерпретатор PHP, выполняя команду функции `ob_start()`, направляет выводимую информацию во внутренний буфер. Функция `ob_get_contents()` возвращает содержимое внутреннего буфера. А поскольку с момента вызова функции `ob_start()` информацию выводила только функция `var_dump()`, то эта информация размещается в переменой `$output`. Далее функция `ob_end_clean()` отменяет команду, которая была дана интерпретатору PHP функцией `ob_start()`, предписывая ему вернуться в нормальный режим вывода информации. И, наконец, функция `error_log()` направляет содержимое переменной `$output` (в данном случае — отладочную информацию, выводимую функцией `var_dump()`) в журнал регистрации ошибок на веб-сервере.

Применение отладчика

Методика вывода и регистрации отладочной информации, описанная в предыдущем разделе, проста в употреблении. Но поскольку она требует видоизменения исходного кода программы, то ее нельзя применять в условиях эксплуатации, где обычные пользователи не должны видеть выводимую отладочную информацию. Кроме того, прежде чем запускать программу на выполнение, необходимо решить, какую именно информацию следует выводить на экран или в журнал регистрации ошибок. Чтобы вывести значение интересующей переменной, приходится снова видоизменять программу, вводя дополнительный код, а затем перезапускать ее.

Все эти затруднения устраняются, если исследовать работу программы с помощью специального отладчика. Отладчик позволяет анализировать работу программы во время ее выполнения, просматривая значения переменных и порядок вызова функций. Для этого не нужно вносить никаких изменений в исходный код программы, а только отдельно настроить режим ее отладки.

Для проверки работоспособности программ на PHP имеется несколько отладчиков, а многие редакторы, перечисленные в табл. 12.1, содержат встроенные отладчики или вполне совместимы с внешними отладчиками для проверки работоспособности программы на PHP, которую можно выполнять непосредственно в редакторе. В этом разделе рассматривается методика проверки работоспособности программ с помощью отладчика phpdbg, входящего в состав PHP.

Отладчик phpdbg входит в состав PHP, начиная с версии 5.6, но ваша установка интерпретатора PHP может быть не настроена на включение в его состав данного отладчика. Если в вашей системе отсутствует программа `phpdbg`, проверьте (или обратитесь к своему системному администратору за помощью проверить), что ваша установка PHP была создана с параметром **`--enable-phpdbg`**.

Отладчик `Xdebug` (`https://xdebug.org/`) является эффективным и полноценным диагностическим средством. Он может взаимодействовать с редакторами и ИСР по определенному протоколу, но не содержит свой собственный, удобный в употреблении клиент. Отладчик `Xdebug` доступен бесплатно.

Отладчик Zend Debugger входит в состав ИСР Zend Studio (`http://www.zend.com/en/products/studio`). В нем применяется собственный протокол для взаимодействия с ИСР Zend Studio, но с ним могут взаимодействовать и другие ИСР, например PhpStorm.

Чтобы начать сеанс отладки в отладчике phpdbg, достаточно запустить проверяемую программу на выполнение с параметром **`-e`**, обозначающим отлаживаемую программу, как показано ниже.

```
phpdbg -e broken.php
```

В ответ на эту команду отладчик выдаст следующее сообщение:

```
Welcome to phpdbg, the interactive PHP debugger, v0.4.0]
To get help using phpdbg type "help" and press enter
[Please report bugs to <http://github.com/krakjoe/phpdbg/issues>]
[Successful compilation of broken.php]

[ Добро пожаловать в phpdbg — диалоговый отладчик кода PHP,
    версия v0.4.0]
  Для получения справки введите команду "help" и нажмите
    клавишу <enter>
  [О выявленных программных ошибках сообщайте по следующему адресу:
    <http://github.com/krakjoe/phpdbg/issues>]
  [Исходный файл broken.php успешно скомпилирован broken.php] ]
```

Это означает, что отладчик phpdbg прочитал исходный файл `broken.php`, выбрал из него команды и готов выполнить их автоматически. Прежде всего необходимо установить *точку прерывания* в отлаживаемой программе. Этим отладчику phpdbg предписывается остановить выполнение программы в определенном ее месте. Как только отладчик phpdbg остановит выполнение программы в точке прерывания, можно приступить к анализу ее внутреннего состояния. В строке кода 7 из рассматриваемого здесь примера неверно написанной программы переменная `$total_price` получает свое значение в теле цикла, поэтому выполнение данной программы целесообразно прервать именно в этой строке кода, введя следующую команду:

```
prompt> break 7
```

В приведенной выше командной строке следует ввести лишь то, что выделено курсивом и полужирным, т.е. непосредственно команду, а не подсказку `prompt>`. Команда **`break 7`** сообщает отладчику phpdbg приостановить выполнение программы, как только будет достигнута строка кода 7. В ответ отладчик phpdbg выдаст следующее сообщение:

```
[Breakpoint #0 added at broken.php:7]

[ Точка прерывания #0 введена в строке
  кода 7 исходного файла broken.php ]
```

Чтобы начать отладку программы, необходимо дать отладчику phpdbg следующую команду на ее выполнение:

```
prompt> run
```

Отладчик начинает выполнение программы построчно от первой до седьмой строки кода, где установлена точка прерывания. В этой точке отладчик phpdbg выдаст следующее сообщение:

```
[Breakpoint #0 at broken.php:7, hits: 1]
>00007:     $total_price = $price * $tax_rate;
 00008: }
 00009:
```

А теперь можно ввести *контрольную точку* для проверки переменной `$total_price`. Этим отладчику phpdbg предписывается приостанавливать выполнение программы всякий раз, когда требуется проверить изменение значения в переменной `$total_price`. Именно это и требуется для диагностики логической ошибки, поскольку в переменной `$total_price` устанавливается не то значение, которое предполагалось. Контрольная точка вводится по команде **`watch`** следующим образом:

```
prompt> watch $total_price
```

Отладчик phpdbg отвечает следующим сообщением:

```
[Set watchpoint on $total_price]

[ Установлена контрольная точка для переменной $total_price ]
```

А теперь, когда контрольная точка установлена, точка прерывания в строке кода 7 больше не требуется. Поэтому ее можно удалить по команде **break del** следующим образом:

```
prompt> break del 0
```

Этим отладчику phpdbg предписывается удалить установленную первую точку прерывания (аналогично индексированию элементов массива в PHP, отладчик phpdbg начинает нумерацию элементов отладки с **0**, а не с **1**). Отладчик phpdbg подтверждает удаление точки прерывания следующим сообщением:

```
[Deleted breakpoint #0]

[ Удалена точка прерывания #0 ]
```

Итак, все готово для того, чтобы продолжить выполнение программы и приостановить его всякий раз, когда изменяется значение переменной `$total_price`. Приведенная ниже команда **continue** предписывает отладчику phpdbg продолжить выполнение программы.

```
prompt> continue
```

Отладчик phpdbg начинает выполнение программы. Первыми выполняются команды из строки кода 7, где изменяется значение в переменной `$total_price`. После этого выполнение программы немедленно останавливается, и отладчик phpdbg выдает следующее сообщение:

```
[Breaking on watchpoint $total_price]
Old value: 0
New value: 6.426
>00007:      $total_price = $price * $tax_rate;
 00008: }
 00009:

[ Прерывание в контрольной точке $total_price]
  Прежнее значение: 0
  Новое значение: 6.426 ]
```

Это очень удобно, поскольку можно ясно видеть, как значение переменной `$total_price` изменяется в коде с **0** на **6.426**. Чтобы выяснить, что произойдет дальше, достаточно ввести команду **continue** снова, как показано ниже.

```
prompt> continue
```

После этого выполнение программы остановится снова, а отладчик phpdbg выдаст следующее сообщение:

```
[Breaking on watchpoint $total_price]
Old value: 6.426
New value: 3.24
>00007:    $total_price = $price * $tax_rate;
 00008: }
 00009:
```

При выполнении цикла в строке кода 7 значение переменной `$total_price` изменяется с **`6.426`** на **`3.24`**. Но ведь это совершенно неверно, поскольку значение переменной `$total_price` должно увеличиться! Выполнение программы можно продолжить дальше по следующей команде:

```
prompt> continue
```

И в последний раз значение переменной `$total_price` изменяется следующим образом:

```
[Breaking on watchpoint $total_price]
Old value: 3.24
New value: 13.5
>00007:     $total_price = $price * $tax_rate;
 00008: }
 00009:
```

На этот раз значение данной переменной увеличивается до **`13.5`**. И в последний раз команда **`continue`** выдается для завершения программы, как показано ниже.

```
prompt> continue
```

Отладчик phpdbg продолжает выполнение программы и фактически выдает окончательный результат ее выполнения:

```
Total price (with tax): $13.50
[$total_price was removed, removing watchpoint]
[Script ended normally]

[ Общая стоимость (с учетом налога): $13.50
  [Переменная $total_price удалена, удаляется
    контрольная точка]
  [Сценарий завершился нормально] ]
```

Когда отладчик phpdbg приостанавливает выполнение программы в контрольной точке во второй раз, становится очевидно, что логическая ошибка кроется в порядке расчета значения переменной `total_price`. К такому же выводу удалось прийти в результате анализа отладочной информации, выводимой по методике, описанной в предыдущем разделе.

Особенности синтаксиса вводимых команд (или выбираемых элементов ГПИ) могут отличаться в разных отладчиках или ИСР, но основной принцип действия остается прежним: отладчик выполняет программу под особым контролем. Выполнение программы можно приостанавливать в разных выбираемых для этой цели местах, анализируя внутреннее состояние программы в момент приостановки ее выполнения.

Обработка неперехватываемых исключений

В разделе “Индикация ошибок с помощью исключений” главы 6 пояснялись основы обработки исключений в PHP, а в примере 6.8 было демонстрировано, что произойдет, если исключение будет сгенерировано, но не перехвачено. В этом случае выполнение программы на PHP прервется, а интерпретатор PHP выведет сведения об ошибке и результат трассировки стека (т.е. перечень функций, которые по очереди вызывались в тот момент, когда было прервано выполнение программы).

Несмотря на то что любой код, который способен сгенерировать исключение, следует всегда заключать в блоки операторов `try/catch`, на практике это не всегда удается сделать, чтобы идеально удовлетворить целям обработки исключений. Можно, например, воспользоваться сторонней библиотекой, даже не подозревая об исключениях, которые она генерирует, или же совершить ошибку и забыть о ситуации, в которой прикладной код может сгенерировать исключение. Для подобных случаев в PHP предоставляется возможность указать специальный обработчик исключений, который будет вызван, если исключение не обрабатывается в прикладном коде. Такой обработчик исключений служит удобным местом для регистрации сведений об исключении и предоставления пользователю программы более удобной информации, чем результат трассировки стека.

Чтобы воспользоваться специальным обработчиком тех исключений, которые иначе не обрабатываются, необходимо сделать следующее.

1. Написать функцию, которая обработает исключение. Эта функция принимает в качестве единственного аргумента исключение, передаваемое ей на обработку.

2. Вызвать функцию `set_exception_handler()`, чтобы сообщить интерпретатору PHP о функции, обрабатывающей исключение.

В примере 12.8 демонстрируется установка обработчика исключений, выводящего удобное для восприятия пользователем сообщение об ошибке и регистрирующего дополнительные сведения о возникшем исключении.

Пример 12.8. Установка специализированного обработчика исключений

```
function niceExceptionHandler($ex) {
    // выдать удобное для восприятия пользователем
    // сообщение об ошибке
    print "Sorry! Something unexpected happened.
           Please try again later.";
    // зарегистрировать дополнительные сведения об исключении
    // для последующего анализа системным администратором
    error_log("{$ex->getMessage()} in {$ex->getFile()}
                {$ex->getLine()}");
    error_log($ex->getTraceAsString());
}
set_exception_handler('niceExceptionHandler');

print "I'm about to connect to a made up, pretend,
       broken database!\n";

// Имя источника данных, предоставленное конструктору
// класса PDO, не обозначает достоверную базу данных или
// параметры подключения к ней, и поэтому конструктор данного
// класса сгенерирует исключение
$db = new PDO('garbage:this is obviously not going to work!');

print "This is not going to get printed.";
```

В функции `niceExceptionHandler()` из примера 12.8 оператор print служит для вывода простого сообщения об ошибке, удобного для восприятия пользователем, а функция `error_log()` вызывается вместе с методами для объекта типа Exception с целью зарегистрировать дополнительные сведения об исключении для последующего их анализа. Если вызвать функцию `set_exception_handler()` со строковым значением **`niceExceptionHandler`** в качестве аргумента, то интерпретатору PHP будет предписано направлять неперехватываемые исключения на обработку функции `niceExceptionHandler()`.

При выполнении кода из примера 12.8 на экран выводится следующий результат:

```
I'm about to connect to a made up, pretend, broken database!
Sorry! Something unexpected happened. Please try again later.

[ Попытка подключиться к вымышленной, ненастоящей, нарушенной
    базе данных!
 Извините, но произошло нечто непредвиденное. Попробуйте
 сделать еще одну попытку! ]
```

А зарегистрированы будут следующие дополнительные сведения:

```
could not find driver in exception-handler.php @ 17
#0 exception-handler.php(17):
    PDO->__construct('garbage:this is...')
#1 (main)

[ не удалось найти драйвер в исходном файле exception-handler.php @ 17 ]
```

Благодаря этому исключается возможная утечка секретных сведений (например, учетных данных или путей к файлам базы данных), которые пользователь мог бы почерпнуть из технических подробностей возникшей ошибки, если бы они были выведены на экран. Вместо этого они сохраняются в журнале регистрации ошибок для последующего анализа.

Специальный обработчик исключений не препятствует прерыванию программы после обработки исключения. Как только обработчик исключений завершит свое выполнение, завершится и сама программа. Именно поэтому в примере 12.8 строка кода с сообщением `"This is not going to get printed."` (Это не предполагается для вывода) так и не выполняется.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Настройка отображения в окне веб-браузера, вывода в журнал регистрации ошибок или того и другого.
- Конфигурирование интерпретатора PHP на уровне сообщения об ошибках.
- Извлечение выгод из текстового редактора, поддерживающего написание программ на PHP.
- Расшифровка сообщений о синтаксических ошибках.
- Выявление и устранение синтаксических ошибок.
- Вывод отладочной информации с помощью оператора `print` и функций `var_dump()` и `error_log()`.
- Отправка отладочной информации, выводимой функцией `var_dump()`, в журнал регистрации ошибок с помощью функций буферизации.

- Проверка работоспособности программы с помощью отладчика во время ее выполнения.
- Обработка исключений, которые нельзя перехватить в каком-нибудь другом коде.

Упражнения

1. В следующей программе содержится синтаксическая ошибка:

```
<?php
$name = 'Umberto';
function say_hello() {
    print 'Hello, ';
    print global $name;
}
say_hello();
?>
```

   Не выполняя программу в интерпретаторе PHP, выясните, о какой ошибке должен сообщить этот интерпретатор при попытке выполнить программу. Какие изменения следует внести в данную программу, чтобы она выполнялась правильно и выводила на экран сообщение `"Hello, Umberto"`?

2. Видоизмените ответ на задание написать функцию `validate_form()` в упражнении 3 из главы 7 таким образом, чтобы она выводила имена и значения всех параметров из переданной на обработку формы в журнал регистрации ошибок на веб-сервере.

3. Видоизмените ответ на задание в упражнении 4 из главы 8 таким образом, чтобы воспользоваться специальной функцией обработки ошибок обращения к базе данных, выводящей разные сообщения в окно веб-браузера и журнал регистрации ошибок на веб-сервере. Эта функция обработки ошибок должна совершать выход из программы после вывода сообщений об ошибках.

4. Приведенная ниже программа предназначена для вывода в алфавитном порядке списка всех посетителей ресторана из таблицы по заданию в упражнении 4 из главы 8. Выявите и устраните имеющиеся в ней ошибки.

```
<?php
// подключиться к базе данных
try {
    $db = new PDO('sqlite::/tmp/restaurant.db');
} catch ($e) {
    die("Can't connect: " . $e->getMessage());
}
// организовать обработку исключений
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// установить режим извлечения строк из таблицы в виде массивов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
// получить массив наименований блюд из базы данных
$dish_names = array();
$res = $db->query('SELECT dish_id,dish_name FROM dishes');
foreach ($res->fetchAll() as $row) {
    $dish_names[ $row['dish_id']]] = $row['dish_name'];
```

```
}
$res = $db->query('SELECT ** FROM customers ORDER BY phone DESC');
$customers = $res->fetchAll();
if (count($customers) = 0) {
    print "No customers.";
} else {
    print '<table>';
    print '<tr><th>ID</th><th>Name</th><th>Phone</th>
           <th>Favorite Dish</th></tr>';
    foreach ($customers as $customer) {
        printf("<tr><td>%d</td><td>%s</td>
                <td>%f</td><td>%s</td>
            </tr>\n",
          $customer['customer_id'],
          htmlentities($customer['customer_name']),
          $customer['phone'],
          $customer['favorite_dish_id']);
    }
    print '</table>';
?>
```

ГЛАВА 13

Тестирование: проверка правильности работы программы

Как же узнать, делает ли программа именно то, что она должна делать? Даже если проявить особое внимание к деталям, можно ли быть уверенным, что функция расчета налога с оборота действует как следует? Как убедиться в этом?

В настоящей главе поясняется, как найти ответы на эти вопросы. *Модульное тестирование* является методикой составления утверждений о небольших фрагментах кода, например: если передать функции одни конкретные значения, то возвратит ли она другие предполагаемые значения. Составляя тесты, проверяющие поведение кода в подходящих ситуациях, можно убедиться, что программа ведет себя предполагаемым образом.

Инструментальное средство PHPUnit фактически стало стандартом для написания модульных тестов кода PHP. Сами тесты состоят из небольших фрагментов кода PHP. В следующем разделе описывается установка PHPUnit, а в разделе "Написание тестов" демонстрируется исходный код и его первый тест. Этим кодом можно воспользоваться для проверки правильности установки PHPUnit и уяснения основных составляющих теста.

Далее, в разделе "Изолирование тестируемого кода", поясняется, как сосредоточить основное внимание на тестируемом коде с наибольшей эффективностью. После этого в разделе "Разработка посредством тестирования" обсуждается расширение тестируемого кода рядом тестов еще не существующего кода, а также рассматривается ввод кода для прохождения тестов. Такая методика разработки может оказаться очень удобной для написания кода, протестированного надлежащим образом. И в завершающем эту главу разделе "Дополнительные сведения о тестировании" подробно разъясняется, где искать дополнительные сведения о PHPUnit в частности и тестировании вообще.

Установка PHPUnit

Чтобы запустить инструментальное средство PHPUnit на выполнение, проще всего загрузить автономный архив PHP всего пакета PHPUnit и сделать его исполняемым. Как поясняется на веб-сайте, посвященном PHPUnit, по адресу `https://phpunit.de/getting-started.html`, проект PHPUnit обеспечивает доступность этого архива по адресу `https://phar.phpunit.de/phpunit.phar`. Этот архивный файл можно свободно загрузить и сделать исполняемым непосредственно, как демонстрируется в примере 13.1, или выполняемым по команде **php** из командной строки, как показано в примере 13.2.

Пример 13.1. Запуск PHPUnit в виде исполняемого архивного PHAR-файла

```
# По следующей команде архивный файл phpunit.phar делается
# исполняемым, при условии что он находится в текущем каталоге
```

```
chmod a+x phpunit.phar
# А по следующей команде этот файл запускается на выполнение
./phpunit.phar --version
```

Пример 13.2. Запуск PHPUnit по команде php из командной строки

```
php ./phpunit.phar --version
```

Как бы ни запускать PHPUnit на выполнение, если все будет сделано правильно, в результате выполнения команды **phpunit.phar - -version** на экран будет выведено следующее сообщение:

```
PHPUnit 4.7.6 by Sebastian Bergmann and contributors.
```

Если вы решите воспользоваться PHPUnit для тестирования кода в крупном проекте, опирающемся на систему Composer для управления пакетами и зависимостями между ними, добавьте ссылку на PHPUnit в разделе `require-dev` файла конфигурации `composer.json`, выполнив следующую команду:

```
composer require-dev phpunit/phpunit
```

Написание тестов

Функция `restaurant_check()` из примера 5.11 вычисляет общую сумму в ресторанном счете с учетом налога на добавленную стоимость и чаевых. Ради напоминания эта функция снова демонстрируется в примере 13.3.

Пример 13.3. Функция `restaurant_check()`

```
function restaurant_check($meal, $tax, $tip) {
    $tax_amount = $meal * ($tax / 100);
    $tip_amount = $meal * ($tip / 100);
    $total_amount = $meal + $tax_amount + $tip_amount;
    return $total_amount;
}
```

Тесты в PHPUnit организуются в виде методов в классе. Класс, создаваемый с целью содержать тесты, должен расширять класс `PHPUnit_Framework_TestCase`. Имя каждого метода, реализующего тест, должно начинаться с префикса **test**. В примере 13.4 демонстрируется класс, содержащий тест для функции `restaurant_check()`.

Пример 13.4. Тестирование функции, вычисляющей общую сумму в ресторанном счете

```
include 'restaurant-check.php';

class RestaurantCheckTest extends PHPUnit_Framework_TestCase {

    public function testWithTaxAndTip() {
        $meal = 100;
        $tax = 10;
        $tip = 20;
        $result = restaurant_check($meal, $tax, $tip);
        $this->assertEquals(130, $result);
    }

}
```

В примере 13.4 допускается, что функция `restaurant_check()` определена в файле `restaurant-check.php`, в который она включается перед определением тестового класса. А вы должны обеспечить загрузку тестируемого кода и его доступность для обращения из тестирующего класса.

Чтобы выполнить тест, достаточно указать имя файла, в котором сохранен его код, в качестве параметра программы PHPUnit, запускаемой на выполнение из командной строки:

```
phpunit.phar RestaurantCheckTest.php
```

В итоге на экран выводится следующий результат:

```
PHPUnit 4.8.11 by Sebastian Bergmann and contributors.

.

Time: 121 ms, Memory: 13.50Mb

OK (1 test, 1 assertion)
```

Каждая точка (`.`) перед строкой `Time: 121 ms, Memory: 13.50Mb` (Время выполнения: 121 мс, Объем использованной оперативной памяти: 13,5 Мбайт) обозначает один выполненный тест. А в последней строке (`OK (1 test, 1 assertion)` — Проведено успешно (1 тест, 1 утверждение)) сообщается состояние всех тестов, количество выполненных тестов, а также количество утверждений, содержащихся во всех тестах. Состояние `OK` означает, что все тесты прошли успешно. В данном примере имелся один тестовый метод `testWithTaxAndTip()`, в теле которого присутствовало следующее единственное утверждение: в результате вызова метода `assertEquals()` должно быть возвращено значение, равное **130**.

Тестовый метод обычно имеет такую же структуру, как и в предыдущем примере. Его имя начинается с префикса **`test`**, описывающего поведение, которое проверяется в данном методе. Сначала в теле тестового метода выполняется инициализация или установка любой переменной для проверки тестируемого кода. Затем вызывается тестируемый код, а далее делается ряд утверждений о происходящем. Эти утверждения доступны в виде методов экземпляра из тестового класса `PHPUnit_Framework_TestCase`, а следовательно, они доступны и в подклассе, производном от данного тестового класса.

Имена утверждающих методов начинаются с префикса **`assert`**. Эти методы позволяют проверять все особенности работы тестируемого кода, в том числе значения на равенство, наличие элементов в массиве или соответствие объекта экземпляру определенного класса. В приложении А к руководству по PHPUnit (`ttps://phpunit.de/manual/current/en/appendixes.assertions.html`) перечислены все утверждающие методы, доступные в этом инструментальном средстве модульного тестирования.

PHPUnit выдает совсем иной результат, если тест не проходит. В примере 13.5 приведен второй тестовый метод, добавляемый в класс `RestaurantCheckTest`.

Пример 13.5. Тест с утверждением, которое не подтверждается

```
include 'restaurant-check.php';

class RestaurantCheckTest extends PHPUnit_Framework_TestCase {

    public function testWithTaxAndTip() {
        $meal = 100;
        $tax = 10;
        $tip = 20;
        $result = restaurant_check($meal, $tax, $tip);
```

```
        $this->assertEquals(130, $result);
    }

    public function testWithNoTip() {
        $meal = 100;
        $tax = 10;
        $tip = 0;
        $result = restaurant_check($meal, $tax, $tip);
        $this->assertEquals(120, $result);
    }
}
```

В тестовом методе `testWithNoTip()` из примера 13.5 утверждается, что общая сумма в ресторанном счете составляет 100 долларов, а с учетом налога на добавленную стоимость и без чаевых — 120 долларов. Но это неверно, поскольку общая сумма должна быть равна 110 долларам. И в таком случае PHPUnit выводит на экран следующий результат тестирования:

```
PHPUnit 4.8.11 by Sebastian Bergmann and contributors.

.F

Time: 129 ms, Memory: 13.50Mb

There was 1 failure:

1) RestaurantCheckTest::testWithNoTip
Failed asserting that 110.0 matches expected 120.

RestaurantCheckTest.php:20

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

В начальной части выводимого результата тест получает состояние **F** (не пройден) вместо состояния **.**, поскольку он не прошел. Затем сообщаются подробные причины, по которым тест не прошел; сведения о тестовом классе и методе, не прошедшем тест; а также об утверждении, которое не удалось подтвердить. В частности, предполагалось получить значение общей суммы в ресторанном счете равным 120 (указано в качестве первого аргумента метода `assertEquals()`), но в итоге было получено значение **110** (указано в качестве второго аргумента метода `assertEquals()`). Если изменить утверждение в методе `testWithNoTip()` таким образом, чтобы в нем ожидалось значение **110**, то данный тест пройдет.

Для охвата тестами самых разных ситуаций и приобретения уверенности в должном поведении кода обычно требуется осмотрительный и творческий подход. Например, необходимо проверить, каким образом функция `restaurant_check()` вычисляет чаевые. Одни рассчитывают чаевые из общей стоимости заказанных блюд, другие — из общей стоимости заказанных блюд и налога. В примере 13.6 демонстрируются тесты, которые вводятся для проверки существующего поведения тестируемой функции, причем чаевые рассчитываются только по стоимости блюд, но без учета налога.

Пример 13.6. Тестирование расчета чаевых

```
include 'restaurant-check.php';

class RestaurantCheckTest extends PHPUnit_Framework_TestCase {
```

```
    public function testWithTaxAndTip() {
        $meal = 100;
        $tax = 10;
        $tip = 20;
        $result = restaurant_check($meal, $tax, $tip);
        $this->assertEquals(130, $result);
    }

    public function testWithNoTip() {
        $meal = 100;
        $tax = 10;
        $tip = 0;
        $result = restaurant_check($meal, $tax, $tip);
        $this->assertEquals(110, $result);
    }

    public function testTipIsNotOnTax() {
        $meal = 100;
        $tax = 10;
        $tip = 10;
        $checkWithTax = restaurant_check($meal, $tax, $tip);
        $checkWithoutTax = restaurant_check($meal, 0, $tip);
        $expectedTax = $meal * ($tax / 100);
        $this->assertEquals($checkWithTax,
                            $checkWithoutTax + $expectedTax);
    }
}
```

В методе `testTipIsNotOnTax()` рассчитываются два ресторанных счета; один — с учетом предоставляемого налога на добавленную стоимость, а другой — без учета такого налога. Поэтому общие суммы по обоим счетам должны отличаться на сумму налога, но не на сумму чаевых. В утверждении, которое делается в данном тестовом методе, проверяется равенство общих сумм по счету с учетом налога и по счету без учета налога плюс ожидаемая сумма налога. Этим гарантируется, что тестируемая функция не рассчитывает чаевые и по величине налога.

Изолирование тестируемого кода

Важный принцип производительного теста заключается в том, что тестируемый код должен быть изолирован в как можно большей степени. В идеальном случае не должно существовать глобального состояния или долгосрочного ресурса за пределами тестируемой функции, содержимое или поведение которой могло бы изменить результаты ее выполнения. Тестируемые функции должны давать одинаковые результаты независимо от порядка, в котором они выполняются.

Рассмотрим функцию `validate_form()` из примера 7.13. Чтобы проверить достоверность поступающих данных, она обращается к автоглобальному массиву `$_POST` и вызывает функцию `filter_input()`, чтобы оперировать непосредственно аргументом `INPUT_POST`. Это краткий способ доступа к данным, которые требуется проверить на достоверность. Но чтобы протестировать данную функцию, вероятно, придется откорректировать значения в массиве `$_POST`, хотя это вряд ли вообще поможет обеспечить нормальную работу функции `filter_input()`. Ведь эта функция все равно обратится к исходным, невидоизмененным данным из переданной на обработку формы, даже если изменить значения в массиве `$_POST`.

Чтобы сделать эту функцию тестируемой, ей нужно передать в качестве аргумента массив данных из предъявленной на обработку формы для проверки их достоверности. На этот массив можно затем ссылаться вместо массива `$_POST`, а его элементы проверить с помощью функции `filter_var()`. В примере 13.7 приведен изолированный подобным образом вариант функции `validate_form()`.

Пример 13.7. Проверка достоверности данных из переданной на обработку формы в изоляции

```
function validate_form($submitted) {
    $errors = array();
    $input = array();
    $input['age'] = filter_var($submitted['age'] ?? NULL,
                               FILTER_VALIDATE_INT);
    if ($input['age'] === false) {
        $errors[] = 'Please enter a valid age.';
    }

    $input['price'] = filter_var($submitted['price'] ?? NULL,
                                 FILTER_VALIDATE_FLOAT);
    if ($input['price'] === false) {
        $errors[] = 'Please enter a valid price.';
    }

    $input['name'] = trim($submitted['name'] ?? '');
    if (strlen($input['name']) == 0) {
        $errors[] = "Your name is required.";
    }
    return array($errors, $input);
}
```

В качестве первого аргумента функции `filter_var()` передается фильтруемая переменная. Здесь применяются обычные для PHP правила в отношении неопределенных переменных и индексов массива, и поэтому в нулеобъединяющей операции `($submitted['age'] ?? NULL)` для фильтрации предоставляется пустое значение `NULL`, если в массиве отсутствует соответствующее значение. А поскольку пустое значение `NULL` не является достоверным целочисленным или числовым значением с плавающей точкой, то функция `filter_var()` возвращает в подобных случаях логическое значение `false`, как если бы ей было предоставлено недостоверное числовое значение.

Когда видоизмененный вариант функции `validate_form()` вызывается в вебприложении, ей в качестве аргумента передается массив `$_POST`, как показано ниже.

```
list ($form_errors, $input) = validate_form($_POST);
```

В тестовом коде данной функции передается массив данных из формы, имитирующей тестируемую ситуацию, а полученные результаты проверяются с помощью утверждений. В примере 13.8 демонстрируется ряд тестов функции `validate_form()`: один — для проверки недопустимости указания возрастов в десятичной форме, другой — для проверки недопустимости указания цен со знаком доллара, третий — для проверки правильности возвращаемых значений, если достоверно предоставлены цена, возраст и имя.

Пример 13.8. Тестирование на достоверность данных из изолированной формы

```
// Функция validate_form() определена в следующем файле:
include 'isolate-validation.php';
```

```
class IsolateValidationTest extends PHPUnit_Framework_TestCase {

    public function testDecimalAgeNotValid() {
        $submitted = array('age' => '6.7',
                           'price' => '100',
                           'name' => 'Julia');
        list($errors, $input) = validate_form($submitted);
        // Ожидается только одна ошибка, связанная с возрастом
        $this->assertContains('Please enter a valid age.', $errors);
        $this->assertCount(1, $errors);
    }

    public function testDollarSignPriceNotValid() {
        $submitted = array('age' => '6',
                           'price' => '$52',
                           'name' => 'Julia');
        list($errors, $input) = validate_form($submitted);
        // Ожидается только одна ошибка, связанная с возрастом
        $this->assertContains('Please enter a valid price.',
                              $errors);
        $this->assertCount(1, $errors);
    }

    public function testValidDataOK() {
        $submitted = array('age' => '15',
                           'price' => '39.95',
                           // Начальные и конечные пробелы
                           // в имени должны быть удалены
                           'name' => ' Julia ');
        list($errors, $input) = validate_form($submitted);
        // Никаких ошибок не ожидается $this->assertCount(0, $errors);
        // Ожидаются три элемента во входных данных
        $this->assertCount(3, $input);
        $this->assertSame(15, $input['age']);
        $this->assertSame(39.95, $input['price']);
        $this->assertSame('Julia', $input['name']);
    }
}
```

В примере 13.8 применяется ряд новых утверждений, проверяемых в методах `assertContains()`, `assertCount()` и `assertSame()`. Утверждения, проверяемые в методах `assertContains()` и `assertCount()`, пригодны для массивов. Первое из них позволяет проверить, находится ли определенный элемент в массиве, а второе — размер массива. Оба эти утверждения выражают предполагаемое условие относительно массива `$errors` в двух первых тестах, а относительно массива `$input` — в третьем тесте.

Утверждение, проверяемое в методе `assertSame()`, подобно утверждению, проверяемому в упоминавшемся ранее методе `assertEquals()`, но два значения проверяются в нем не только на равенство, но и на одинаковость. Утверждение проходит проверку, если методу `assertEquals()` передается строка **`'130'`** и целое число **`130`**, тогда как в методе `assertSame()` оно не проходит проверку. С помощью утверждения из метода `assertSame()` в методе `testValidDataOK()` проверяется, правильно ли установлены типы переменных во входных данных посредством функции `filter_var()`.

Разработка посредством тестирования

Тесты широко применяются в общепринятой методике *разработки посредством тестирования* (TDD). Главный принцип TDD заключается в следующем: если требуется реализовать новое средство, прежде чем писать код, следует написать для него тест. В этом тесте выражается все, что должен делать код. А затем для нового средства пишется код, чтобы тест прошел.

И хотя методика TDD не является идеальной на все случаи жизни, тем не менее, она помогает ясно понять, что нужно делать, и обеспечивает полный охват разрабатываемого кода тестами. В качестве примера воспользуемся методикой TDD для внедрения в функцию `restaurant_check()` дополнительного средства, предписывающего учитывать налог на добавленную стоимость в общей сумме по ресторанному счету при расчете чаевых. Это средство реализуется в виде дополнительного четвертого аргумента данной функции. Если в качестве этого аргумента указывается логическое значение `true`, то в расчете суммы чаевых следует учесть налог на добавленную стоимость, а если указывается логическое значение `false`, то налог не учитывается. Если же в качестве четвертого аргумента не предоставляется вообще никакого значения, то функция `restaurant_check()` должна вести себя прежним образом.

Сначала составим тест, в котором функции `restaurant_check()` предписывается учитывать налог на добавленную стоимость в расчете суммы чаевых, а затем проверяется правильность общей суммы по ресторанному счету. Необходимо также составить тест, в котором проверяется правильность работы функции, когда ей явно предписывается не учитывать налог на добавленную стоимость в расчете суммы чаевых. Оба эти теста реализуются в соответствующих методах, которые демонстрируются в примере 13.9. (Ради большей ясности в данном примере показаны лишь два новых метода, а не весь тестовый класс.)

Пример 13.9. Внедрение тестов новой логики расчета чаевых

```
public function testTipShouldIncludeTax() {
    $meal = 100;
    $tax = 10;
    $tip = 10;
    // Логическое значение true четвертого аргумента
    // означает, что в расчете суммы чаевых следует
    // учитывать налог на добавленную стоимость
    $result = restaurant_check($meal, $tax, $tip, true);
    $this->assertEquals(121, $result);
}

public function testTipShouldNotIncludeTax() {
    $meal = 100;
    $tax = 10;
    $tip = 10;
    // Логическое значение false четвертого аргумента
    // означает, что в расчете суммы чаевых не следует
    // учитывать налог на добавленную стоимость
    $result = restaurant_check($meal, $tax, $tip, false);
    $this->assertEquals(120, $result);
}
```

Нет ничего удивительного, что новый тест в методе `testTipShouldIncludeTax()` не проходит, как показано ниже.

```
PHPUnit 4.8.11 by Sebastian Bergmann and contributors.
```

```
...F.

Time: 138 ms, Memory: 13.50Mb

There was 1 failure:

1) RestaurantCheckTest:TestTipShouldIncludeTax
Failed asserting that 120.0 matches expected 121.
RestaurantCheckTest.php:40

FAILURES!

Tests: 5, Assertions: 5, Failures: 1.

[ Версия PHPUnit 4.8.11,
    авторы: Кристиан Бергманн и прочие участники проекта
...F.

  Время выполнения: 138 мс, Объем использованной
    оперативной памяти: 13,5 Мбайт

  Обнаружено 1 непрохождение теста:

  1) RestaurantCheckTest::testTipShouldIncludeTax
  Не удалось утвердить, что значение 120.0 совпадает
  с предполагаемым значением 121.
  RestaurantCheckTest.php:40

  НЕ ПРОЙДЕНО!
  Тестов: 5, Утверждений: 5, Непрохождений: 1 ]
```

Чтобы этот тест прошел, функция `restaurant_check()` должна обработать четвертый аргумент, определяющий логику расчета суммы чаевых (пример 13.10).

Пример 13.10. Изменение логики расчета чаевых

```
function restaurant_check($meal, $tax, $tip,
                          $include_tax_in_tip = false) {
    $tax_amount = $meal * ($tax / 100);
    if ($include_tax_in_tip) {
        $tip_base = $meal + $tax_amount;
   } else {
        $tip_base = $meal;
   }
   $tip_amount = $tip_base * ($tip / 100);
   $total_amount = $meal + $tax_amount + $tip_amount;

   return $total_amount;
}
```

Благодаря внедрению новой логики в примере 13.10 функция `restaurant_check()` реагирует на свой четвертый аргумент, соответственно изменяя основание для расчета суммы чаевых. В этом новом варианте функции `restaurant_check()` все тесты благополучно проходят.

```
PHPUnit 4.8.11 by Sebastian Bergmann and contributors.

.....

Time: 120 ms, Memory: 13.50Mb

OK (5 tests, 5 assertions)
```

Тестовый класс включает в себя не только новые тесты для проверки новых функциональных возможностей, но и все прежние тесты. Тем самым гарантируется, что существующий код, где функция `restaurant_check()` применялась до внедрения нового средства, будет работать и впредь. Полный охват тестами надежно гарантирует, что изменения в коде не нарушат существующие функциональные возможности.

Дополнительные сведения о тестировании

По мере разрастания масштабов проектов возрастают и преимущества комплексного тестирования. Поначалу создается впечатление, будто приходится писать немало кажущегося лишним кода для проверки таких вполне очевидных вещей, как, например, элементарные математические операции в функции `restaurant_check()`. Но по мере накапливания в проекте все большего количества функциональных средств (а возможно, и большего количества людей, задействованных в проекте) накапливаемые тесты становятся просто бесценными.

Если не принимать во внимание некоторые усложненные формальные методики в области вычислительной техники, которые редко внедряются в современных прикладных программах на PHP, результаты тестирования позволяют получить ответ на вопрос: делает ли программа именно то, что она должна делать? Тесты позволяют выяснить, что же действительно программа делает, поскольку она выполняется разными способами, чтобы добиться предполагаемых результатов.

В этой главе были продемонстрированы основы внедрения инструментального средства PHPUnit в разрабатываемый на PHP проект и написания простых тестов. Для дальнейшего изучения рекомендуются следующие дополнительные ресурсы, посвященные PHPUnit в частности и тестированию вообще.

- Удобное и исчерпывающее руководство по PHPUnit, оперативно доступное по адресу `https://phpunit.de/manual/current/en/index.html`. Оно содержит сведения в форме учебного материала об основных задачах PHPUnit, а также справочный материал по функциональным средствам PHPUnit.
- Отличный список презентаций PHPUnit, приведенный по адресу `https://phpunit.de/presentations.html`.
- Каталог **`test`** с общедоступными пакетами PHP, где можно посмотреть, каким образом организованы их тесты. В информационном хранилище GitHub можно обнаружить тесты для компонентов `zend-form` (`https://github.com/zendframework/zend-form/tree/master/test`) и `zend-validator` (`https://github.com/zendframework/zend-validator/tree/master/test`) каркаса Zend Framework, а также тесты для общедоступного пакета Monolog (`https://github.com/Seldaek/monolog/tree/master/tests/Monolog`).
- Вполне естественно, что имеются многочисленные тесты (`https://github.com/sebastianbergmann/phpunit/tree/master/tests`) для проверки поведения самого инструментального средства PHPUnit. И эти тесты выполняются средствами PHPUnit!

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о преимуществах тестирования кода.
- Установка и выполнение PHPUnit.
- Представление о взаимодействии тестовых классов, тестовых методов и утверждений в PHPUnit.
- Написание тестов, проверяющих поведение функций.
- Выполнение тестов в PHPUnit.
- Представление о результатах, выводимых средствами PHPUnit, когда тесты проходят и не проходят.
- Представление о причинах для изолирования тестируемого кода.
- Удаление глобальных переменных из кода с целью сделать его более удобным для тестирования.
- Представление о методике разработки посредством тестирования.
- Написание тестов для новых функциональных средств прежде написания для него кода.
- Написание кода для прохождения новых тестов.
- Ресурсы для поиска дополнительных сведений о PHPUnit в частности и тестировании вообще.

Упражнение

1. Следуя инструкциям, приведенным в разделе “Установка PHPUnit” в начале этой главы. Для установки инструментального средства PHPUnit, напишите тестовый класс с единственным тестом, содержащим одно простое утверждение, которое передается соответствующему методу (например, `$this-> assertEquals(2, 1 + 1);`), а затем выполните созданный вами тестовый класс средствами PHPUnit.
2. Введите в код из примера 13.8 тест, обеспечивающий возврат ошибки, если в заполняемой форме не предъявлено имя.
3. Напишите тесты для проверки поведения функции `select()` из примера 7.29, рассмотрев следующие случаи.
   - Если предоставляется ассоциативный массив параметров, каждый дескриптор `<option>` должен быть воспроизведен с ключом из массива в качестве атрибута `value` дескриптора `<option>` и значением из массива в качестве текста, заключенного в дескрипторы `<option>` и `</option>`.
   - Если предоставляется числовой массив параметров, каждый дескриптор `<option>` должен быть воспроизведен с индексом массива в качестве значения атрибута `value` этого дескриптора и значением из массива в качестве текста, заключенного в дескрипторы `<option>` и `</option>`.
   - Если ни один из атрибутов не предоставлен, то открывающим должен быть дескриптор `<select>`.
   - Если предоставлен атрибут с логическим значением `true`, то в открывающем дескрипторе `<select>` должно быть указано только имя этого атрибута.

- Если предоставлен атрибут с логическим значением `false`, то такой атрибут не следует указывать в открывающем дескрипторе `<select>`.
- Если предоставлен атрибут с любым другим значением, то сам атрибут и его значение должны быть указаны в открывающем дескрипторе `<select>` в виде пары *атрибут=значение*.
- Если установлен атрибут `multiple`, к значению атрибута `name` должны быть присоединены квадратные скобки (**`[]`**) в открывающем дескрипторе `<select>`.
- Любые значения атрибутов или текстовые описания параметров, содержащие такие специальные символы, как **`<`** или **`&`**, должны быть воспроизведены с помощью закодированных HTML-представлений вроде **`<`** или **`&`**.

4. В спецификации форм HTML5 (`https://www.w3.org/TR/html5/forms.html`) подробно перечисляются конкретные атрибуты, разрешенные в каждом элементе формы. Полный набор возможных атрибутов многочислен и внушителен. Но действие некоторых атрибутов относительно ограничено. Например, в дескрипторе `<button>` поддерживаются следующие возможные значения для его атрибута `type`: **`submit`**, **`reset`** и **`button`**.

   Не видоизменяя вспомогательный класс `FormHelper`, напишите ряд новых тестов, в которых проверяется значение атрибута `type`, предоставляемое для дескриптора `<button>`. Этот атрибут является необязательным, но если он предоставляется, то должен принимать одно из трех допустимых значений. Завершив эти тесты, напишите для вспомогательного класса `FormHelper` новый код, благодаря которому эти тесты проходят.

ГЛАВА 14

Надлежащие нормы практики в программотехнике

В отличие от предыдущих глав, в этой главе подробно не обсуждается, что следует делать в программе на PHP. Вместо этого в ней рассматривается ряд инструментальных средств и методик, применяемых в разработке программного обеспечения вообще. Эти методики особенно полезны для координирования работы в коллективе разработчиков, хотя они могут оказаться ценными и для самостоятельной работы над проектом.

Код PHP, написанный для выполнения на компьютере конкретных задач, не составляет весь программный проект. Необходимо также следить каким-то образом за изменениями в написанном коде, чтобы иметь возможность вернуться к предыдущей его версии, если в него проникнет программная ошибка, или согласовать изменения, чтобы разработчики имели доступ к одним и тем же частям кода. Если проявятся программные ошибки или пользователи запросят новые функциональные средства, то как отслеживать все эти задачи? Была ли исправлена программная ошибка? Какой код пришлось изменить для ее исправления? Кто это сделал? Имеется ли версия кода с исправленной программной ошибкой, еще не доступная для пользователей? Для ответа на все эти вопросы в разделе "Контроль версий исходного кода" рассматриваются различные системы управления версиями исходного кода, а в разделе "Отслеживание ошибок" — система отслеживания ошибок и прочих недостатков в программах.

Когда вносятся изменения во все проекты, кроме самых мелких, то нет смысла редактировать код, уже работающий на конкретном, активно посещаемом веб-сайте, чтобы создавать новые трудности для его пользователей. Ведь их вряд ли обрадует, если какой-нибудь рабочий файл будет сохранен с опечаткой, а внесенные изменения погрузят сервер в пучину вычислений, отнимающих немало времени.

Вместо этого лучше работать с рядом исходных файлов, выпускаемых на серверах, с которыми взаимодействуют пользователи, лишь тогда, когда программа полностью готова к эксплуатации. В разделе "Среды и разработка" поясняется, как это делается и как добиться слаженного выполнения программ на PHP в разных контекстах. И в завершающем эту главу разделе "Масштабирование в перспективе" вкратце обсуждаются вопросы производительности веб-сайта и потребности в оптимизации его работы.

Контроль версий исходного кода

Система контроля версий исходного кода отслеживает изменения в исходных файлах. Она позволяет просматривать предысторию изменений в исходном коде, выяснять, кто и какие изменения в него вносил, а также сравнивать разные версии кода. С помощью системы контроля версий исходного кода два разработчика могут независимо друг от друга работать над изменениями в коде и

легко объединять их вместе.

Система контроля версий исходного кода приобретает особое значение, когда над проектом работает коллектив разработчиков, но она приносит пользу и в том случае, если проект выполняется одним разработчиком. Возможность заглянуть в предысторию проекта и увидеть код, который был написан на предыдущем этапе работы над проектом, оказывается настоящим спасением для разработчика, когда он пытается выяснить место и время внесения программной ошибки.

Имеется немало общедоступных систем контроля версий исходного кода, и выбираются они исходя из личных предпочтений (для собственных проектов) или на основании заранее принятых решений (в работе над существующим проектом, где такая система уже применяется). Так, ведение исходного кода для самого механизма PHP осуществляется с помощью системы контроля версий исходного кода Git. Исходный код интерпретатора PHP доступен для свободного просмотра по адресу `http://git.php.net`. К числу других общедоступных систем контроля версий исходного кода относятся Mercurial (`https://www.mercurial-scm.org/`) и Subversion (`http://subversion.apache.org/`).

Знакомство с системой Git

Система контроля версий исходного кода Git общедоступна, эффективна и исчерпывающа. Если вам не приходилось раньше пользоваться этой или любой другой системой контроля версий исходного кода, уделите немного времени изучению отличного учебного материала по Git, свободно доступного по адресу **`https://try.github.io`**. С помощью имитируемого в окне браузера приглашения из командной строки терминала вы сможете быстро овладеть основами работы с системой Git. В частности, вы научитесь давать Git команды отслеживать изменения в исходных файлах, вносить и отменять изменения в исходном коде, а также отображать перечень внесенных вами изменений.

Системы контроля версий исходного кода отличаются тем, что они манипулируют текстовыми файлами. А поскольку код PHP, по существу, хранится в ряде текстовых файлов, то не нужно принимать никаких особых мер, чтобы этими файлами можно было удобно управлять в любой общедоступной системе контроля версий исходного кода. Тем не менее, чтобы упростить ведение исходного кода, необходимо соблюсти ряд условий.

Первым условием является организация классов в исходных файлах. При написании объектно-ориентированного кода на каждый исходный файл должен приходиться лишь один класс, а имя файла должно совпадать с именем класса (плюс расширение **`.php`**). Если же классы определяются в пространствах имен, необходимо создать каталог для каждой составляющей пространства имен и разместить исходные файлы в этих каталогах.

Например, класс `CheeseGrater` размещается в исходном файле `CheeseGrater.php`. Если этот класс определен в пространстве имен `Utensils`, то исходный файл `CheeseGrater.php` размещается в подкаталоге `Utensils`. Для нескольких уровней пространства имен должны быть предусмотрены соответствующие подкаталоги. Так, класс, полностью описываемый именем `\Kitchen\Utensils\CheeseGrater`, находится по пути `Kitchen/Utensils/CheeseGrater.php`.

Данное условие называется PSR-4, где сокращение PSR означает *PHP Standard Recommendation* — стандартная рекомендация по PHP. Стандартные рекомендации по PHP (`http://www.php-fig.org/psr/`) являются нормами практики, принятыми для стиля программирования и организации большинства основных проектов, выполняемых на языке PHP.

Отслеживание ошибок

Существует немало способов отслеживания кода, с которым приходится работать. Системы формального отслеживания ошибок служат надежным средством для хранения перечней программных

ошибок, запросов новых функциональных средств и прочих видов работ, которые необходимо выполнить. Такие системы гарантируют, что каждое задание поручается лицу, ответственному за его выполнение. Каждое задание связано с соответствующими метаданными, включая приоритетность, оцениваемую продолжительность выполнения, состояние хода выполнения и комментарии к заданию. Эти метаданные намного упрощают сортировку, поиск и уяснение сути каждой ошибки.

Существует немало систем отслеживания ошибок, которые подобно системам контроля версий исходного кода выбираются, исходя из потребностей конкретного проекта, над которым предстоит работать коллективно или индивидуально. Если вы ищете свободно доступную систему отслеживания ошибок для опробования, рекомендуется обратить внимание на систему MantisBT (`http://www.mantisbt.org/`), поскольку ее открытый исходный код написан на PHP.

Системы отслеживания ошибок действуют безотносительно к конкретному применяемому языку программирования, и поэтому не нужно принимать никаких особых мер, чтобы программами на PHP можно было удобно управлять в любой общедоступной системе отслеживания ошибок. В то же время можно свободно ссылаться на идентификаторы ошибок в своей программе при написании кода, связанного с конкретной ошибкой.

Каждая ошибка отслеживается в системе по своему идентификатору, который может состоять из цифр, букв и любого их сочетания и обеспечивает кратный и однозначный способ ссылки на ошибку. Допустим, что программной ошибке с описанием “регистрация не действует при наличии знака + в адресе электронной почты” присваивается идентификатор **`MXH-26`** при входе в систему. При написании кода для устранения данной ошибки достаточно сделать ссылку на ее идентификатор в комментарии, как показано ниже.

```
// MXH-26: адрес электронной почты, закодированный в формате DRL,
// чтобы избежать затруднений при интерпретации знака +
$email = urlencode($email);
```

Таким образом, когда другой разработчик будет просматривать исходный код, он обратит внимание на идентификационный номер ошибки и найдет ее в системе отслеживания ошибок, чтобы ознакомиться с контекстом и пояснением причин ее возникновения в исходном коде.

Среды и разработка

В идеальном случае редактируемые исходные файлы при написании программы на PHP не совпадают с файлами, которые читает веб-сервер, отвечая на запросы пользователей. Непосредственное редактирование таких файлов для внесения правок “по живому” может вызвать немало осложнений, в том числе следующие.

- Пользователи сразу же обнаружат ошибку, если сохранить файл с опечаткой.
- Недобросовестные личности могут получить доступ к резервным копиям файлов, которые редактор сохраняет автоматически.
- Нельзя протестировать внесенные изменения, прежде чем их обнаружат реальные пользователи.

Во избежание подобных осложнений рекомендуется поддерживать разные *среды* — отдельные контексты, где можно выполнять свой код. Для этого, как минимум, потребуется среда *разработки*, а также *рабочая* среда. Именно в среде разработки вы выполняете свою основную работу над проектом, тогда как в рабочей среде — код, с которым взаимодействуют реальные пользователи. Как правило, среду разработки вы развертываете на своем компьютере, а рабочую среду — на сервере в информационном центре или на месте размещения у такого поставщика услуг облачного хостинга, как Amazon Web Services или Google Cloud Platform.

Как и в отношении остальных аспектов программотехники, обсуждаемых в этой главе, имеются самые разные способы организации различных сред, обмена кодом между ними и управления всеми задействованными компьютерами. Такие инструментальные средства и методики, как правило, действуют безотносительно к конкретному языку программирования. Но кое-что можно сделать и в самом коде PHP, чтобы упростить его благополучное выполнение в разных средах.

Самое главное — отделить информацию о конфигурации конкретной среды от исходного кода, чтобы вносить коррективы в конфигурацию среды, не изменяя сам код. К подобной информации относятся такие данные, как имена серверов баз данных и учетные данные регистрации, места расположения файлов регистрации, другие пути к элементам файловой системы, а также словесное описание регистрации. Как только эта информация окажется в отдельном файле, для ее применения в своей программе можно воспользоваться рядом методов и функций, предоставляемых в языке PHP.

В частности, функция `parse_ini_file()` преобразует содержимое пары "ключ- значение" из файла конфигурации (в том же самом формате, который используется в файле php.ini конфигурации интерпретатора PHP) в ассоциативный массив. Рассмотрим в качестве примера следующий файл конфигурации:

```
;
; Строки комментариев в файле конфигурации начинаются
; со знака точки с запятой
;

; Сведения о базе данных
; Значение параметра dsn следует заключить в кавычки
; из-за наличия в нем знака =
dsn="mysql:host=db.dev.example.com;dbname=devsnacks"
dbuser=devuser
dbpassword=raisins
```

В примере 14.1 демонстрируется чтение данных конфигурации, при условии, что они хранятся в файле **`config.ini`**, чтобы подключиться с их помощью к базе данных.

Пример 14.1. Чтение содержимого файла конфигурации

```
$config = parse_ini_file('config.ini');
$db = new PDO($config['dsn'], $config['dbuser'],
              $config['dbpassword']);
```

В коде из примера 14.1 массив, возвращаемый функцией `parse_ini_file()`, содержит ключи и значения, соответствующие строке *`ключ=значение`* в файле конфигурации `config.ini`. В другой среде с иной информацией о подключении к базе данных не придется ничего менять в самой программе на PHP. Для правильного подключения к базе данных потребуется лишь новый файл конфигурации `config.ini`.

Масштабирование в перспективе

В кругу разработчиков программного обеспечения или деловых людей, интересующихся построением или эксплуатацией крупных систем, можно нередко услышать вопросы вроде следующего: "Допускает ли система масштабирование?" Их не интересуют подробности. Им нужно знать, хотя бы приблизительно, что произойдет, когда система станет крупной и перегруженной. Замедлит ли свою работу веб-сайт, когда его одновременно посетят не 3, а 3 тысячи или 3 миллиона пользователей?

Начинающим разработчикам можно рекомендовать пока что не беспокоиться о масштабировании системы. Намного важнее добиться работоспособности системы под небольшой нагрузкой, чем пытаться заранее добиться нормальной ее работы под большой нагрузкой.

Масштабирование в PHP

Начать программировать на PHP совсем не трудно, и поэтому этот язык применяется для разработки множества веб-сайтов – не только мелких, но и довольно крупных. В социальной сети Facebook была даже создана своя версия интерпретатора PHP под названием HHVM (**`http://hhvm.com/`**) для более эффективного выполнения кода PHP. Этот интерпретатор применяется также в вебслужбах Baidu, Wikipedia и Etsy.

Более того, как только вы начинаете замечать снижение производительности в своем веб-приложении, главное затруднение, вероятно, представляет не ваш код PHP. На производительность веб-приложения оказывают влияние многие факторы. В частности, неэффективный запрос базы данных, для обработки которого требуется несколько секунд, сильно замедляет загрузку веб-страницы, даже если на выполнение своей части программы на PHP, посылающей запрос в базу данных или ответ из базы данных в формате HTML, потребуется всего лишь несколько миллисекунд. Веб-страница, которую сервер быстро отправляет клиенту, может медленно загружаться по мнению пользователя, если она содержит ссылки на сотни изображений, которые требуется загрузить и отобразить в окне веб-браузера.

Когда вы дойдете до стадии, когда потребуется ускорить работу тех частей вашего веб-приложения, которые написаны на PHP, воспользуйтесь *профилировщиком*, чтобы собрать сведения о том, насколько эффективно интерпретатор PHP выполняет ваш код. К числу самых распространенных профилировщиков с открытым исходным кодом относятся Xdebug (`https://xdebug.org/`) и XHProf (`http://www.php.net/xhprof`). Если профилировщик XHProf не был обновлен для работы с версией PHP 7, то в профилировщике Xdebug поддерживалась версия PHP 7 на момент ее выпуска 2.4.0rc1 в ноябре 2015 года.

Как упоминалось в главе 12, профилировщик и отладчик Xdebug вполне сопрягается с несколькими ИСР, включая PhpStorm и NetBeans. О том, как выполняется профилирование в ИСР PhpStorm, можно узнать из соответствующей статьи, доступной по адресу `https://www.jetbrains.com/help/phpstom/2016.2/profiling-with-xdebug.html`. Если вы не пользуетесь ИСР PhpStorm, обращайтесь за справкой к документации на Xdebug (`https://xdebug.org/docs/profiler`), где поясняется, как устанавливать и запускать на выполнение профилировщик Xdebug, а затем просматривать выводимые им результаты.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о системах контроля версий исходного кода.
- Организация классов в файлы по условию PSR-4.
- Применение систем отслеживания ошибок.
- Указание ссылок на идентификаторы ошибок в комментариях к исходному коду.
- Организация работы в отдельных средах для разработки и эксплуатации программного обеспечения.
- Размещение сведений о среде в файле конфигурации.

- Чтение файла конфигурации с помощью функции `parse_ini_file()`.
- Рекомендации не беспокоиться поначалу о масштабировании разрабатываемых веб-приложений.
- Ознакомление с особенностями работы профилировщиков Xdebug и XHProf по указанным первоисточникам.

ГЛАВА 15

Манипулирование датами и временем

Даты и время повсеместно применяются в веб-приложениях. Так, в корзине для покупок требуется обрабатывать даты поставки закупленных продуктов. На форуме требуется отслеживать моменты публикации сообщений. И во всех видах веб-приложений требуется отслеживать время последней регистрации каждого из пользователей, чтобы извещать их, например, о том, что с момента последней регистрации было опубликовано пятнадцать новых сообщений.

Манипулировать датами и временем в программах на PHP надлежащим образом несколько сложнее, чем символьными строками или числами. Дата или время не является единым значением, а совокупностью значений, например, они содержат месяц, день и год или часы, минуты и секунды. В силу этого выполнять математические операции над ними может быть непросто. Вместо этого для сложения или вычитания целых дат и моментов времени придется рассматривать отдельные составляющие их части и те значения, которые допустимы в каждой из них. Так, часы могут изменяться от **0** до **23**, минуты и секунды — от **0** до **59**, и не все месяцы содержат одинаковое количество дней.

Чтобы облегчить бремя манипулирования датами и временем, в языке PHP предоставляется специальный класс `DateTime`, инкапсулирующий всю информацию о конкретном моменте времени. С помощью методов из этого класса можно выводить на экран дату или время в выбранном формате, складывать или вычитать даты и манипулировать промежутками времени.

В данной книге под термином *составляющие части даты и времени* подразумевается массив или группа составляющих даты и времени, в том числе месяц, год, часы, минуты и секунды. А термин *строка отформатированной даты* или *отформатированного времени* обозначает символьную строку, содержащую конкретную совокупность составляющих частей даты и времени, например, `"четверг, 20 октября 2016 года"` или `"3 часа 54 минуты по полудни"`.

Отображение даты или времени

Чтобы отобразить дату или время, проще всего сообщить пользователям текущее время. С этой целью можно вызвать метод `format()` для созданного объекта класса `DateTime` показано в примере 15.1.

Пример 15.1. Вывод текущего времени на экран.

```
$d = new DateTime();
print 'It is now: ';
print $d->format('r');
print "\n";
```

Если интерпретатор PHP выдаст сообщение "`It is not safe to rely on the system's timezone settings`" (Полагаться на системные установки часовых поясов ненадежно) при выполнении кода из примера 15.1, обратитесь к разделу "Манипулирование часовыми поясами" далее в этой главе, чтобы выяснить, что именно это сообщение означает и как избавиться от его появления.

При выполнении кода из примера 15.1 в полдень 20 октября 2016 года на экран выводится следующий результат:

```
It is now: Thu, 20 Oct 2016 12:00:00 +0000
```

При создании объекта класса `DateTime` конструктору данного класса предоставляются время или дата для хранения в новом объекте. Если же этому конструктору не предоставляется никаких аргументов, как в примере 15.1, то используются текущие дата и время. Строка форматирования, передаваемая методу `format()`, определяет порядок форматирования даты и времени для вывода на экран.

Отдельные буквы в строке форматирования преобразуются в соответствующие величины времени. Так, в примере 15.2 демонстрируется вывод на экран месяца, дня и года.

Пример 15.2. Вывод на экран строки отформатированной даты

```
$d = new DateTime();
print $d->format('m/d/y');
```

При выполнении кода из примера 15.2 в полдень 20 октября 2016 года на экран выводится следующий результат:

```
10/20/16
```

В строке форматирования из примера 15.2 символ **m** преобразуется в числовой эквивалент текущего месяца (**10**), символ **d** — в числовой эквивалент дня месяца (**20**), а символ **y** — в числовой эквивалент года, состоящий из двух цифр (**04**). Знаки обратной косой черты не являются символами форматирования, и поэтому они остаются на своих местах в символьной строке, возвращаемой методом `format()`. В табл. 15.1 перечислены все специальные символы форматирования, распознаваемые в методе `DateTime::format()`.

Таблица 15.1. Символы форматирования даты и времени

Тип	Символ форматирования	Описание	Пределы изменения/пример
День	**j**	День месяца; числовой эквивалент	**1-31**
День	**d**	День месяца; числовой эквивалент с начальными нулями	**01-31**
День	**S**	Суффикс порядкового номера дня месяца на английском языке; текстовый эквивалент	**st, th, nd, rd**
День	**z**	День года; числовой эквивалент	**0-365**
День	**w**	День недели; числовой эквивалент, где **0** обозначает воскресенье	**0-6**
День	**N**	День недели; числовой эквивалент, где **1** обозначает понедельник	**1-7**

Продолжение табл. 15.1

Тип	Символ форматирования	Описание	Пределы изменения/пример
День	`D`	Сокращенное обозначение дня недели на английском языке; текстовый эквивалент	`Mon-Sun`
День	`L`	Полное обозначение дня недели на английском языке; текстовый эквивалент	`Monday-Sunday`
Неделя	`W`	Порядковый номер недели в году (по стандарту (ISO-8601); числовой эквивалент с начальными нулями, где `01` обозначает первую неделю года, насчитывающую не меньше четырех дней в текущем году, а понедельник является первым днем недели	`01-53`
Месяц	`M`	Сокращенное обозначение месяца на английском языке; текстовый эквивалент	`Jan-Dec`
Месяц	`F`	Полное обозначение месяца на английском языке; текстовый эквивалент	`January-December`
Месяц	`n`	Месяц; числовой эквивалент	`1-12`
Месяц	`m`	Месяц; числовой эквивалент с начальными нулями	`01-12`
Месяц	`t`	Протяженность месяца в днях; числовой эквивалент	`28-31`
Год	`y`	Год без столетия; числовой эквивалент	`00-99`
Год	`Y`	Год со столетием; числовой эквивалент	`0000-9999`
Год	`o`	Год со столетием (по стандарту (ISO-8601); числовой эквивалент года, к которому относится порядковый номер текущей недели (`W`)	`0000-9999`
Год	`L`	Признак високосного года, где 1 обозначает високосный год	`0,1`
Час	`g`	Час; числовой эквивалент в 12-часовом формате	`1-12`
Час	`h`	Час; числовой эквивалент в 12-часовом формате с начальными нулями	`01-12`
Час	`G`	Час; числовой эквивалент в 24-часовом формате	`0-23`
Час	`H`	Час; числовой эквивалент в 24-часовом формате с начальными нулями	`00-23`
Час	`a`	Час до или после полудня	`a.m.` или `p.m.`
Час	`A`	Час до или после полудня	`AM`, `PM`

Окончание табл. 15.1

Тип	Символ форматирования	Описание	Пределы изменения/пример
Минуты	`i`	Минуты; числовой эквивалент с начальными нулями	`00-59`
Секунды	`s`	Секунды; числовой эквивалент с начальными нулями	`00-59`
Микросекунды	`u`	Микросекунды; числовой эквивалент с начальными нулями	`000000-999999`
Часовой пояс	`e`	Идентификатор часового пояса; текстовый эквивалент	Из поддерживаемого списка часовых поясов (`http://www.php.net/timezones`)
Часовой пояс	`T`	Сокращенное обозначение часового пояса; текстовый эквивалент	`GMT`, `CEST`, `MDT` и т.д.
Часовой пояс	`O`	Смещение часового пояса относительно всеобщего скоординированного времени (UTC) в часах со знаком; текстовый эквивалент	`-1100 - +1400`
Часовой пояс	`p`	Относительно всеобщего скоординированного времени (UTC) в часах со знаком и двоеточием; текстовый эквивалент	`-11:00 - +14:00`
Часовой пояс	`Z`	Относительно всеобщего скоординированного времени (UTC) в секундах; числовой эквивалент	`-39600-50400`
Прочее	`I`	Признак перехода на летнее время, где 1 обозначает летнее время	`0,1`
Прочее	`B`	Образцовое время в Интернете; числовой эквивалент	`000-999`
Прочее	`c`	Дата, отформатированная по стандарту ISO-8601; текстовый эквивалент	`016-10-20T12:33:56+06:00`
Прочее	`r`	Дата, отформатированная по стандарту RFC-2822; текстовый эквивалент	`Thu, 20 Oct 2016 12:33:56 +0600`
Прочее	`U`	Количество секунд всеобщего скоординированного времени (UTC), прошедших после полудня (12:00:00) 1 января 1970 г.	`1476945236`

Синтаксический анализ даты и времени

Чтобы создать объект класса `DateTime`, представляющий конкретное время, это время следует передать в качестве первого аргумента конструктору данного класса. Этот аргумент указывается в виде символьной строки, обозначающей дату и время, которые должен представлять создаваемый объект. Объект типа `DateTime` распознает символьные строки с датой и временем в самых разных форматах. Исчерпывающий перечень всех форматов даты и времени можно найти по адресу `http://www.php.net/datetime.formats`. В примере 15.3 демонстрируется ряд форматов даты и времени, распознаваемых объектом типа `DateTime`.

Пример 15.3. Образцы строк отформатированных даты и времени, распознаваемые объектом типа `DateTime`

```
// Если предоставляется только время, то используется
// текущая дата в формате день/месяц/год
$a = new DateTime('10:36 am');
// Если предоставляется только дата, то используется
// текущее время в формате часы/минуты/секунды
$b = new DateTime('5/11');
$c = new DateTime('March 5th 2017');
$d = new DateTime('3/10/2018');
$e = new DateTime('2015-03-10 17:34:45');
// Объект типа DateTime распознает и микросекунды
$f = new DateTime('2015-03-10 17:34:45.326425');
// Отметка времени эпохи должна быть снабжена префиксом @
$g = new DateTime('@381718923');
// Типичный формат даты и времени для журнала регистрации
$h = new DateTime('3/Mar/2015:17:34:45 +0400');

// Допускаются также форматы относительной даты и времени!
$i = new DateTime('next Tuesday');
$j = new DateTime("last day of April 2015");
$k = new DateTime("November 1, 2012 + 2 weeks");
```

При выполнении кода из примера 15.3 в полдень 20 октября 2016 года в соответствующих переменных сохраняются полные величины даты и времени, как показано ниже.

```
Thu, 20 Oct 2016 10:36:00 +0000
Wed, 11 May 2016 00:00:00 +0000
Sun, 05 Mar 2017 00:00:00 +0000
Sat, 10 Mar 2018 00:00:00 +0000
Tue, 10 Mar 2015 17:34:45 +0000
Tue, 10 Mar 2015 17:34:45 +0000
Fri, 05 Feb 1982 01:02:03 +0000
Tue, 03 Mar 2015 17:34:45 +0400
Tue, 25 Oct 2016 00:00:00 +0000
Thu, 30 Apr 2015 00:00:00 +0000
Thu, 15 Nov 2012 00:00:00 +0000
```

Если имеются отдельные составляющие части даты и времени, переданные, например, из элементов формы, где пользователь может указать месяц, год, день или часы, минуты и секунды, эти части можно далее передать методам `setTime()` и `setDate()` для коррекции даты и времени,

хранящихся в объекте типа DateTime. Применение методов setTime() и setDate() демонстрируется в примере 15.4.

Пример 15.4. Установка отдельных составляющих частей даты и времени

```
// Элементы массива $_POST['mo'], $_POST['dy'] и $_POST['yr']
// содержат порядковый номер месяца, день и год, переданные
// из формы на обработку
//
// А элементы массива $_POST['hr'] и $_POST['mn'] содержат
// часы и минуты, переданные из формы на обработку

// Переменная $d содержит текущее время, но вскоре оно
// будет перезаписано
$d = new DateTime();

$d->setDate($_POST['yr'], $_POST['mo'], $_POST['dy']);
$d->setTime($_POST['hr'], $_POST['mn']);

print $d->format('r');
```

Если элемент массива $_POST['yr'] содержит значение **2016**, элемент массива $_POST['mo'] — значение **5**, элемент массива $_POST['dy'] — значение **12**, элемент массива $_POST['hr'] — значение **4**, а элемент массива $_POST['mn'] — значение **15**, то при выполнении кода из примера 15.4 на экран выводится следующий результат:

```
Thu, 12 May 2016 04:15:00 +0000
```

Переменная $d инициализируется текущими датой и временем при выполнении кода из примера 15.4. Тем не менее хранящее в ней содержимое создаваемого объекта типа DateTime изменяется в результате вызова методов setDate() и setTime().

Объект типа DateTime пытается максимально приспосабливаться при синтаксическом анализе поступающих данных. Рассмотрим, например, что произойдет, если в примере 15.4 элемент массива $_POST['mo'] примет значение **3**, а элемент массива $_POST['dy'] — значение **35**. И хотя такого числа, как 35-е марта, не бывает, объект типа DateTime это не беспокоит. Он трактует 35-е марта как 4-е апреля, исходя из того, что в марте 31 день, а следовательно, 35 - 31 = 4 (т.е. 4-е апреля). Так, в результате вызова $d->setDate(2016, 3, 35) в объекте типа DateTime устанавливается дата 4 апреля 2016 г.

Для более строгой проверки достоверности дней и месяцев в году служит метод checkdate(). Этот метод сообщает, насколько достоверными оказались указанные день и месяц для предоставленного ему года (пример 15.5).

Пример 15.5. Проверка достоверности дней и месяцев в году

```
if (checkdate(3, 35, 2016))
    print "March 35, 2016 is OK";

if (checkdate(2, 29, 2016))
    print "February 29, 2016 is OK";

if (checkdate(2, 29, 2017))
    print "February 29, 2017 is OK";
```

В примере 15.5 метод `checkdate()` возвращает логическое значение `true` лишь при втором вызове. Исход первого вызова метода `checkdate()` оказывается неудачным, поскольку в марте всегда 31 день, а не 35 дней. Исход третьего вызова данного метода также оказывается неудачным, поскольку 2017 год не является високосным.

Расчет даты и времени

Как только будет получен объект типа `DateTime`, представляющий конкретный момент времени, рассчитать дату и время не составит большого труда. Допустим, что в форме требуется предоставить ряд вариантов выбора даты и времени из списка. В примере 15.6 демонстрируется HTML-разметка такого списка дескриптором `<select>`, где можно выбрать конкретный день. В качестве первого варианта выбора указывается дата, соответствующая первому четвергу после запуска программы на выполнение. А далее следуют варианты выбора с датами через день.

Пример 15.6. Отображение ряда дней, выбираемых из списка

```
$daysToPrint = 4;
$d = new DateTime('next Tuesday');
print "<select name='day'>\n";
for ($i = 0; $i < $daysToPrint; $i++) {
    print " <option>" . $d->format('l F jS') . "</option>\n";
    // добавить 2 дня к текущей дате
    $d->modify("+2 day");
}
print "</select>";
```

На каждом шаге цикла в методе `modify()` из примера 15.6 изменяется дата, хранящаяся в объекте типа `DateTime`. Этот метод принимает символьную строку, содержащую дату в одном из относительных форматов, описанных на странице, оперативно доступной по адресу `http://www.php.net/datetime.formats.relative`, и соответственно корректирует эту дату. В данном случае дата передается методу `modify()` в относительном формате `+2 day`, и поэтому на каждом шаге цикла текущая дата продвигается вперед на 2 дня.

При выполнении кода из примера 15.6 в полдень 20 октября 2016 года на экран выводится следующий результат:

```
<select name='day'>
<option>Tuesday October 25th</option>
<option>Thursday October 27th</option>
<option>Saturday October 29th</option>
<option>Monday October 31st</option>
</select>
```

В методе `diff()` из объекта типа `DateTime` определяется разность между двумя датами. Этот метод возвращает объект типа `DateInterval`, инкапсулирующий промежуток между датами. Так, в примере 15.7 проверяется, означает ли указанная дата рождения человека, что ему больше 13 лет.

Пример 15.7. Расчет промежутка дат

```
$now = new DateTime();
$birthdate = new DateTime('1990-05-12');
$diff = $birthdate->diff($now);
```

```
if (($diff->y > 13) && ($diff->invert == 0)) {
    print "You are more than 13 years old.";
} else {
    print "Sorry, too young.";
}
```

В результате вызова `$birthdate->diff($now)` в коде из примера 15.7 возвращается новый объект типа `DateInterval`. Свойства этого объекта описывают промежуток между датами, хранящимися в переменных `$birthdate` и `$now`. В частности, свойство `y` содержит количество лет, а свойство `invert` — значение **`0`**, если разность дат положительна. Если же дата в переменной `$birthdate` следует после даты в переменной `$now`, то свойство `invert` содержит значение **`1`**. Остальными являются свойства `m` (месяцы), `d` (количество дней в месяце), `h` (часы), `i` (минуты), `s` (секунды) и `days` (общее количество дней в промежутке между двумя датами).

Манипулирование часовыми поясами

К сожалению, даты и время являются не просто совокупностями часов, минут, секунд, месяцев, дней и лет. Ради полноты одни должны также включать в себя часовые пояса. Так, полдень 20 октября 2016 года наступит в Нью-Йорке и Лондоне в разные моменты времени.

Интерпретатор PHP должен быть настроен на применение часового пояса, устанавливаемого по умолчанию. Для этого проще всего установить параметр `date.timezone` в файле конфигурации PHP[1]. Если же эти коррективы нельзя внести в файл конфигурации, то в программе на PHP следует вызвать функцию `date_default_timezone_set()`, прежде чем производить какие-нибудь манипуляции над любой датой или временем. В версии PHP 7 по умолчанию устанавливается часовой пояс UTC (Universal Coordinated Time — Всеобщее скоординированное время) для интерпретатора PHP, если не указать собственное значение по умолчанию.

На странице документации по PHP, оперативно доступной по адресу `http://www.php.net/timezones`, приводится довольно обширный перечень возможных значений часовых поясов, распознаваемых интерпретатором PHP. Но вместо того чтобы использовать местный часовой пояс, нередко принято устанавливать часовой пояс UTC, чтобы упростить разработку программного обеспечения. Этот часовой пояс соответствует долготе нуль градусов и не корректируется при переходе на летнее и зимнее время года. Несмотря на то что для преобразования отметки времени UTC в местное время достаточно произвести несложные расчеты в уме, применение времени UTC упрощает манипулирование датами и временем, поступающими из многих других серверов, находящихся в разных часовых поясах. Благодаря этому исключаются недоразумения при переходе на летнее время, поскольку показание истинного времени не изменяется.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Определение терминологии для манипулирования датой и временем, в том числе *составляющих частей даты и времени* и *строки отформатированной даты и времени*.
- Получение текущих даты и времени.
- Вывод на экран строк отформатированной даты и времени с помощью метода `format()` из объекта типа `DateTime`.
- Исследование символов форматирования, распознаваемых методом `format()`.

[1] В разделе “Видоизменение директив конфигурации PHP” приложения А к данной книге поясняется, как настраивать параметры конфигурации.

- Синтаксический анализ даты и времени в абсолютном и относительном формате.
- Расчет одной величины даты и времени относительной другой величины даты и времени.
- Вычисление разности двух дат.
- Пояснение причин, по которым время UTC является удобным для установки часового пояса по умолчанию.

ГЛАВА 16

Управление пакетами

Леность, упоминавшуюся в главе 5 как достоинство, можно поставить на еще более выгодную службу, опираясь на целые пакеты кода, написанные другими. В этой главе поясняется, как пользоваться системой управления пакетами Composer, чтобы находить существующие библиотеки и внедрять их в свои программы.

Если вам приходилось когда-нибудь внедрять сторонние библиотеки без диспетчера пакетов, то вы, должно быть, знаете все стадии данного процесса: загрузку архивного файла, содержащего библиотеку, распаковку этого файла, размещение распакованных файлов в особом месте и, наконец, видоизменение программы таким образом, чтобы она могла находить новые библиотечные файлы.

Система Composer позволяет выполнить весь этот процесс одной командой. А когда появятся новые версии применяемых пакетов, она способна сразу же обновить их автоматически.

Если же вам приходилось раньше пользоваться диспетчером пакетов в других языках программирования (например, `npm` вместе с JavaScript, `gem` вместе с Ruby или `cpan` вместе с Perl), этот опыт пригодится вам при освоении системы Composer, сделав данный процесс более приятным.

Установка системы Composer

Загрузите и запустите на выполнение программу установки Composer, введя следующую команду по приглашению командного процессора в окне терминала:

```
curl -sS https://getcomposer.org/installer | php
```

Если вы пользуетесь ОС Windows, загрузите (по адресу `https://getcomposer.org/Composer-Setup.exe`) и запустите на выполнение программу установки Composer из файла `Composer-Setup.exe`. После успешной установки и запуска Composer на выполнение из командной строки по команде **`php composer.phar`** или просто **`composer`** появится справочный экран, на котором перечислены команды, поддерживаемые в системе Composer.

Ввод пакета в программу на PHP

По команде **`require`** пакет вводится в программу на PHP. Команде **`require`** следует, как минимум, указать имя вводимого пакета. В примере 16.1 в программу на PHP вводится библиотека Swift Mailer.

Пример 16.1. Ввод пакета по команде `require`

```
php composer.phar require swiftmailer/swiftmailer
```

По данной команде загружается пакет, устанавливаются его файлы в каталоге `vendor`, создаваемом в каталоге текущего проекта, а также обновляется файл `composer.json`. В файле `composer.json` отслеживаются установленные пакеты, а также другие параметры настройки для ведения проекта средствами Composer. В системе Composer поддерживается также файл `composer.lock`, предназначенный для отслеживания отдельных версий установленных пакетов.

Как только система Composer установит пакет, останется лишь организовать в программе на PHP ссылку на файл автозагрузки Composer в следующей простой строке кода:

```
require "vendor/autoload.php";
```

Логика в этом файле содержит отображение имен классов на имена файлов. Когда делается ссылка на класс в установленном пакете, эта логика гарантирует загрузку файлов, определяемых в данном классе.

Файл `vendor/autoload.php` нужно загрузить лишь один раз независимо от количества установленных пакетов. Строка кода `require "vendor/autoload.php";` обычно вводится в самом начале программ, опирающихся на пакеты, установленные средствами Composer. В примере 16.2 эта строка кода демонстрируется в контексте применения библиотеки Swift Mailer для составления сообщения, как обсуждается более подробно в главе 17.

Пример 16.2. Применение библиотеки, установленной средствами Composer

```
// дать интерпретатору PHP команду загрузить логику Composer
// для обнаружения классов require 'vendor/autoload.php';
// Класс Swift_Message теперь доступен автоматически
$message = Swift_Message::newInstance();
$message->setFrom('julia@example.com');
$message->setTo(array('james@example.com' => 'James Beard'));
$message->setSubject('Delicious New Recipe');
$message->setBody(<<<_TEXT_
Dear James,

You should try this: puree 1 pound of chicken with two pounds
of asparagus in the blender, then drop small balls of the mixture
into a deep fryer. Yummy!
Love,
Julia

_TEXT_
);
```

Если программа на PHP отслеживается в системе контроля версий исходного кода (см. главу 14), следует предпринять ряд дополнительных шагов, чтобы эта система нормально взаимодействовала с системой Composer. Во-первых, необходимо включить оба файла, `composer.json` и `composer.lock`, в состав тех файлов, которые отслеживаются системой контроля версий исходного кода. Это необходимо для того, чтобы кто-нибудь другой мог снять программу с регистрации в системе контроля версий исходного кода и установить те же самые версии пакетов, которые применяются в данной программе. И во-вторых, следует исключить каталог `vendor` из числа отслеживаемых в системе контроля версий исходного кода. Весь код, находящийся в каталоге `vendor`, ведется системой Composer. Если обновляется версия пакета, то отслеживать изменения требуется только

в файлах `composer.json` и `composer.lock`, но не в каждом из файлов, которые находятся в каталоге `vendor` и могли измениться.

Если в системе контроля версий исходного кода отслеживаются только файлы `composer.json` и `composer.lock`, а не весь каталог `vendor`, кто-нибудь другой, желающий снять код программы с регистрации в данной системе, должен выполнить команду **`php composer.phar install`**, чтобы все версии требующихся пакетов оказались доступными в нужном месте.

Поиск пакетов

Безусловно, практическое применение системы управления пакетами вроде Composer зависит от применения устанавливаемых пакетов. Так как же найти подходящие для установки пакеты, чтобы разрешить имеющие затруднения? Наиболее широко употребительным в системе Composer является хранилище пакетов Packagist, на сайте которого по адресу `https://packagist.org/` пакеты индексированы для удобства просмотра и загрузки.

Допустим, что требуется геокодировать ряд адресов, т.е. определить долготу и широту, соответствующие каждому отдельному адресу. Если ввести **`geocode`** в поле поиска, находящемся в верхней части начальной страницы веб-сайта Packagist, а затем щелкнуть на расположенной рядом стрелке, чтобы отсортировать результаты поиска по количеству загрузок, то в ответ будет сразу же получен целый ряд результатов, как показано на рис. 16.1.

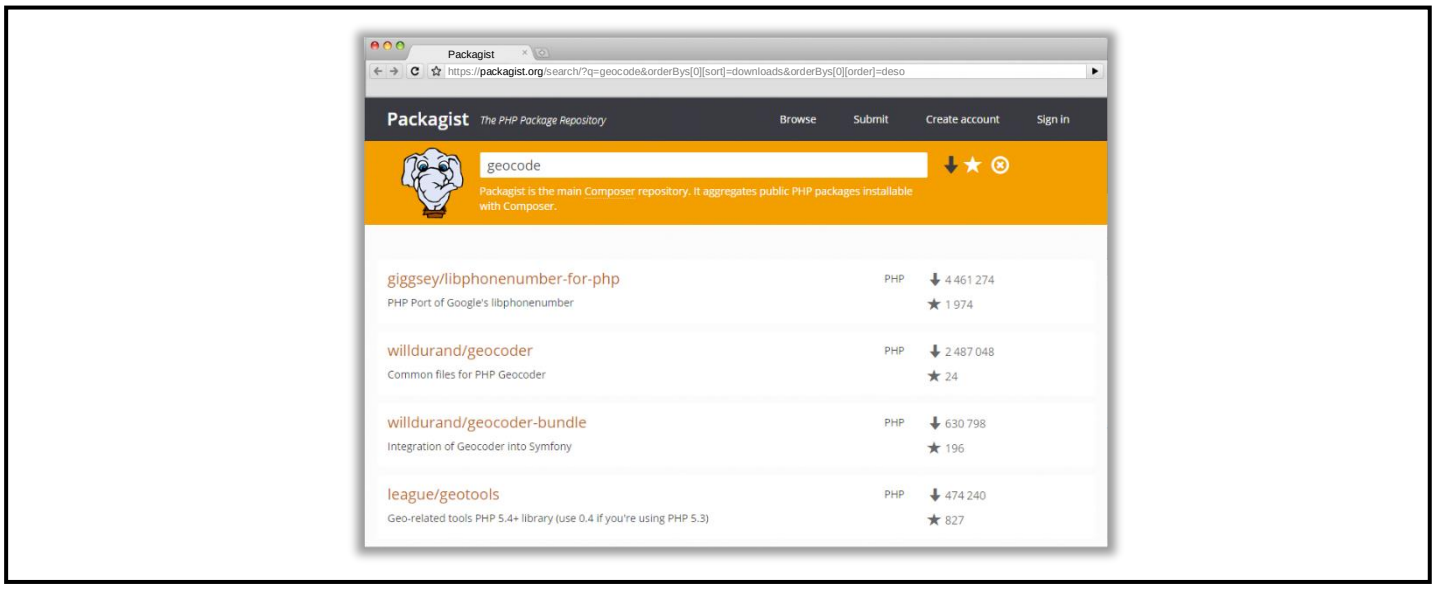


Рис. 16.1: Поиск пакетов на веб-сайте `packagist.org`

Чтобы ввести один из найденных пакетов в свой проект, достаточно набрать одну из следующих строк кода:

```
php composer.phar require willdurand/geocoder
```

ИЛИ

```
php composer.phar require league/geotools
```

На рис. 16.2 показано, что происходит по приглашению в окне терминала при установке пакета `willdurand/geocoder`.

Как показано на рис. 16.2, вверху, процесс установки найденного пакета начинается по команде **`php composer.phar require willdurand/geocoder`**. Затем система Composer выявляет самую последнюю устойчивую версию пакета (3.2) и те зависимости, которые требуется установить, а затем загружает и устанавливает эти пакеты. Так, одна из зависимостей (`http-adapter`) предполагает, но не требует, чтобы был установлен ряд других пакетов. Поэтому система Composer выводит на экран сообщения об этих пакетах вместо их установки.

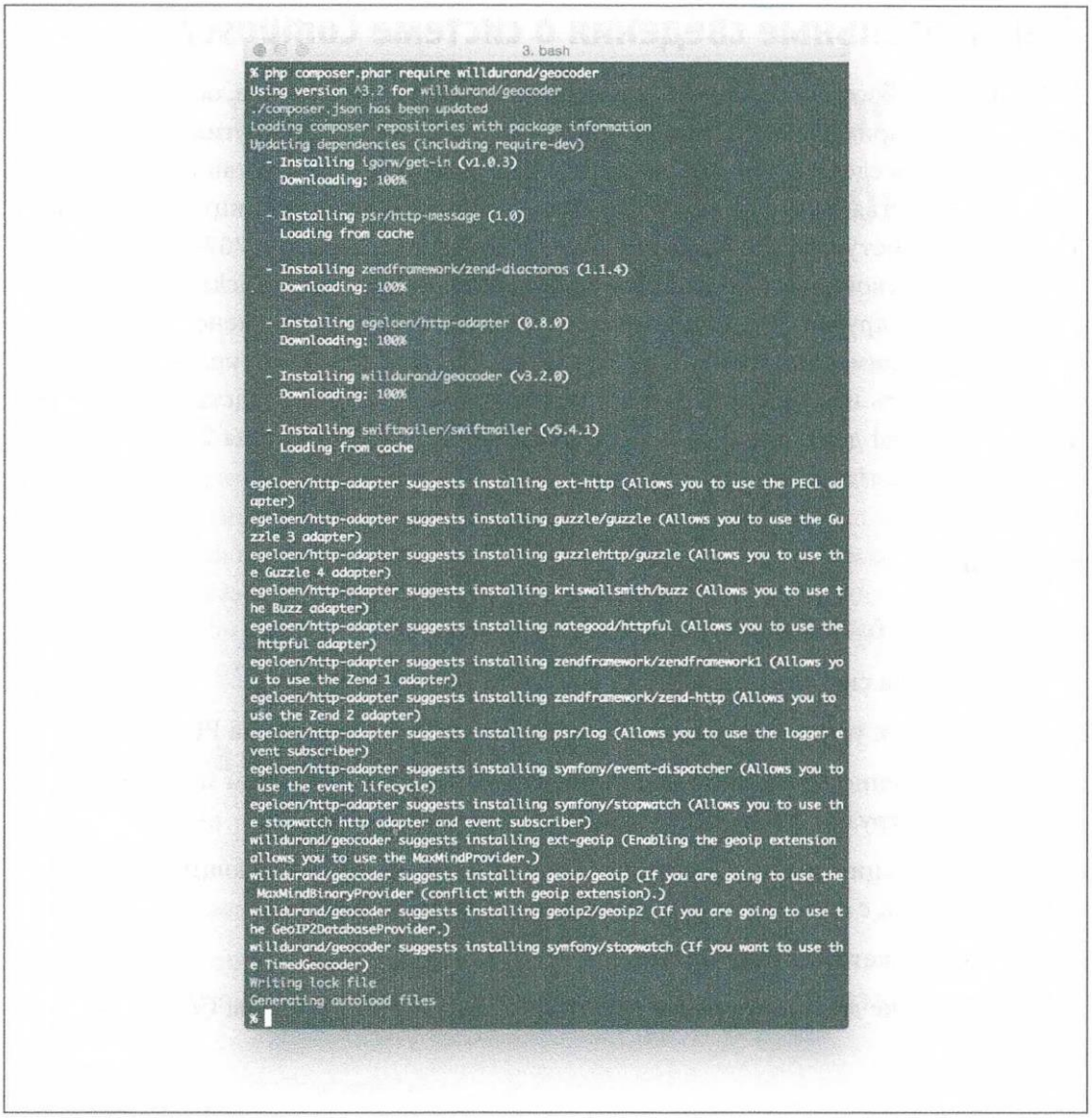


Рис. 16.2: Установка пакета **willdurand/geocoder**

Дополнительные сведения о системе Composer

В кратком обзоре системы Composer, приведенном в этой главе, основное внимание уделяется применению в вашем проекте кода, написанного другими. Если же вас интересует, как сделать ваш код доступным другим людям для установки в виде библиотеки средствами Composer, обратитесь за справкой на страницу документации по Composer, доступную по адресу `https://getcomposer.org/doc/02-libraries.md`. Опубликовать свой пакет можно свободно и легко на веб-сайте Packagist.

Имеются и другие хранилища пакетов, доступные для применения в системе Composer. Например, из хранилища WordPress Packagist (`https://wpackagist.org/`) можно загрузить и установить темы и подключаемые модули средствами Composer. А пакеты Drupal доступны через систему Composer из хранилища Drupal Packagist (`https://packagist.drupal-composer.org/`).

Резюме

В этой главе были рассмотрены следующие вопросы.

- Установка системы управления пакетами Composer.

- Загрузка и установка пакета для применения в программе на PHP.
- Обеспечение доступа из кода программы на PHP к файлам пакетов посредством загрузки файла `autoload.php` из системы Composer.
- Организация взаимодействия программ на PHP, пользующихся системой Composer, с системой контроля версий исходного кода.
- Поиск пакетов для установки.
- Получение дополнительных сведений о применении системы Composer.

ГЛАВА 17

Отправка сообщений по электронной почте

Большая часть взаимодействия с пользователями происходит через веб-страницы. Тем не менее полезно отправлять и получать сообщения по электронной почте. Электронная почта служит отличным средством для отправки обновлений, подтверждений заказов и ссылок, дающих пользователям возможность восстановить или сменить свои пароли. В этой главе излагаются основы применения библиотеки Swift Mailer для отправки сообщений по электронной почте.

Библиотека Swift Mailer

Прежде всего установите библиотеку Swift Mailer средствами Composer по следующей команде:

```
php composer.phar require swiftmailer/swiftmailer
```

В итоге библиотека Swift Mailer будет доступна для применения, при условии, что вы введете в самом начале своей программы на PHP следующую стандартную строку кода:

```
require "vendor/autoload.php";
```

Библиотека Swift Mailer представляет сообщения в виде объектов класса `Swift_Message`. Чтобы составить сообщение электронной почты, необходимо сначала создать один из объектов данного класса, а затем вызвать для него методы с целью сформировать сообщение. Далее объект готового сообщения передается экземпляру класса `Swift_Mailer` для последующей отправки. В свою очередь, экземпляр класса `Swift_Mailer` настраивается средствами специального класса `Swift_Transport`. Этот транспортный класс воплощает логику отправки сообщений путем подключения к удаленному серверу или применения утилит электронной почты на локальном сервере. В примере 17.1 демонстрируется порядок составления простого сообщения электронной почты из темы, адресов отправителя и получателя и, наконец, тела самого сообщения, набранного простым текстом.

Пример 17.1. Составление сообщения электронной почты

```
$message = Swift_Message::newInstance();
$message->setFrom('julia@example.com');
$message->setTo(array('james@example.com' => 'James Bard'));
$message->setSubject('Delicious New Recipe');
$message->setBody (<<<_TEXT_
Dear James,
```

```
You should try this: puree 1 pound of chicken with two pounds
of asparagus in the blender, then drop small balls of the mixture
into a deep fryer. Yummy!
Love,
Julia

_TEXT_
);
```

В качестве аргументов методам `setFrom()` и `setTo()` можно передать адреса электронной почты, представленные в виде символьных строк или пар "ключ-значение", состоящих из адресов электронной почты и полных имен адресатов. Чтобы указать несколько получателей, достаточно передать массив, содержащий любое сочетание символьных строк (адресов) и или пар "ключ-значение" (адресов и полных имен адресатов).

В методе `setBody()` задается тело сообщения, набранное простым текстом. Чтобы ввести вариант HTML-разметки в тело сообщения, следует вызвать метод `addPart()` с альтернативным телом в качестве первого аргумента, а также подходящий тип MIME в качестве второго аргумента, как показано в следующем примере кода:

```
$message->addPart(<<<_HTML_
<p>Dear James,</p>
<p>You should try this:</p>
<ul>
<li>puree 1 pound of chicken with two pounds of asparagus
    in the blender</li>
<li>drop small balls of the mixture into a deep fryer.</li>
</ul>

<p><em>Yummy!</em></p>

<p>Love,</p>
<p>Julia</p>

_HTML_
    // тип MIME в качестве второго аргумента
    , "text/html");
```

Сообщению требуется средство доставки почтой, а средству доставки почтой — транспортный протокол, чтобы знать, как отправлять сообщение. В этой главе демонстрируется применение транспортного протокола SMTP (Simple Mail Transfer Protocol — простой протокол пересылки электронной почты), осуществляющего подключение к стандартному серверу электронной почты для отправки сообщения. Чтобы создать экземпляр класса `Swift_SmtpTransport`, нужно знать имя хоста и порт сервера электронной почты, а, возможно, также имя пользователя и пароль. В поисках этой информации обратитесь к своему системному администратору или выясните параметры настройки своей учетной записи в программе электронной почты.

В примере 17.2 демонстрируется создание сначала объекта класса `Swift_Smtp Transport` с помощью образца SMTP-сервера с именем хоста **`smtp.example.com`** и портом **`25`**, а затем объекта класса `Swift_Mailer`, пользующегося этим объектом в качестве транспортного протокола.

Пример 17.2. Создание объектов, требующихся для применения транспортного протокола SMTP

```
$transport =
    Swift_SmtpTransport::newInstance('smtp.example.com', 25);
$mailer = Swift_Mailer::newInstance($transport);
```

Как только объект класса `Swift_Mailer` будет получен, его методу `send()` можно передать объект класса `Swift_Message`, чтобы отправить подготовленное сообщение:

```
$mailer->send($message);
```

В библиотеке Swift Mailer поддерживается намного больше функциональных возможностей, чем описано выше. Средствами этой библиотеки можно, в частности, прикреплять файлы к сообщениям, добавлять произвольные заголовки в сообщения, запрашивать уведомления о получении сообщений, подключаться к почтовым серверам по сетевому протоколу SSL и делать многое другое. Подробнее обо всех этих возможностях библиотеки Swift Mailer можно узнать из ее оперативно доступной документации по адресу `http://swiftmailer.org/docs/introduction.html`.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Установка библиотеки Swift Mailer.
- Назначение объектов, представляющих в библиотеке Swift Mailer сообщение, средство доставки и транспортный протокол электронной почты.
- Создание объекта класса `Swift_Message` и настройка его содержимого.
- Создание объекта класса `Swift_SmtpTransport` и его применение вместе с объектом класса `Swift_Mailer`.
- Отправка сообщения по электронной почте.
- Изучение дополнительных возможностей библиотеки Swift Mailer в ее документации.

ГЛАВА 18

Каркасы

Каркас приложений представляет собой набор функций, классов и условий, упрощающих решение типичных задач программирования. У многих языков программирования имеются свои общедоступные каркасы, и в этом отношении язык PHP не является исключением. В этой главе дается краткий обзор трех наиболее употребительных в PHP каркасов. Эти каркасы упрощают разработку с нуля и до вполне работоспособного веб-приложения.

Каркасы, предназначенные для веб-разработки, обычно предоставляют стандартные способы решения, по крайней мере, следующих задач.

Маршрутизация — преобразование URL с пользовательскими запросами в конкретные методы или функции, предназначенные для формирования ответов.

Объектно-реляционное преобразование — способ трактовать строки из таблицы базы данных в виде объектов в прикладном коде и предоставлять для этих объектов методы, видоизменяющие содержимое базы данных.

Управление пользователями — стандартные механизмы для ведения информации о пользователях приложения и предоставления пользователям прав выполнять определенные операции.

Применяя каркас, можно сэкономить время на реализации всех его функциональных возможностей собственными силами. Имеется также возможность сразу же подключить к работе над проектом новых разработчиков, знакомых с каркасом. Правда, для работы с каркасом придется потратить время на его изучение и адаптацию к его условиям достижения поставленных целей.

В этой главе рассматриваются следующие три каркаса: Laravel, Symfony и Zend Framework. Каждый из них предоставляет совершенно разные решения проблемы “каркаса”. Они отличаются порядком своей установки, пояснениями в документации, соблюдением равновесия между простотой и потенциалом, а также возможностями находить дополнительную информацию, зайдя в тупик.

Имеется также немало других каркасов приложений на PHP. Три упомянутых выше каркаса рассматриваются в этой главе потому, что они наиболее употребительны и доступны, хотя это не исключает возможность опробовать какой-нибудь другой каркас. В Интернете можно найти немало рекомендаций по поводу выбора наиболее подходящего каркаса приложений на PHP. Самые последние сведения о каркасах приложений на PHP можно найти на социальном веб-сайте Quora по адресу `https://www.quora.com/topic/PHP-Frameworks` или по критерию поиска **`php framework`** на новостном веб-сайте Hacker News по адресу `https://news.ycombinator.com/` либо на веб-сайте SitePoint по адресу `https://www.sitepoint.com/`.

Laravel

Создатель каркаса Laravel описывает его как средство для тех, кто ценит “изящество, простоту и удобочитаемость” исходного кода. У этого каркаса имеется грамотно составленная документация, живая экосистема пользователей, а также доступные поставщики услуг хостинга и учебные материалы.

Для установки каркаса Laravel достаточно выполнить следующую команду:

```
php composer.phar global require "laravel/instailer=~1.1"
```

Затем следует создать новый проект, в котором предполагается применять каркас Laravel, а далее — выполнить команду **`laravel new имя_проекта`**, подставив имя своего проекта вместо параметра ***`имя_проекта`***.[1] Так, если выполнить команду **`laravel new menu`**, будет создан каталог `menu` с необходимым кодом и конфигурационным наполнением для нормальной работы каркаса Laravel.

Чтобы увидеть конфигурационное наполнение данного каркаса в действии, запустите встроенный в PHP веб-сервер, указав путь к файлу `server.php` в каталоге своего проекта. Например, по команде

```
php -S localhost:8000 -t menu2/public menu/server.php
```

можно получить доступ к новому проекту Laravel в подкаталоге `menu` по адресу `http://localhost:8000`.

Маршрутизация в Laravel реализуется в коде из файла `app/Http/routes.php`. При каждом вызове статического метода из класса `Route` каркасу Laravel сообщается, что он должен делать при поступлении HTTP-запроса по определенному методу и пути в URL. Так, в примере 18.1 каркасу Laravel предписывается отвечать на запрос по методу `GET` и пути `/show`.

Пример 18.1. Ввод маршрута в Laravel

```
Route::get('/show', function)) {
    $now = new DateTime();
    $items = [ "Fried Potatoes", "Boiled Potatoes",
               "Baked Potatoes" ];
    return view('show-menu', [ 'when' => $now,
                               'what' => $items ]);
});
```

При вызове метода `Route::get()` из примера 18.1 каркасу Laravel предписывается отвечать на HTTP-запросы по методу `GET` (а не `POST`). В качестве первого аргумента `"/show"` каркасу Laravel сообщается, что в данном вызове метода `Route::get()` он может найти информацию о том, что ему нужно делать, когда происходит обращение по пути `/show` в URL. А в качестве второго аргумента при вызове метода `Route::get()` указывается функция, которую должен выполнить каркас Laravel, чтобы сформировать ответ на HTTP-запрос по методу `GET` и пути `/show`. В этой функции устанавливаются две переменные, `$now` и `$items`, которые затем передаются представлению `show-menu` по ключам `when` и `what`.

Представление служит шаблоном, содержащим логику представления, т.е. того, что должно отображать приложение. Функция `view()` из каркаса Laravel осуществляет сначала поиск файла в заранее определенном месте, а затем выполняет код PHP из этого файла для формирования ответа на запрос. Так, при вызове метода `view('show-menu')` каркасу предписывает искать файл `show-menu.php` в каталоге `resources/views`. Код для данного представления приведен в примере 18.2.

[1] Для этого требуется указать глобальный каталог двоичного кода Composer в системной переменной `$PATH`. Подробнее об этом см. в разделе “Выполнение цикла PHP REPL” главы 19.

Пример 18.2. Представление в Laravel

```
<p> At <?php echo $when->format('g:i a') ?>,
         here is what's available: </p>
<ul>
<?php foreach ($what as $item) { ?>
<li><?php echo $item ?></li>
<?php } ?>
</ul>
```

Представление состоит из обычного кода PHP, размеченного в формате HTML. Любые данные, поступающие из внешнего источника (например, от пользователя или из базы данных), должны быть надлежащим образом экранированы, чтобы предотвратить атаки типа межсайтового выполнения сценариев, как пояснялось в разделе "HTML и JavaScript" главы 7. В каркасе Laravel поддерживается движок шаблонной обработки Blade, намного упрощающий дело, экранируя, в частности, выводимые результаты по умолчанию.

Symfony

Каркас Symfony (`https://symfony.com/`) описывается как набор повторно используемых компонентов и как каркас для проектов веб-приложений. Это означает, что даже если вы и не пользуетесь этим каркасом приложений для маршрутизации запросов или решения других задач веб-разработки, вам ничто не мешает применять отдельные его компоненты для решения таких задач, как шаблонная обработка, ведение файлов конфигурации или отладка кода.

Подобно каркасу Laravel, у каркаса Symfony имеется своя программа, применяемая в режиме командной строки для создания и ведения проектов. Установите и переименуйте на `symfony` программу `installer`, загруженную по адресу `http://symfony.com/installer`. Переместите ее в каталог, доступный по системному пути. В ОС Linux или Mac OS X сделайте программу symfony исполняемой по команде **`chmod a+x /path/to/symfony`**, где **`/path/to/symfony`** — полный путь к месту нахождения программы `symfony`.

Затем создайте новый проект веб-приложения, в котором применяется каркас Symfony, выполните команду **`symfony new`** ***`имя_проекта`***, подставив имя своего проекта вместо ***`имя_проекта`***. Например, по команде

```
symfony new menu
```

будет создан каталог **`menu`** с необходимым кодом и конфигурационным наполнением для нормальной работы каркаса Symfony.

В состав каркаса Symfony входит связующее средство, упрощающее выполнение проекта на встроенном в PHP веб-сервере. Достаточно сменить текущий каталог проекта (например, по команде **`cd menu`**), выполнить команду **`php app/console server:run`** и перейти по адресу `http://localhost:8000/`, чтобы в окне браузера появилась приветственная страница `"Welcome to Symfony"` с подробной диагностической информацией внизу.

В каркасе Symfony маршруты не указываются в одном центральном месте. Вместо этого отдельные классы из каталога `src/AppBundle/Controller` определяют методы, вызываемые по маршрутам, обрабатываемым в веб-приложении. Специальная аннотация в комментарии к методу указывает на маршрут, обрабатываемый данным методом. Так, в примере 18.3 определяется обработчик запросов по методу `GET` и пути `/show`. Его следует разместить в исходном файле `MenuController.php`, находящемся в каталоге `src/AppBundle/Controllers`.

Пример 18.3. Указание маршрута средствами Symfony

```
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Symfony\Component\HttpFoundation\Response;

class MenuController extends Controller {
/**
 * @Route("/show")
 * @Method("GET")
*/
public function showAction()
{
    $now = new \DateTime();
    $items = [ "Fried Potatoes", "Boiled Potatoes",
               "Baked Potatoes" ];
    return $this->render("show-menu.html.twig",
                         [ 'when' => $now,
                           'what' => $items ]);
    }
}
```

Аннотации, указанные в комментарии к коду из примера 18.3 перед определением метода `showAction()`, обозначают обрабатываемый этим методом маршрут: путь `/show` в URL с методом `GET`. А метод `render()` возвращает структуру данных Symfony, содержащую ответ. В качестве первого аргумента этому методу передается имя файла используемого шаблона представления, а в качестве второго аргумента — данные, передаваемые шаблону. В качестве языка шаблонной обработки допускается употреблять простой код PHP, но по умолчанию в Symfony применяется движок шаблонной обработки Twig (`http://twig.sensiolabs.org/`), что и указано в данном примере.

Представления Symfony находятся в каталоге `app/Resources/views`. Это означает, что если передать значение **`show-menu.html.twig`** в качестве первого аргумента методу `render()`, то каркас Symfony буде искать файл `app/Resources/views/show-menu.html.twig` в каталоге текущего проекта. Сохраните в этом месте содержимое представления, приведенное в примере 18.4.

Пример 18.4. Определение представления в Symfony

```
{% extends 'base.html.twig' %}

{% block body %}
<p> At {{ when|date("g:i a") }}, here is what's available: </p>
<ul>
{% for item in what %}
<li>{{ item }}</li>
{% endfor %}
</ul>
{% endblock %}
```

В движке Twig ограничители **`{% %}`** обозначают команду языка шаблонной обработки, а ограничители `{{ }}` — переменную, значение которой (с соответствующим экранированием HTML-разметки) должно быть включено в выводимый результат. К такому синтаксису нужно привыкнуть, хотя Twig является эффективным языком ускоренной шаблонной обработки.

Zend Framework

Каркас Zend Framework (`https://framework.zend.com/`) соответствует определению *набор компонентов* в большей степени, чем два описанных ранее каркаса приложений. С одной стороны, это упрощает внедрение одного или двух компонентов в существующий проект, не согласуясь с определенной структурой файлов или условиями для маршрутизации запросов, а с другой стороны, означает необходимость начинать все с самого начала, что несколько усложняет дело.

Чтобы установить "скелетное" приложение Zend Framework в каталоге `menu`, где содержится все необходимое для применения данного каркаса, выполните следующую команду Composer, введя ее в одной строке:

```
composer create-project --no-interaction --stability="dev"
zendframework/skeleton-application menu
```

Затем выберите встроенный в PHP веб-сервер для нового приложения Zend Framework, перейдите в каталог текущего проекта и выполните такую команду:

```
php -S localhost:8000 -t public/public/index.php
```

Перейдите далее по адресу `http://localhost:8000`, чтобы увидеть начальную страницу, выбираемую по умолчанию для нового приложения.

Каркас Zend Framework организует связанный вместе прикладной код в *модули*. В крупном приложении можно создать модули для отдельных высокоуровневых частей программы. В рассматриваемом здесь примере приложения весь код размещается в уже существующем базовом модуле `Application`. Этот модуль содержит некоторую стандартную логику маршрутизации, преобразующую пути к ресурсам, размещаемым в каталоге `/Application`, в код классов контроллеров, находящихся в конкретном месте файловой системы. В примере 18.5 демонстрируется новый контроллер меню из исходного файла `MenuController.php`. Сохраните этот файл в каталоге `module/Application/src/Application/Controller` своего проекта Zend Framework.

Пример 18.5. Контроллер меню в Zend Framework

```
namespace Application\Controller;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;

class MenuController extends AbstractActionController
{
    public function showAction()
    {
        $now = new \DateTime();
        $items = [ "Fried Potatoes", "Boiled Potatoes",
                   "Baked Potatoes" ];

        return new ViewModel(array('when' => $now,
                                   'what' => $items));
    }
}
```

Затем найдите приведенную ниже часть в файле конфигурации `module/Application/config/module.config.php`, чтобы сообщить каркасу о новом классе.

```
'controllers' => array(
    'invokables' => array(
         'Application\Controller\Index' =>
         'Application\Controller\IndexController'
    ),
),
```

Введите следующую строку в качестве второго элемента в массив `invokables`:

```
'Application\Controller\Menu' =>
     'Application\Controller\MenuController'
```

Не забудьте добавить запятую после строкового значения `'Application\Controller\Index Controller'`, чтобы элементы данного массива имели надлежащий синтаксис. По завершении эта часть файла конфигурации должна выглядеть следующим образом:

```
'controllers' => array(
    'invokables' => array(
          'Application\Controller\Index' =>
          'Application\Controller\IndexController',
          'Application\Controller\Menu' =>
          'Application\Controller\MenuController'
    ),
),
```

Итак, вы создали новый контроллер, и каркас знает, как им пользоваться. Теперь остается лишь ввести представление, чтобы можно было воспроизвести время и информацию о пунктах меню. В каркасе Zend Framework по умолчанию допускается употреблять простой код PHP в качестве языка шаблонной обработки. Сохраните исходный код из примера 18.6 в файле `module/Application /view/application/menu/show.phtml`, размещаемом в каталоге вашего проекта.

Пример 18.6. Представление в каркасе Zend Framework

```
<p> At <?php echo $when->format("g:i a") ?>,
         here is what's available: </p>
<ul>
<?php foreach ($what as $item) { ?>
<li><?php echo $this->escapeHtml($item) ?></li>
<?php } ?>
</ul>
```

Ключи в массиве, передаваемом конструктору в операции `new ViewModel()` из класса контроллера, являются именами локальных переменных в представлении. Благодаря этому значительно упрощается доступ к их значениям. Но поскольку в качестве языка шаблонной обработки употребляется простой код PHP, то HTML-представления и прочие специальные символы не экранируются по умолчанию. В примере 18.6 для экранирования специальных символов в наименовании каждого пункта меню применяется вспомогательный метод `escapeHtml()`.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Представление о каркасах приложений и причинах для их возможного применения.

- Установка каркаса Laravel.
- Создание нового проекта Laravel.
- Ввод маршрута средствами Laravel.
- Ввод представления для маршрута средствами Laravel.
- Установка каркаса Symfony.
- Создание нового проекта Symfony.
- Ввод маршрута средствами Symfony.
- Ввод представления для маршрута средствами Symfony.
- Установка каркаса Zend Framework.
- Создание нового проекта Zend Framework.
- Ввод маршрута средствами Zend Framework.
- Ввод представления для маршрута средствами Zend Framework.

ГЛАВА 19

Применение PHP в режиме командной строки

Как правило, интерпретатор PHP вызывается веб-сервером в ответ на запрос, поступивший от веб-клиента. Но интерпретатор PHP можно также выполнять и в режиме командной строки на своем компьютере. Если вы выполняли все примеры кода, приведенные до сих пор в данной книге, то, скорее всего, запускали интерпретатор PHP в режиме командной строки, пользуясь средствами PHPUnit и Composer.

Написание программы на PHP, предназначенной для применения в режиме командной строки, мало чем отличается от написания такой же программы, предназначенной для применения на веб-сайте. В программе, выполняемой из командной строки, доступны те же самые функции для манипулирования символьными строками, обработки данных в форматах JSON и XML, манипулирования файлами и прочее, но в нее не поступают данные из формы или веб-ресурса по заданному URL. Вместо этого информация поступает в такую программу из аргументов командной строки. Стандартный оператор `print` выводит данные на консоль. В следующем разделе "Написание консольных программ на PHP" представлены основы написания программы, выполняемой из командной строки, а короче — консольной программы.

В состав интерпретатора PHP входит также миниатюрный веб-сервер, который можно вызывать, запуская интерпретатор PHP из командной строки. В разделе "Применение встроенного в PHP веб-сервера" далее в этой главе поясняется, каким образом это делается. Встроенный в PHP веб-сервер удобен для быстрого тестирования.

Еще одним удобным способом применения PHP в режиме командной строки является диалоговый командный процессор, называемый иначе *циклом "чтение-вычисление-вывод"* (Read-Eval-Print Loop — REPL). Эта программа предоставляет приглашение для ввода исходного кода PHP, а затем выполняет этот код и выводит результаты. Что касается исследования функций PHP и быстрого получения удовлетворительного результата, то цикл REPL ничто не может превзойти. В разделе "Выполнение цикла PHP REPL" далее в этой главе поясняется принцип действия встроенного в PHP цикла REPL, а также предоставляется информация о других аналогичных циклах.

Написание консольных программ на PHP

Простая программа на PHP, выводящая данные, оказывается вполне работоспособной в режиме командной строки. В примере 19.1 демонстрируется применение прикладного программного интерфейса Yahoo! Weather API для вывода текущих погодных условий по указанному почтовому индексу.

Пример 19.1. Поиск сводки погоды

```
// Почтовый индекс для поиска сводки погоды
$zip = "98052";

// Запрос YQL для поиска сводки погоды.
// Подробнее см. по адресу https://developer.yahoo.com/weather/
$yqi =
    'select item.condition from weather-forecast where woeid in '.
    '(select woeid from geo.places(1) where text="'.$zip.'")';

// Параметры, которые ожидаются в конечной точке
// запроса YQL веб-службы Yahoo!
$params = array("q" => $yql,
                "format" => "json",
                "env" =>
                     "store://datatables.org/alltableswithkeys");

// Составить URL с запросом YQL, присоединив параметры запроса
$url = "https://query.yahooapis.com/v1/public/yql?"
        . http_build_query($params);
// сделать запрос
$response = file_get_contents($url);
// декодировать ответ в формате JSON $json = json_decode($response);
// выбрать из вложенного ответа в формате JSON объект,
// содержащий полезную информацию
$conditions = $json->query->results->channel->item->condition;
// вывести сводку погоды
print "At {$conditions->date} it is {$conditions->temp} degrees " .
      "and {$conditions->text} in $zip\n";
```

Если сохранить код из примера 19.1 в исходном файле `weather.php`, то его можно выполнить из командной строки по команде `php weather.php`, чтобы получить сводку погоды. Но такую сводку погоды можно получить лишь для почтового индекса **98052**, а иначе придется внести соответствующие коррективы в упомянутый выше исходный файл, что не совсем удобно. Было бы намного удобнее, если бы почтовый индекс можно было предоставить консольной программе в качестве аргумента при ее запуске из командной строки. В примере 19.2 демонстрируется обновленная версия данной программы, в которой аргументы командной строки извлекаются из массива `$_SERVER['argv']`. Консольная версия интерпретатора PHP автоматически заполняет этот массив аргументами, предоставляемыми в командной строке.

Пример 19.2. Доступ к аргументам командной строки

```
// Почтовый индекс для поиска сводки погоды
if (isset($_SERVER['argv'][1])) {
    $zip = $_SERVER['argv'][1];
} else {
    print "Please specify a zip code.\n";
    exit();
}

// Запрос YQL для поиска сводки погоды.
// Подробнее см. по адресу https://developer.yahoo.com/weather/
```

```
$yql =
    'select item.condition from weather-forecast where woeid in ' .
    '(select woeid from geo.places(1) where text="'.$zip.'")';
// Параметры, которые ожидаются в конечной точке
// запроса YQL веб-службы Yahoo!
$params = array("q" => $yql,
                "format" => "json",
                "env" =>
                    "store://datatables.org/alltableswithkeys");

// Составить URL с запросом YQL, присоединив параметры запроса
$url = "https://query.yahooapis.com/v1/public/yql?"
        . http_build_query($params);
// сделать запрос
$response = file_get_contents($url);
// декодировать ответ в формате JSON
$json = json_decode($response);
// выбрать из вложенного ответа в формате JSON объект,
// содержащий полезную информацию
$conditions = $json->query->results->channel->item->condition;
// вывести сводку погоды
print "At {$conditions->date} it is {$conditions->temp} degrees " .
      "and {$conditions->text} in $zip\n";
```

Сохранив код из примера 19.2 в файле `weather2.php`, можно выполнить его по команде **`php weather2 19096`**, чтобы получить сводку погоды для почтового индекса 19096. Следует, однако, иметь в виду, что в качестве первого аргумента служит элемент массива `$_SERVER['argv'][1]`, несмотря на то, что в языке PHP индексирование массивов всегда начинается с нуля (см. раздел “Создание числовых массивов” главы 4). Дело в том, что элемент массива `$_SERVER['argv'][0]` содержит имя выполняемой программы. Так, если выполняется консольная программа по команде **`php weather2.php 19096`**, то ее имя **`weather2.php`** сохраняется в элементе массива `$_SERVER['argv'][0]`.

Применение веб-сервера, встроенного в PHP

Если требуется проверить поведение написанного кода PHP при получении реальных запросов из веб-браузера, то для этой цели проще всего воспользоваться вебсервером, встроенным в интерпретатор PHP.

Встроенный веб-сервер стал доступным в версии PHP 5.4.0.

Достаточно выполнить команду **`php`** с аргументом **`-S`**, предоставляющим имя хоста и номер порта, чтобы получить в свое распоряжение работающий веб-сервер, предоставляющий доступ к файлам из того каталога, где была выполнена команда **`php`**. Например, чтобы запустить веб-сервер, работающий на локальной машине через порт **`8000`**, следует выполнить команду **`php -S localhost:8000`**. Если после запуска указанного веб-сервера перейти по адресу `http://`

`localhost:8000/pizza.php`, этот веб-сервер выполнит код из исходного файла `pizza.php` и отправит полученные результаты обратно веб-браузеру.

Если же веб-серверу не удастся найти файлы, которые он, на ваш взгляд, должен был найти, проверьте каталог, в котором вы находились, выполняя команду **`php -S`**. По умолчанию встроенный в PHP веб-сервер обслуживает файлы из того каталога, где была выполнена команда **`php -S`**, а также из находящихся в нем подкаталогов. Чтобы предоставить другой корневой каталог документов, в команду **`php`** следует ввести аргумент **`-t`**. Например, по команде

```
php -S localhost:8000 -t /home/mario/web
```

обслуживаются файлы, находящиеся в каталоге `/home/mario/web` по адресу `http://localhost:8000`.

Встроенный в PHP веб-сервер не делает ничего особенного, чтобы отобразить URL на имена файлов. Он просто ищет имя файла в своем базовом каталоге, как указано в URL. Если же в URL отсутствует имя файла, веб-сервер попытается найти файлы `index.php` и `index.html`, прежде чем сообщить, что файл не найден.

Встроенный в PHP веб-сервер одновременно обрабатывает только один запрос. Поэтому он лучше всего подходит для тестирования функциональных возможностей и экспериментов на той машине, где ведется веб-разработка. Этот веб-сервер намного проще активизировать, чем крупный веб-сервер типа Apache или nginx, который необходимо конфигурировать, хотя его и нельзя считать полноценным сервером. Когда же настанет черед развертывать прикладной код в рабочей среде, следует воспользоваться тем веб-сервером, который способен удовлетворить требования общего пользования в отношении масштабирования и безопасности.

Выполнение цикла PHP REPL

Встроенный в PHP веб-сервер удобен для быстрой проверки написанного кода PHP в действии. Еще одним удобным инструментальным средством для исследования и тестирования служит встроенный в PHP цикл REPL. Достаточно выполнить команду **`php -a`**, чтобы появилось приглашение **`php >`**, по которому можно ввести какой-нибудь код и сразу же увидеть на экране результаты его выполнения (пример 19.3).

Пример 19.3. Применение цикла PHP REPL

```
% php -a
Interactive shell
php > print strlen("mushrooms");
9
php > $releases = simplexml_load_file(
      "https://secure.php.net/releases/feed.php");
php > print $releases->entry[0]->title;
PHP 7.0.5 released!
php >
```

В примере 19.3 первоначально выводится приглашение **`%`** командного процессора Unix, по которому вводится команда **`php -a`** для выполнения встроенного в PHP цикла REPL. В итоге цикл REPL выводит сообщение `"Interactive shell"` (Диалоговый командный процессор) и приглашение **`php >`**.

В этом цикле выполняется все, что вы введете, нажав клавишу <Enter> или <Return>, а на экран сразу же выводятся результаты. Так, если ввести следующую строку кода:

```
print strlen("mushrooms");
```

а затем нажать клавишу <Enter> или <Return>, в цикле REPL будет вызвана функция `strlen ("mushrooms")`, передающая результаты своего выполнения оператору `print`, который выведет на экран результат **9**, обозначающий количество символов в заданной строке. Не забывайте завершать вводимую строку кода знаком **;**. Ведь код PHP, вводимый в цикле REPL, следует тем же правилам синтаксиса, что и код PHP, написанный в обычной программе.

Если вы просто введете код `strlen("mushrooms");`, он будет выполнен без ошибок, но вы не увидите на экране никаких результатов, прежде чем появится очередное приглашение **php >**. В цикле PHP REPL отображаются только те результаты, которые выводятся на экран при выполнении введенного кода PHP.

В цикле PHP REPL значения переменных запоминаются в промежутках между выполняемыми командами. Так, если ввести следующую строку кода:

```
$releases = simplexml_load_file(https://secure.php.net/releases/feed.php");
```

то с помощью функции `simplexml_load_file()` по указанному URL будет извлечен XML-документ, а результаты, полученные в виде объекта SimpleXML, — сохранены в переменной `$releases`.[1] Расширение SimpleXML предоставляет иерархию объектов, соответствующую структуре возвращаемого XML-документа, и поэтому значение элемента разметки `title`, расположенного ниже первого элемента разметки `entry`, находящегося ниже самого верхнего элемента XML-разметки, равно `$releases->entry[0]->title`. Так, при выполнении кода из примера 19.3 первым в ленте подачи информации о выпусках PHP оказался элемент PHP 7.0.5.

Помимо встроенного в PHP цикла REPL, имеются и другие аналогичные циклы. Характерный тому пример — консольное инструментальное средство PsySH (`http://psysh.org/`), которое можно установить средствами Composer по следующей команде:

```
php composer.phar global require psy/psysh
```

Модификатор `global`, указанный перед командой `require`, предписывает системе Composer установить инструментальное средство PsySH не в каталоге любого отдельного пакета, а в каталоге системы Composer. В Mac OS X и Linux это каталог **`.composer`**, расположенный в начальном каталоге, а в Windows — каталог `AppData\Roaming\Composer`, находящийся в начальном каталоге. Так, если войти в систему под именем пользователя `squidsy`, системным для Composer окажется каталог `/Users/squidsy/.composer` в Mac OS X и Linux, тогда как в Windows — каталог `C:\Users\squidsy\AppData\Roaming\Composer`.

Конкретная программа `psysh` размещается в каталоге `vendor/bin`, находящемся в каталоге системы Composer. Следовательно, чтобы запустить эту программу на выполнение из командной строки, необходимо ввести полный путь к ней (например, `/Users/squidsy/.composer/vendor/bin /psysh`) или ввести каталог `vendor/bin` в системную переменную `$PATH`, в которой указываются каталоги, где по умолчанию осуществляется поиск программ по введенным их именам.

Запустив программу `psysh` на выполнение, вы получите приглашение ввести какой-нибудь код PHP. В отличие от встроенного в PHP цикла REPL, эта программа выводит на экран значение, вычисляемое во введенной строке кода, даже если она не содержит оператор `print`. А разнотипные переменные выделяются в исходном коде разным цветом.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Выполнение написанных на PHP программ из командной строки.
- Доступ к аргументам командной строки из программы на PHP.

[1] Расширение SimpleXML (`http://www.php.net/simplexml`) упрощает обработку XML-документов в PHP.

- Выполнение написанных на PHP программ на встроенном в PHP веб-сервере.
- Выполнение команд во встроенном в PHP цикле REPL.
- Установка и запуск инструментального средства PsySH на выполнение из глобального каталога системы Composer.

ГЛАВА 20

Интернационализация и локализация

Как упоминалось в разделе “Текст” главы 2, символьные строки в языке PHP состоят из последовательностей байтов, а отдельный байт может состоять из 256 возможных двоичных значений. Это означает, что представить текст, в котором употребляются только символы в коде US-ASCII (т.е. буквы английского алфавита, цифры и обычные знаки препинания), в коде PHP не составит особого труда. Но для обработки текста, содержащего другие символы, придется предпринять дополнительные шаги.

Стандарт на Юникод (Unicode) определяет порядок кодировки многих тысяч символов, которые могут употребляться в тексте. Помимо букв различных алфавитов (кириллицы, латинского, греческого, арабского, китайского, японского), этот стандарт определяет самые разные знаки и символы. Так, в кодировке *UTF-8* определяются байты, представляющие каждый символ. В частности, каждая буква английского алфавита представлена только одним байтом. Но для представления других символов может потребоваться два, три или даже четыре байта.

В этой главе представлены основы обработки в программах на PHP многобайтовых символов в кодировке UTF-8. В следующем разделе “Манипулирование текстом” поясняются основные способы манипулирования текстом, в том числе определение длины символьных строк и извлечение подстрок. Ниже, в разделе “Сортировка и сравнение”, будет показано, как сортировать и сравнивать символьные строки с учетом правил расположения символов в разных языках. А в разделе “Локализация выводимых результатов” представлены примеры применения имеющихся в PHP средств для форматирования сообщений, чтобы отображать в программе информацию на том языке, который предпочитает пользователь.

Примеры кода, приведенные в данной главе, опираются на функции PHP из расширений `mbstring` и `intl`. Так, в примерах, приведенных в разделе “Манипулирование текстом”, для функций, имена которых начинаются с префикса **`mb_`**, требуется расширение `mbstring`. А в примерах, приведенных в разделах “Сортировка и сравнение” и “Локализация выводимых результатов”, для классов `Collator` и `MessageFomatter` требуется расширение `intl`. В свою очередь, расширение `intl` опирается на стороннюю библиотеку ICU (`http://site.icu-project.org/`). Если эти расширения недоступны, обратитесь за помощью в их установке к своему системному администратору или поставщику услуг хостинга или же сделайте это самостоятельно, следуя инструкциям, приведенным в приложении А.

Манипулирование текстом

Функция `strlen()` подсчитывает только количество байтов, сообщая неверные результаты, когда для представления символа требуется не один байт. Чтобы правильно подсчитать количество

символов в строке независимо от количества требующихся для него байтов, следует воспользоваться функцией `mb_strlen()`, как показано в примере 20.1.

Пример 20.1. Определение длины символьной строки

```
$english = "cheese";
$greek = "tupl";

print "strlen() says " . strlen($english) . " for $english and " .
      strlen($greek) . " for $greek.\n";
print "mb_strlen() says " . mb_strlen($english) . " for $english and " .
      mb_strlen($greek) . " for $greek.\n";
```

Для представления каждой из букв греческого алфавита требуются два байта, и поэтому при выполнении кода из примера 20.1 на экран выводится следующий результат:

```
strlen() says 6 for cheese and 8 for tupl.
mb_strlen() says 6 for cheese and 4 for tupl.

[ Слово "сыр" по-английски состоит из 6 байтов,
 а по-гречески - из 8 байтов ]
```

Операции, зависящие от расположения символов в строках (например, извлечение подстрок), должны также выполняться с учетом символьного, а не байтового представления, когда в тексте употребляются многобайтовые символы. Так, в примере 2.12 демонстрируется применение функции `substr()` для извлечения первых 30 байт из сообщения, введенного пользователем. А в примере 20.2 для аналогичной операции демонстрируется применение функции `mb_substr()`.

Пример 20.2. Извлечение подстроки

```
$message = "In Russia, I like to eat каша and drink квас";
print "substr() says: " . substr($message, 0, 30) . "\n";
print "mb_substr() says: " . mb_substr($message, 0, 30) . "\n";
```

При выполнении кода из примера 20.2 на экран выводится следующий результат:

```
substr() says: In Russia, I like to eat ка
mb_substr() says: In Russia, I like to eat каша
```

Первая выведенная на экран строка, демонстрирующая результат выполнения функции `substr()`, выглядит совершенно испорченной! Ведь для представления каждой буквы кириллицы требуется не один байт, и поэтому последовательность из 30 байт, извлекаемых из указанной символьной строки, оканчивается посредине последовательности байт, представляющих отдельный символ (в данном случае букву ш русского алфавита). А при извлечении тех же самых 30 байт с помощью функции `mb_substr()` их последовательность правильно оканчивается на границе символа, что и демонстрирует вторая строка результата, выведенного на экран.

Прописные и строчные буквы также различаются по-разному в разных наборах символов. Так, функции `b_strtolower()` и `mb_strtoupper()` предоставляют варианты результатов, возвращаемых функциями `strtolower()` и `strtoupper()`, но с учетом представления символов. В примере 20.3 показано применение этих функций на практике.

Пример 20.3. Смена регистра букв

```
$english = "Please stop shouting.";
$danish = "Venligst stoppe rÅben.";
$vietnamese = "Hay dung la hét.";

print "strtolower() says: \n";
print " " . strtolower($english) . "\n";
print " " . strtolower($danish) . "\n";
print " " . strtolower($vietnamese) . "\n";

print "mb_strtolower() says: \n";
print " " . mb_strtolower($english) . "\n";
print " " . mb_strtolower($danish) . "\n";
print " " . mb_strtolower($vietnamese) . "\n";

print "strtoupper() says: \n";
print " " . strtoupper($english) . "\n";
print " " . strtoupper($danish) . "\n";
print " " . strtoupper($vietnamese) . "\n";

print "mb_strtoupper() says: \n";
print " " . mb_strtoupper($english) . "\n";
print " " . mb_strtoupper($danish) . "\n";
print " " . mb_strtoupper($vietnamese) . "\n";
```

При выполнении кода из примера 20.3 на экран выводится следующий результат:

```
strtolower() says:
    please stop shouting.
    venligst stoppe r◆ben.
    h?y dung la h◆t.
mb_strtolower() says:
    please stop shouting,
    venligst stoppe rÅben.
    hay dung la hét.
strtoupper() says:
    PLEASE STOP SHOOTING.
    VENLIGST STOPPE RÅBEN.
    HãY D{NG LA HéT.
mb_strtoupper() says:
    PLEASE STOP SHOOTING.
    VENLIGST STOPPE RABEN.
    HAY DONG LA HËT.
```

Функции `strtoupper()` и `strtolower()` оперируют отдельными байтами и поэтому не заменяют целые многобайтовые символы верными эквивалентами подобно функциям `mb_strtoupper()` и `mb_strtolower()`.

Сортировка и сравнение

Встроенные в PHP функции сортировки и сравнения фрагментов текста оперируют отдельными байтами, следуя порядку расположения букв в английском алфавите. Чтобы выполнять сортировку

и сравнение фрагментов текста с учетом многобайтового представления символов, рекомендуется обратиться к классу `Collator`.

Прежде всего необходимо построить объект класса `Collator`, передать его конструктору *строку языкового стандарта*, в которой указываются конкретная страна и язык, а также правила применения указанного языкового стандарта в классе `Collator`. Формирование содержимого строки языкового стандарта имеет немало особенностей (подробнее об этом можно узнать по адресу `http://userguide.icu-project.org/locale`), но, как правило, данная строка состоит из двухбуквенного кода языка, знака подчеркивания `_` и двухбуквенного кода страны. Например, строка `"en_US"` обозначает американский английский языковой стандарт, строка `"fr_BE"` — бельгийский французский языковой стандарт, а строка `"ko_KR"` — южнокорейский языковой стандарт. Коды языка и страны, предоставляемые в строке языкового стандарта, позволяют учесть особенности употребления одного и того языка в разных странах.

Метод `sort()` из класса `Collator` делает то же самое, что и встроенная в PHP функция `sort()`, но с учетом языкового стандарта. Этот метод сортирует массив значений на месте. В примере 20.4 показано, каким образом действует эта функция.

Пример 20.4. Сортировка массивов

```
// Американский английский
$en = new Collator('en_US');

// Датский
$da = new Collator('da_DK');

$words = array('absent','Åben', 'zero');

print "Before sorting: " . implode(', ', $words) . "\n";

$en->sort($words);
print "en_US sorting: " . implode(', ', $words) . "\n";

$da->sort($words);
print "da_DK sorting: " . implode(', ', $words) . "\n";
```

В примере 20.4, согласно правилам, принятым в американском английском языке, датское слово `åben` ставится перед английским словом `absent`, а буква `å` сортируется в конце алфавита. Поэтому слово `åben` оказывается в конце массива.

В классе `Collator` имеется также метод `asort()`, являющийся аналогом встроенной в PHP функции `asort()`, а метод `compare()` действует аналогично функции `strcmp()`. Этот метод возвращает значение **`-1`**, если первая указанная символьная строка сортируется прежде второй символьной строки; значение **`0`**, если обе символьные строки равны; или значение **`1`**, если первая символьная строка сортируется после второй символьной строки.

Локализация выводимых результатов

Веб-приложение, которым пользуются люди по всему миру, должно не только правильно оперировать наборами символов, но и формировать сообщения на разных языках. Для одного пользователя оно должно выводить сообщение `"Click here"`, для другого — `"Cliquez ici"`, а для третьего — `"Щелкните здесь"`. Класс `MessageFormatter` помогает формировать сообщения, надлежащим образом локализованные на разных языках.

Прежде всего необходимо составить каталог сообщений. Это перечень переведенных сообщений по каждому из поддерживаемых языковых стандартов. В этом каталоге могут находиться простые

символьные строки вроде "Click here", или же они могут содержать маркеры для вставки значений. Например, в символьной строке "My favorite food is {0}" маркер {0} следует заменить конкретным словом.

В крупном веб-приложении могут присутствовать сотни разных элементов в каталоге сообщений по каждому языковому стандарту. Чтобы пояснить, каким образом действует класс MessageFormatter, в примере 20.5 демонстрируется ряд элементов из каталога сообщений.

Пример 20.5. Определение каталога сообщений

```
$messages = array();
$messages['en_US'] = array('FAVORITE_FOODS' =>
                                    'My favorite food is {0}',
                           'COOKIE' => 'cookie',
                           'SQUASH' => 'squash');
$messages['en_GB'] = array('FAVORITE_FOODS' =>
                                    'My favourite food is {0}',
                           'COOKIE' => 'biscuit',
                           'SQUASH' => 'marrow');
```

Ключи в массиве $messages являются строками языковых стандартов, а значения — сообщениями, переведенными по соответствующим языковым стандартам и индексированными по тем ключам, по которым в дальнейшем будет происходить обращение к этим сообщениям. Чтобы сформировать сообщение, локализованное на конкретном языке, следует создать новый объект класса MessageFormatter, предоставив его конструктору языковой стандарт и формат сообщения, как показано в примере 20.6.

Пример 20.6. Локализация сообщения

```
$fmtfavs = new MessageFormatter('en_GB',
                                $messages['en_GB']['FAVORITE_FOODS']);
$fmtcookie = new MessageFormatter('en_GB',
                                  $messages['en_GB']['COOKIE']);

// В результате следующего вызова возвращается слово "biscuit"
$cookie = $fmtcookie->format(array());

// В следующей строке кода выводится предложение
// с подставленным словом "biscuit"
print $fmtfavs->format(array($cookie));
```

При выполнении кода из примера 20.6 на экран выводится следующий результат:

```
My favourite food is biscuit
```

Если формат сообщения содержит фигурные скобки, элементы массива, передаваемого в качестве аргумента методу format(), подставляются вместо значения, указанного в фигурных скобках.

Большую часть работы по выбору подходящих строк языковых стандартов пришлось выполнить в примере 20.6 вручную, и здесь проку от класса MessageFormatter оказалось не так уж и много. Но он действительно помогает, когда требуется отформатировать числа и другие данные с учетом региональных особенностей. В примере 20.7 демонстрируется надлежащая обработка числовых величин и денежных сумм по разным языковым стандартам.

Пример 20.7. Форматирование чисел по языковым стандартам

```
$msg = "The cost is {0,number,currency}.";

$fmtUS = new MessageFormatter('en_US', $msg);
$fmtGB = new MessageFormatter('en_GB', $msg);

print $fmtUS->format(array(4.21)) . "\n";
print $fmtGB->format(array(4.21)) . "\n";
```

При выполнении кода из примера 20.7 на экран выводится следующий результат:

```
The cost is $4.21.
The cost is £4.21.
```

Класс `MessageFormatter` опирается на эффективную библиотеку ICU, и поэтому в нем применяется внутренняя база данных знаков денежных единиц, правил форматирования чисел и организации информации в разных регионах и языках для вывода результатов в надлежащем виде.

Возможности класса `MessageFormatter` намного шире описанных выше. В частности, он позволяет форматировать даты и время, текстовые фразы в единственном и множественном числе, учитывать особенности тех языков, где род определяет написание слова. Подробнее о возможностях данного класса в частности и библиотеки ICU вообще можно узнать из документации, оперативно доступной по адресу `http://userguide.icu-project.org/formatparse`.

Резюме

В этой главе были рассмотрены следующие вопросы.

- Причины, по которым некоторые символы должны быть представлены не одним байтом.
- Определение длины строки в символах, а не в байтах.
- Извлечение подстрок по позиции символа в исходной строке.
- Надежная смена регистра букв.
- Сортировка текста по языковым стандартам.
- Сравнение символьных строк по языковым стандартам.
- Локализация выводимых результатов по языковым стандартам.

ПРИЛОЖЕНИЕ A

Установка и конфигурирование интерпретатора PHP

Для написания программ на PHP требуется интерпретатор PHP, позволяющий превратить их исходный текст в конкретные интерактивные веб-страницы. Чтобы воспользоваться интерпретатором PHP, проще всего подписаться на недорогие или бесплатные услуги веб-хостинга у их поставщика, предоставляющего доступ к интерпретатору PHP. Но этот интерпретатор можно выполнять и на своем компьютере.

Применение интерпретатора PHP, предоставляемого поставщиком услуг веб-хостинга

Если у вас уже имеется учетная запись у поставщика услуг веб-хостинга, значит, у вас, вероятнее всего, имеется также доступ к серверу с поддержкой PHP. В настоящее время редкий поставщик услуг веб-хостинга не предоставляет поддержку PHP. Как правило, поставщики хостинга конфигурируют свои серверы таким образом, чтобы файлы, имена которых оканчиваются на **`.php`**, трактовались как программы на PHP. Чтобы выяснить, поддерживается ли PHP на размещаемом веб-сайте, необходимо сначала сохранить код из примера A.1 в файле `phptest.php` на своем сервере.

Пример A.1. Тестовая программа на PHP

```
<?php print "PHP enabled"; ?>
```

Загрузите этот файл в свой браузер, перейдя на свой сайт по указанному URL (например, `http://www.example.com/phptest.php`). Если появится сообщение `"PHP enabled"`, значит, на хосте, где размещается сайт, поддерживается PHP. Если же появится все содержимое страницы (`<?php print "PHP enabled"; ?>`), значит, поставщик услуг хостинга не предоставляет поддержку PHP. В таком случае нужно выяснить, не активизировал ли он поддержку PHP для файла с другим расширением или сделал какой-нибудь другой нестандартный выбор конфигурации.

Установка интерпретатора PHP

Устанавливать интерпретатор PHP на своем компьютере целесообразно в том случае, если отсутствует учетная запись у поставщика услуг хостинга или просто требуется поэкспериментировать с PHP, не делая свои программы доступными из Интернета. Если вы не пользуетесь услугами хостинга и желаете установить интерпретатор PHP на своем компьютере, следуйте инструкциям,

приведенным в этом разделе. Установив интерпретатор PHP, вы сможете выполнять свои программы, написанные на PHP.

Чтобы установить интерпретатор PHP, достаточно загрузить ряд файлов, разместив их в нужных местах на своем компьютере. Возможно, потребуется также сконфигурировать свой веб-сервер таким образом, чтобы ему было известно о PHP. В этом разделе приведены инструкции, поясняющие, как это сделать на компьютерах, работающих под управлением ОС Linux и Mac OS X, а также даются ссылки на ресурсы, где поясняется, как устанавливать интерпретатор PHP в ОС Windows. Если в ходе этого процесса у вас возникнут вопросы, обращайтесь за справкой на вебстраницу по адресу `http://ua2.php.net/manual/ru/faq.installation.php`, где приведены ответы на часто задаваемые вопросы по установке интерпретатора PHP.

Установка интерпретатора PHP в Mac OS X

В состав ОС Mac OS X входит устанавливаемая версия PHP 5.5. Но для того чтобы установить более новую версию PHP и иметь возможность легко управлять дополнениями и расширениями, придется установить свой интерпретатор PHP, используя диспетчер пакетов Homebrew. Этот диспетчер помогает устанавливать в Mac OS X программы и библиотеки, от которых они зависят.

Прежде всего установите диспетчер пакетов Homebrew, если этого еще не было сделано. Подробнее о том, как это сделать, можно узнать по адресу `http://brew.sh/`. С другой стороны, можно просто ввести приведенную ниже команду (одной строкой) по приглашению в окне терминала. Если это трудно сделать, перейдите на сайт Homebrew по указанному выше адресу, откуда можно скопировать приведенную ниже команду в буфер обмена и вставить ее в окне терминала.

```
ruby -e "$ (curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Как только диспетчер пакетов Homebrew будет установлен, ему нужно сообщить, где следует искать самую последнюю версию PHP. С этой целью выполните следующие команды:

```
brew tap homebrew/dupes
brew tap homebrew/versions
brew tap homebrew/homebrew-php
```

Далее выполните команду

```
brew install php70
```

чтобы установить версию PHP 7. Вот, собственно, и все!

В конце процесса установки диспетчер пакетов Homebrew выведет на экран массу инструкций по конфигурированию произведенной установки. Уделите особое внимание этим инструкциям, поскольку их нужно придерживаться, чтобы сообщить копии веб-сервера Apache в Mac OS X, где следует искать интерпретатор PHP.

В состав диспетчера пакетов Homebrew входит целый ряд расширений, в том числе упоминавшиеся в главе 20 расширения `intl` и `mbstring`. Но помимо них, предлагается установить другие расширения PHP. Чтобы получить перечень пакетов расширений, достаточно выполнить следующую команду:

```
brew search php70-
```

Чтобы установить одно из этих расширений и связанные с ними библиотеки, достаточно выполнить команду **`brew install`**, указав в ней имя пакета расширения. Например, по команде

```
brew install php70-gmp
```

устанавливается расширение GMP (GNU Multiple Precision) для выполнения математических операций над большими числами с произвольной точностью. Подробнее об установке интерпретатора PHP средствами Homebrew можно узнать на веб-сайте Джастина Хайлмена (Justin Hileman) по адресу `https://bitly.com/hileiaan-php/`.

Установка интерпретатора PHP в Linux

В состав большинства распространяемых версий ОС Linux входит уже установленный интерпретатор PHP или двоичные пакеты PHP, которые можно установить. Так, если вы работаете с версией Fedora Linux (https://getfedora.org/), воспользуйтесь командой yum для установки пакета php. А если вы работаете с версией Ubuntu (http://www.ubuntu.com/), то воспользуйтесь командой **apt-get** для установки аналогичного пакета. Самым современным на момент написания данной книги был пакет php5 (версии PHP 5), тогда как официальный пакет php7 не был доступен. А вполне поддерживаемый пакет версии PHP 7 для Ubuntu доступен из другого источника. Прежде всего выполните следующие команды:

```
sudo add-apt-repository ppa:ondrej/php
```

и

```
sudo apt-get update
```

а затем установите пакет php7.0 по команде **apt-get**.

Если эти пакеты устарели, интерпретатор PHP можно заново построить самостоятельно. Загрузите архивный файл **.tar.gz** пакета Current Stable по адресу http://www.php.net/downloads.php. Введите по приглашению командного процессора следующие команды, чтобы распаковать этот архивный файл:

```
gunzip php-7.0.5.tar.gz
tar xvf php-7.0.5.tar
```

В итоге будет создан каталог **php-7.0.5**, содержащий исходный код интерпретатора PHP. Прочитайте инструкции по установке, приведенные в файле INSTALL, находящемся в каталоге верхнего уровня с исходным кодом. Краткий обзор процесса установки интерпретатора PHP в Linux и Unix можно найти на странице по адресу php.net, а инструкции по установке интерпретатора PHP на веб-сервере Apache 2.0 — по адресу http://ua2.php.net/manual/ru/install.unix.apache2.php.

Установка интерпретатора PHP в Windows

Установка интерпретатора PHP в Windows несколько отличается от его установки в Mac OS X или Linux. Допущения, которые интерпретатор PHP может сделать относительно того, что ему потребуется при установке, в данном случае отличаются, как, впрочем, инструментальные средства, которые могут быть доступны для его компиляции.

Правда, имеется ряд удобных и универсальных пакетов, объединяющих в себе PHP, Apache и MySQL для Windows. К их числу относятся пакеты WampServer (http://www.wampserver.com/en/), Bitnami WAMP Stack (https://bitnami.com/stack/wamp) и Apache FriendsXAMPP (https://www.apachefriends.org/index.html).

Корпорация Microsoft поддерживает веб-сайт, посвященный выполнению интерпретатора PHP на сервере IIS и доступный по адресу http://php.iis.net/. Кроме того, на официальном веб-сайте по адресу http://windows.php.net/ имеются разные версии PHP для Windows, доступные для загрузки.

Видоизменение директив конфигурации PHP

В предыдущих главах упоминались различные *директивы конфигурации* PHP. Эти параметры настройки оказывают влияние на поведение интерпретатора PHP, в том числе порядок сообщения об ошибках, места для поиска включаемых файлов и расширений и многое другое.

Прочитайте этот раздел, когда у вас возникнет потребность видоизменить директиву конфигурации или вас заинтересует, как настраивать параметры интерпретатора PHP, независимо от того, пользуетесь ли вы PHP на своем компьютере или же услугами веб-хостинга. Так, внесение коррективов в директиву `output_buffering`, упоминавшуюся в разделе "Причины для размещения вызовов функций `setcookie()` и `session_start()` вначале страницы" главы 10, намного упрощает манипулирование cookie-файлами и сеансами.

Значения директив конфигурации могут быть изменены в нескольких местах: в файле `php.ini` конфигурации интерпретатора PHP, в файле `httpd.conf` конфигурации веб-сервера Apache, в файлах конфигурации `.htaccess` или непосредственно в программах на PHP. Но не все директивы конфигурации можно изменять во всех этих местах. Если файл конфигурации `php.ini` или `httpd.conf` допускает редактирование, то именно в нем проще всего устанавливать директивы конфигурации. Но если эти файлы не допускают редактирование, то некоторые директивы конфигурации можно все-таки видоизменить непосредственно в программах на PHP.

Если ваш веб-сервер предписывает интерпретатору PHP пользоваться интерфейсом CGI или FastCGI, интерпретатор PHP ищет файл конфигурации `.user.ini` в том же самом каталоге, где выполняется программа на PHP. Если программа на PHP находится в корневом каталоге документов, то интерпретатор PHP ищет сначала родительский каталог этой программа, затем родительский каталог этого каталога и так далее до самого корневого каталога документов. Синтаксис для создания файлов конфигурации `.user.ini` такой же, как и для главного файла конфигурации `php.ini`.

Файл `php.ini` содержит директивы конфигурации интерпретатора PHP на системном уровне. Когда на веб-сервере запускается процесс, интерпретатор PHP читает файл конфигурации `php.ini` и соответственно настраивает конфигурацию. Чтобы найти местоположение файла конфигурации `php.ini` в своей системе, проверьте результат, выводимый функцией `phpinfo()`. Эта функция выводит отчет о конфигурации интерпретатора PHP. В примере А.2 демонстрируется крошечная программа, создающая страницу, внешний вид которой представлен на рис. А.1.

Пример А.2. Получение сведений о конфигурации с помощью функции phpinfo()

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

Как следует из шестой строки (`Configuration File (php.ini) Path`) на рис. А.l, файл конфигурации `php.ini` находится по пути `/php7/etc/php.ini`. Но у вас этот файл может находиться в другом месте. Строки, начинающиеся с точки с запятой (`;`) в файле конфигурации `php.ini`, содержат комментарии. А строки, в которых устанавливаются значения директив конфигурации, выглядят так, как показано в примере А.3.

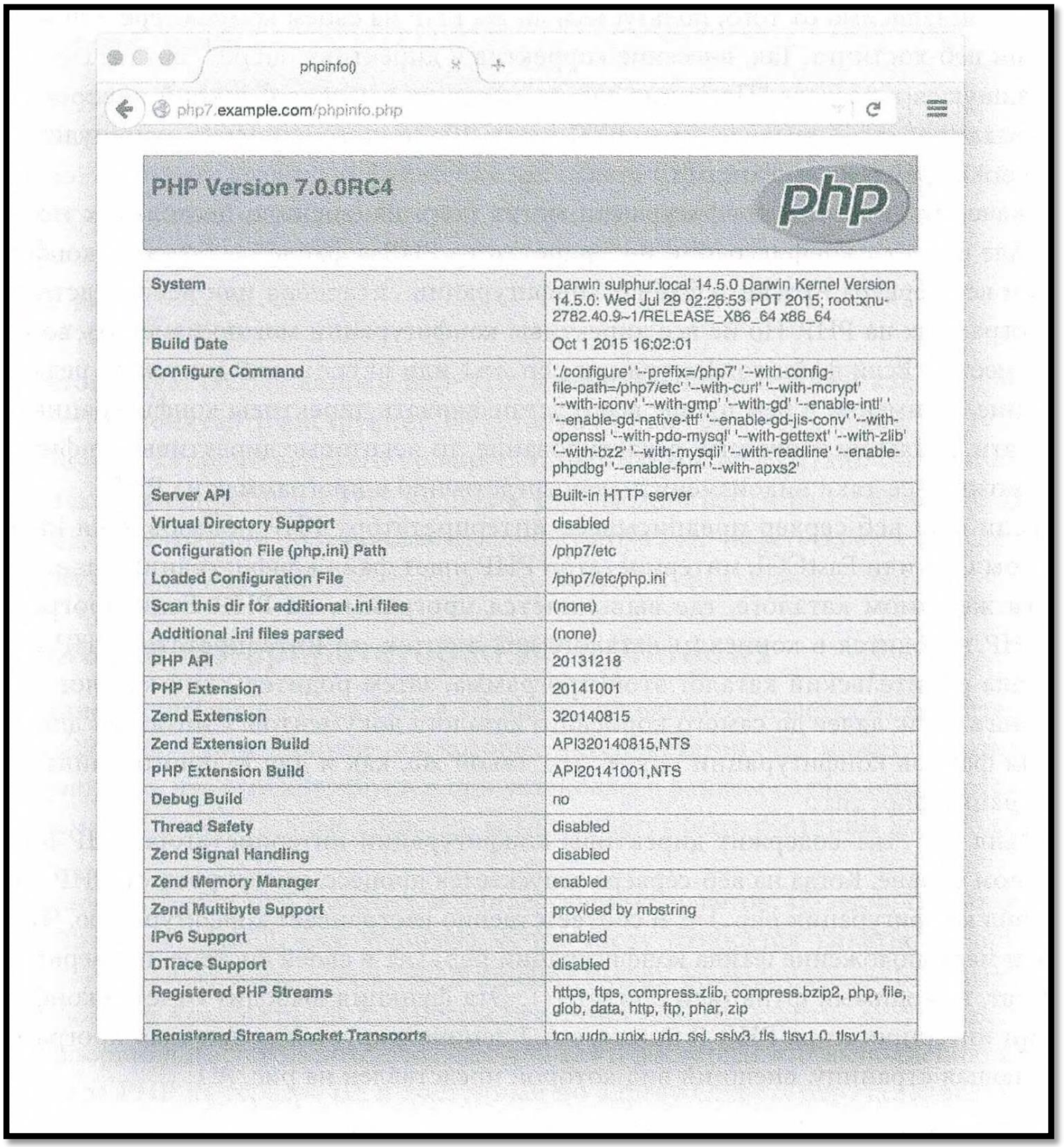
Пример А.3. Образцы строк из файла конфигурации php.ini

```
; Порядок указания каталогов в Unix следующий:
; косая черта в качестве разделителя и двоеточие
; между именами каталогов
include_path = ".:/usr/local/lib/php/includes"

; Порядок указания каталогов в Windows:
; обратная косая черта в качестве разделителя и
; точка с запятой между именами каталогов
; Windows: "\pathl;\path2"
include_path = ".;c:\php\includes"

; Сообщать обо всех ошибках, кроме замечаний
error_reporting = E_ALL & ~E_NOTICE

; Записывать ошибки в журнал регистрации
```



System	Darwin sulphur.local 14.5.0 Darwin Kernel Version 14.5.0: Wed Jul 29 02:26:53 PDT 2015; root:xnu-2782.40.9~1/RELEASE_X86_64 x86_64
Build Date	Oct 1 2015 16:02:01
Configure Command	'./configure' '--prefix=/php7' '--with-config-file-path=/php7/etc' '--with-curl' '--with-mcrypt' '--with-iconv' '--with-gmp' '--with-gd' '--enable-intl' '--enable-gd-native-ttf' '--enable-gd-jis-conv' '--with-openssl' '--with-pdo-mysql' '--with-gettext' '--with-zlib' '--with-bz2' '--with-mysqli' '--with-readline' '--enable-phpdbg' '--enable-fpm' '--with-apxs2'
Server API	Built-in HTTP server
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/php7/etc
Loaded Configuration File	/php7/etc/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131218
PHP Extension	20141001
Zend Extension	320140815
Zend Extension Build	API320140815,NTS
PHP Extension Build	API20141001,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1,

Рис. А.1: Результат, выводимый на экран функцией `phpinfo()`

```
log_errors = On

; Объем выгружаемого файла не может превышать 2 Мбайта
upload_max_filesize = 2M

; Срок действия сеансов истекает через 1440 секунд
session.gc_maxlifetime = 1440
```

Для установки директивы конфигурации `error_reporting` применяются встроенные константы в сочетании с логическими операциями. Например, в следующей строке:

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

устанавливается значение **E_ALL** директивы конфигурации `error_reporting`, а не значение **E_NOTICE**. Для установки значений в директивах конфигурации могут употребляться следующие логические операции: **&** (логическое И), **|** (логическое ИЛИ) и **~** (логическое отрицание, НЕ). Таким образом, для интерпретатора PHP выражение `E_ALL & ~E_NOTICE` означает установку значения **E_ALL**, но не значения **E_NOTICE**, а выражение `E_ALL | E_NOTICE` — установку значения `E_ALL` или же `E_NOTICE`.

Если в директиве конфигурации (например, `upload_max_filesize`) устанавливается числовое значение, к этому значению можно присоединить суффикс **M** (для обозначения мегабайт) или **K** (для обозначения килобайт) для умножения на **1048576** или **1024** соответственно. Так, обе установки, `upload_max_filesize=2M` и `upload_max_filesize=2097152`, одинаковы, поскольку 1 Мбайт составляет **1048576** байт, а **2097152 = 2 * 1048576**.

Для изменения директив в файле конфигурации `httpd.conf` или `.htaccess` вебсервера Apache служит несколько иной синтаксис, как показано в примере А.4.

Пример А.4. Примеры строк из файла `httpd.conf` конфигурации интерпретатора PHP

```
; Порядок указания каталогов в Unix следующий:
; косая черта в качестве разделителя и двоеточие
; между именами каталогов
php_value include_path ".:/usr/local/lib/php/includes"

; Порядок указания каталогов в Windows:
; обратная косая черта в качестве разделителя и
; точка с запятой между именами каталогов
; Windows: "\path1;\path2"
php_value include_path ".;c:\php\includes"

; Сообщать обо всех ошибках, кроме замечаний
php_value error_reporting "E_ALL & ~E_NOTICE"

; Записывать ошибки в журнал регистрации
php_flag log_errors On

; Объем выгружаемого файла не может превышать 2 Мбайта
php_value upload_max_filesize 2M

; Срок действия сеансов истекает через 1440 секунд
php_value session.gc_maxlifetime 1440
```

Ключевые слова `php_flag` и `php_value` в примере А.4 указывают веб-серверу Apache на то, что остальную часть строки составляет директива конфигурации интерпретатора PHP. После ключевого слова `php_flag` указывается имя директивы конфигурации и значение **`On`** или **`Off`**, а после ключевого слова `php_value` — имя директивы конфигурации и ее значение. Если устанавливаемое значение содержит пробелы (например, **`E_ALL & Ê_NOTICE`**), его следует заключить в кавычки. Между именем директивы конфигурации и ее значением знак равенства не ставится.

Для изменения значений директив конфигурации непосредственно в программе на PHP служит функция `ini_set()`. В примере А.5 демонстрируется установка значения директивы конфигурации `error_reporting` непосредственно в программе на PHP.

Пример А.5. Изменение значения в директиве конфигурации с помощью функции `ini_set()`

```
ini_set('error_reporting',E_ALL & ~E_NOTICE);
```

В качестве первого аргумента функции `ini_set()` указывается имя устанавливаемой директивы конфигурации, а в качестве второго аргумента — значение, устанавливаемое в этой директиве. Так, значением для установки директивы конфигурации `error_reporting` служит такое же логическое выражение, как и задаваемое в файле конфигурации `php.ini`. Для установки строковых и числовых значений в директивах конфигурации в качестве аргументов функции `ini_set()` передаются символьные строки и числа, а для установки логического значения **`On`** или **`Off`** — значение **`1`** или **`0`** соответственно.

Для извлечения значений директив конфигурации непосредственно в программе на PHP служит функция `ini_get()`. Достаточно передать этой функции имя директивы конфигурации, чтобы она возвратила ее значение. В примере А.6 демонстрируется добавление каталога в директиве конфигурации `include_path`, что нередко оказывается полезно сделать.

Пример А.6. Внесение изменений в директиву конфигурации `include_path` с помощью функций `i_get()` и `ini_set()`

```
// В следующих строках кода каталог /home/ireneo/php
// добавляется в конце директивы конфигурации include_path
$include_path = ini_get('include_path');
ini_set('include_path', $include_path . ':/home/ireneo/php');
```

Как упоминалось ранее, не все директивы конфигурации можно изменять повсеместно. Имеются такие директивы конфигурации, которые нельзя устанавливать непосредственно в программах на PHP. О таких директивах конфигурации интерпретатору PHP должно быть известно заранее, прежде чем он начнет чтение исходного текста программы. Например, директива конфигурации `output_buffering` вносит в поведение данного интерпретатора такие изменения, которые должны быть активизированы, прежде чем он начнет чтение исходного текста программы. Поэтому установить директиву конфигурации `output_buffering` с помощью функции `ini_set()` нельзя. Кроме того, одни директивы конфигурации запрещено устанавливать средствами PHP в файлах `.htaccess`, а другие — в файле `httpd.conf` конфигурации веб-сервера Apache. В то же время все директивы конфигурации из файла `php.ini` разрешается устанавливать в программах на PHP.

На странице руководства по PHP, оперативно доступной по адресу `http://www.php.net/ini.list`, перечислены все директивы конфигурации и контексты, в которых допускается их изменение. Некоторые из наиболее употребительных директив конфигурации перечислены в табл. А.1.

Таблица А.1. Наиболее употребительные директивы конфигурации

Директива	Рекомендуемое значение	Описание
`allow_url_fopen`	`On`	Разрешает или запрещает таким функциям, как `file_get_contents()`, оперировать веб-ресурсами по заданным URL, помимо локальных файлов
`auto_append_file`		Задает имя файла с кодом PHP, исполняемым после выполнения программы интерпретатором PHP. Это удобно для вывода общего для всех страниц нижнего колонтитула
`auto_prepend_file`		Задает имя файла с кодом PHP, исполняемым до выполнения программы интерпретатором PHP. Это удобно для определения функций или включения файлов, применяемых на веб-сайте в целом
`date.timezone`	`UTC`	Интерпретатору PHP требуется часовой пояс, устанавливаемый по умолчанию до вызова любых функций манипулирования датой или временем. Применение часового пояса UTC упрощает решение многих задач, связанных с датами и временем, как пояснялось в разделе "Манипулирование часовыми поясами" главы 15
`display_errors`	`On` – для отладки, `Off` – для эксплуатации	Если в этой директиве установлено значение On, интерпретатор PHP выводит на экран ошибки вместе с результатами выполнения программы
`error_reporting`	`E_ALL`	Определяет типы ошибок, о которых сообщает интерпретатор PHP. Подробнее об этом см. в разделе "Управление выводом сообщений об ошибках" главы 12
`extension`		В каждой строке с директивой `extension` в файле конфигурации `php.ini` загружается расширение PHP. Для этого в вашей систем должна присутствовать библиотека расширений
`extension_dir`		Задает каталог, в котором интерпретатор PHP ищет расширения, указанные в директиве `extension`
`file_uploads`	`On`	Разрешает выгрузку файлов через формы
`include_path`		Задает перечень каталогов, в которых интерпретатор PHP ищет файлы, загружаемые по директивам `include`, `require`, `include_once` и `require_once`
`log_errors`	`On`	Разрешает интерпретатору PHP направлять программные ошибки в журнал регистрации на веб-сервере

Окончание табл. А.1

Директива	Рекомендуемое значение	Описание
`output_buffering`	`On`	Разрешает интерпретатору PHP ожидать до тех пор, пока не будет выполнен сценарий, прежде чем посылать HTTP-заголовки, что упрощает пользование cookie-файлами и сеансами. Подробнее об этом см. в разделе "Причины для размещения вызовов функций **`setcookie()`** и **`session_start()`** в начале страницы" главы 10
`session.auto_start`	`On`, если применяются сеансы	Разрешает интерпретатору PHP начинать сеанс в начале каждой страницы, чтобы не вызывать функцию **`session_start()`**
`session.gc_maxlifetime`	`1440`	Количество секунд, определяющих продолжительность сеанса. Устанавливаемого по умолчанию значения **`1440`** должно быть достаточно для большинства приложений
`session.gc_probability`	`1`	Вероятность (**`1`** шанс из **`100`**), что истекшие сеансы будут очищены в начале любого запроса. Устанавливаемого по умолчанию значения **`1`** должно быть достаточно для большинства приложений
`short_open_tag`	`Off`	Если установлено значение **`Off`** данной директивы, блок кода PHP может начинаться с дескриптора **`<?`** или **`<? php`**. А поскольку не все серверы настроены на восприятие коротких дескрипторов, то рекомендуется всегда пользоваться дескриптором **`<? php`**
`track_errors`	`On` – для отладки, `Off` – для эксплуатации	Если установлено значение **`On`** данной директивы, интерпретатор PHP сохраняет сообщения об ошибках в глобальной переменной **`$php_errormsg`** при возникновении ошибок. Подробнее об этом см. в разделе "Выявление ошибок" главы 9
`upload_max_filesize`	`2M`	Задает максимально разрешаемый размер файла, выгружаемого через форму. Не увеличивайте это значение, если только не разрабатываете приложение, требующее от пользователей выгружать очень крупные файлы. Из-за большого количества крупных выгружаемых файлов может замедлиться работа веб-сервера

Резюме

В этом приложении были рассмотрены следующие вопросы.

- Поддержка PHP поставщиком услуг веб-хостинга.
- Установка интерпретатора PHP в ОС Linux, Mac OS X и Windows.
- Выяснение конфигурации интерпретатора PHP с помощью функции `phpinfo()`.
- Представление о структуре файла конфигурации `php.ini`.
- Настройка интерпретатора PHP в файле конфигурации `httpd.conf`.
- Чтение и запись значений директив конфигурации с помощью функций `ini_get()` и `ini_set()`.
- Применение наиболее употребительных директив конфигурации.

ПРИЛОЖЕНИЕ Б

Ответы на упражнения

Глава 2

Упражнение 1

1. Дескриптор `<?php`, открывающий код PHP, должен быть введен без пробела между знаками `<?` и ключевым словом `php`.

2. Символьная строка `I'm fine` содержит знак ', и поэтому она должна быть заключена в двойные кавычки (`"I'm fine"`), или же знак ' должен быть экранирован (`'I\'m fine'`).

3. Закрывающим код PHP должен быть дескриптор `?>`, а не `??>`. Если код PHP завершает исходный файл, то закрывающий его дескриптор может быть опущен.

Упражнение 2

```
$hamburger = 4.95;
$shake = 1.95;
$cola = 0.85;

$tip_rate = 0.16;
$tax_rate = 0.075;

$food = (2 * $hamburger) + $shake + $cola;
$tip = $food * $tip_rate;
$tax = $food * $tax_rate;

$total = $food + $tip + $tax;

print 'The total cost of the meal is $' . $total;
```

Упражнение 3

```
$hamburger = 4.95;
$shake = 1.95;
$cola = 0.85;
```

```
$tip_rate = 0.16;
$tax_rate = 0.075;

$food = (2 * $hamburger) + $shake + $cola;
$tip = $food * $tip_rate;
$tax = $food * $tax_rate;

$total = $food + $tip + $tax;

printf("%d %-9s at \$%.2f each: \$%5.2f\n", 2, 'Hamburger',
       $hamburger, 2 * $hamburger);
printf("%d %-9s at \$%.2f each: \$%5.2f\n", 1, 'Shake',
       $shake, $hamburger);
printf("%d %-9s at \$%.2f each: \$%5.2f\n", 1, 'Cola',
       $cola, $cola);
printf("%25s: \$%5.2f\n", 'Food Total', $food);
printf("%25s: \$%5.2f\n", 'Food and Tax Total', $food + $tax);
printf("%25s: \$%5.2f\n", 'Food, Tax, and Tip Total', $total);
```

Упражнение 4

```
$first_name = 'Srinivasa';
$last_name = 'Ramanujan';
$name = "$first_name $last_name";
print $name;
print strlen($name);
```

Упражнение 5

```
$n = 1; $p = 2; print "$n, $p\n";
$n++; $p *= 2; print "$n, $p\n";
$n++; $p *= 2; print "$n, $p\n";
$n++; $p *= 2; print "$n, $p\n";
$n++; $p *= 2; print "$n, $p\n";
```

Глава 3

Упражнение 1

```
a. Ложно (false)
b. Истинно (true)
c. Истинно (true)
d. Ложно (false)
e. Ложно (false)
f. Истинно (true)
g. Истинно (true)
h. Ложно (false)
```

Упражнение 2

Данная программа выводит следующий результат:

```
Message 3.Age: 12. Shoe Size: 14
```

Упражнение 3

```
$f = -50;
while ($f <= 50) {
    $c = ($f - 32) * (5/9);
    printf("%d degrees F = %d degrees C\n", $f, $c);
    $f += 5;
}
```

Упражнение 4

```
for ($f = -50; $f <= 50; $f += 5) {
    $c = ($f - 32) * (5/9);
    printf("%d degrees F = %d degrees C\n", $f, $c);
}
```

Глава 4

Упражнение 1

```
<table>
<tr><th>City</th><th>Population</th></tr>
<?php
$census = ['New York, NY' => 8175133,
           'Los Angeles, CA' => 3792621,
           'Chicago, IL' => 2695598,
           'Houston, TX' => 2100263,
           'Philadelphia, PA' => 1526006,
           'Phoenix, AZ' => 1445632,
           'San Antonio, TX' => 1327407,
           'San Diego, CA' => 1307402,
           'Dallas, TX' => 1197816,
           'San Jose, CA' => 945942];

$total = 0;
foreach ($census as $city => $population) {
    $total += $population;
    print "<tr><td>$city</td><td>$population</td></tr>\n";
}
print "<tr><td>Total</td><td>$total</td></tr>\n";
print "</table>";
```

Упражнение 2

```
$census = ['New York, NY' => 8175133,
           'Los Angeles, CA' => 3792621,
```

```
            'Chicago, IX' => 2695598,
            'Houston, TX' => 2100263,
            'Philadelphia, PA' => 1526006,
            'Phoenix, AZ' => 1445632,
            'San Antonio, TX' => 1327407,
            'San Diego, CA' => 1307402,
            'Dallas, TX' => 1197816,
            'San Jose, CA' => 945942];

// сортировать ассоциативный массив по значениям
asort($census);

print "<table>\n";
print "<tr><th>City</th><th>Population</th></tr>\n";
$total = 0;
foreach ($census as $city => $population) {
    $total += $population;
    print "<tr><td>$city</td><td>$population</td></tr>\n";
}
print "<tr><td>Total</td><td>$total</td></tr>\n";
print "</table>";

// сортировать ассоциативный массив по ключам
ksort($census);

print "<table>\n";
print "<tr><th>City</th><th>Population</th></tr>\n";
$total = 0;
foreach ($census as $city => $population) {
    $total += $population;
    print "<tr><td>$city</td><td>5population</td></tr>\n";
}
print "<tr><td>Total</td><td>$total</td></tr>\n";
print "</table>";
```

Упражнение 3

```
<table>
<tr><th>City</th><th>Population</th></tr>
<?php
// Каждый элемент массива $census является трехэлементным
// массивом, содержащим название города, сокращенное обозначение
// штата и численность населения города
$census = [ ['New York', 'NY', 8175133],
            ['Los Angeles', 'CA' , 3792621],
            ['Chicago', 'IL' , 2695598],
            ['Houston', 'TX' , 2100263],
            ['Philadelphia', 'PA' , 1526006],
            ['Phoenix', 'AZ' , 1445632],
            ['San Antonio', 'TX' , 1327407],
            ['San Diego', 'CA' , 1307402],
            ['Dallas', 'TX' , 1197816],
```

```
            ['San Jose', 'CA' , 945942] ];

$total = 0;
$state_totals = array();
foreach ($census as $city_info) {
    // обновить общую численность населения в каждом штате
    $total += $city_info[2];
    // Если данный штат еще не встречался, инициализировать
    // нулем переменную, хранящую общую численность его населения
    if (! array_key_exists($city_info[1], $state_totals)) {
        $state_totals[$city_info[1]] = 0;
    }
    // обновить общую численность населения в каждом штате
    $state_totals[$city_info[1]] += $city_info[2];
    print "<tr><td>$city_info[0], $city_info[1]</td><td>
               $city_info[2]</td></tr>\n";
}
print "<tr><td>Total</td><td>$total</td></tr>\n";
// вывести общую численность населения в каждом штате
foreach ($state_totals as $state => $population) {
    print "<tr><td>$state</td><td>$population</td></tr>\n";
}
print "</table>";
```

Упражнение 4

```
/* Классы и идентификационные номера учащихся в классе:
   ассоциативный массив, ключом которого служит Ф.И.О. учащегося,
   а значением — ассоциативный массив, содержащий класс, а также
   идентификационный номер учащегося
*/
$students = [ 'James D. McCawley' =>
                          [ 'grade' => 'A+','id' => 271231 ],
              'Buwei Yang Chao' =>
                          [ 'grade' => 'A', 'id' => 818211] ];
/* Количество каждого товара в запасах на складе:
   ассоциативный массив, ключом которого служит наименование
   товара, а значением - порядковый номер хранения на складе
*/
$inventory = [ 'Wok' => 5, 'Steamer' => 3, 'Heavy Cleaver' => 3,
               'Light Cleaver' => 0 ];
/* Школьные обеды, состоящие из разных блюд (закусок, салатов,
   напитков и т.д.), а также их стоимость на каждый день недели:
   ассоциативный массив, ключом которого служит день недели, а
   значением - ассоциативный массив, описывающий блюдо. Этот
   ассоциативный массив содержит пару "ключ-значение" на стоимость
   каждого обеда, а также пары "ключ-значение" на каждое блюдо
   составляющее обед на данный день недели
*/
$lunches = [ 'Monday' => [ 'cost' => 1.50,
                           'entree' => 'Beef Shu-Mai',
                           'side' => 'Salty Fried Cake',
```

```
                        'drink' => 'Black Tea' ],
            'Tuesday' => [ 'cost' => 2.50,
                           'entree' => 'Clear-steamed Fish',
                           'side' => 'Turnip Cake',
                           'drink' => 'Bubble Tea' ],
            'Wednesday' => [ 'cost' => 2.00,
                             'entree' => 'Braised Sea Cucumber',
                             'side' => 'Turnip Cake',
                             'drink' => 'Green Tea' ],
            'Thursday' => [ 'cost' => 1.35,
                            'entree' => 'Stir-fried Two Winters',
                            'side' => 'Egg Puff',
                            'drink' => 'Black Tea' ],
             'Friday' => [ 'cost' => 3.25,
                           'entree' => 'Stewed Pork with Taro',
                           'side' => 'Duck Feet',
                           'drink' => 'Jasmine Tea' ] ];

/* Имена членов вашей семьи:
   числовой массив, индексы которого не указаны явно, а
   значениями служат имена членов вашей семьи
*/
$family = [ 'Bart', 'Lisa', 'Homer', 'Marge', 'Maggie' ];
/* Имена, возраст и родство членов семьи:
   ассоциативный массив, ключами которого служат имена членов
   семьи, а значениями — ассоциативные массивы, состоящие из
   пар "ключ-значение", обозначающих возраст и степень родства
*/
$family = [ 'Bart' => [ 'age' => 10,
                        'relation' => 'brother' ],
            'Lisa' => [ 'age' => 7,
                        'relation' => 'sister' ],
            'Homer' => [ 'age' => 36,
                         'relation' => 'father' ],
            'Marge' => [ 'age' => 34,
                         'relation' => 'mother' ],
            'Maggie' => [ 'age' => 1,
                          'relation' => 'self' ] ];
```

Глава 5

Упражнение 1

```
function html_img($url, $alt = null, $height = null,
                  $width = null) {
    $html = '<img src="' . $url . '"';
    if (isset ($alt)) {
        $html .= ' alt="' . $alt . '"';
    }
    if (isset($height)) {
        $html .= ' height="' . $height . '"';
    }
```

```
    if (isset($width)) {
        $html .= ' width="' . $width . '"';
    }
    $html .= '/>';
    return $html;
}
```

Упражнение 2

```
function html_img2($file, $alt = null, $height = null,
                   $width = null) {
    if (isset($GLOBALS['image_path'])) {
        $file = $GLOBALS['image_path'] . $file;
    }
    $html = '<img src="' . $file . '"';
    if (isset($alt)) {
        $html .= ' alt="' . $alt . '"';
    }
    if (isset($height)) {
        $html .= ' height="' . $height . '"';
    }
    if (isset($width)) {
        $html .= ' width="' . $width . '"';
    }
    $html .= '/>';
    return $html;
}
```

Упражнение 3

```
// В следующем файле хранится функция html_img2()
// из предыдущего упражнения
include "html-img2.php";

$image_path = '/images/';

print html_img2('puppy.png');
print html_img2('kitten.png', 'fuzzy');
print html_img2('dragon.png',null,640,480);
```

Упражнение 4

На экран выводится следующее:

```
I can afford a tip of 11% (30)
I can afford a tip of 12% (30.25)
I can afford a tip of 13% (30.5)
I can afford a tip of 14% (30.75)
```

Упражнение 5

```
/* Применение функции dechex(): */
function web_color1($red, $green, $blue) {
    $hex = [ dechex($red), dechex($green), dechex($blue) ];
    // предварить начальным нулем, если требуется, значения
    // цвета, состоящие из одной цифры
    foreach ($hex as $i => $val) {
        if (strlen($i) == 1) {
            $hex[$i] = "0$val";
        }
    }
    return '#' . implode('', $hex);
}

/* Выполняя преобразование шестнадцатеричных значений в
   десятичные, можно опираться и на символ форматирования
   %x, употребляемый при вызове функции sprintf(),
   как показано ниже */
function web_color2($red, $green, $blue) {
    return sprintf('#%02x%02x%02x', $red, $green, $blue);
}
```

Глава 6

Упражнение 1

```
class Ingredient {
    protected $name;
    protected $cost;

    public function __construct($name, $cost) {
        $this->name = $name;
        $this->cost = $cost;
    }

    public function getName() {
        return $this->name;
    }

    public function getCost() {
        return $this->cost;
    }
}
```

Упражнение 2

```
class Ingredient {
    protected $name;
    protected $cost;

    public function __construct($name, $cost) {
```

```
        $this->name = $name;
        $this->cost = $cost;
    }

    public function getName() {
        return $this->name;
    }

    public function getCost() {
        return $this->cost;
    }

    // В следующем методе задается новая величина
    // стоимости ингредиента блюда
    public function setCost($cost) {
        $this->cost = $cost;
    }
}
```

Упражнение 3

```
class PricedEntree extends Entree {
    public function ___construct($name, $ingredients) {
        parent::__construct($name, $ingredients);
        foreach ($this->ingredients as $ingredient) {
            if (! $ingredient instanceof Ingredient) {
                throw new Exception('Elements of $ingredients
                                     must be Ingredient objects');
            }
        }
    }

    public function getCost() {
        $cost = 0;
        foreach ($this->ingredients as $ingredient) {
            $cost += $ingredient->getCost();
        }
        return $cost;
    }
}
```

Упражнение 4

Класс `Ingredient` размещается в собственном пространстве имен следующим образом:

```
namespace Meals;

class Ingredient {
    protected $name;
    protected $cost;

    public function ___construct($name, $cost) {
```

```
        $this->name = $name;
        $this->cost = $cost;
    }

    public function getName() {
        return $this->name;
    }

    public function getCost() {
        return $this->cost;
    }

    // В следующем методе задается новая величина
    // стоимости ингредиента блюда
    public function setCost($cost) {
        $this->cost = $cost;
    }
}
```

А обращение к данному пространству имен из класса `PricedEntree` осуществляется таким образом:

```
class PricedEntree extends Entree {
    public function ___construct($name, $ingredients) {
        parent::__construct($name, $ingredients);
        foreach ($this->ingredients as $ingredient) {
            if (! $ingredient instanceof \Meals\Ingredient) {
                throw new Exception('Elements of $ingredients
                                     must be Ingredient objects');
            }
        }
    }

    public function getCost() {
        $cost = 0;
        foreach ($this->ingredients as $ingredient) {
            $cost += $ingredient->getCost();
        }
        return $cost;
    }
}
```

Глава 7

Упражнение 1

Массив `$_POST` будет содержать следующее:

```
$_POST['noodle'] = 'barbecued pork';
$_POST['sweet'] = [ 'puff', 'ricemeat' ];
$_POST['sweet_q'] = '4';
$_POST['submit'] = 'Order';
```

Упражнение 2

```
/* Следующая функция оперирует данными из переданной на обработку
   формы, и поэтому она обращается непосредственно к массиву
   $_POST, а не к массиву $input проверенных на достоверность
   данных */
function process_form() {
    print '<ul>';
    foreach ($_POST as $k => $v) {
        print '<li>' . htmlentities($k) . '='
                     . htmlentities($v) . '</1i>';
    }
    print '</ul>';
}
```

Упражнение 3

```
<?php

// В этой программе предполагается, что исходный файл
// FormHelper.php находится в том же самом файле, где и
// исходный файл данной программы
require 'FormHelper.php';

// Установить массив с вариантами выбора из списка.
// Следующий массив требуется в функциях display_form(),
// validate_form() и process_form(), поэтому он объявляется
// в глобальной области действия
$ops = array('+','-','*','/');

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их.
        process_form($input);
        // А затем отобразить форму снова, чтобы выполнить
        // очередное вычисление
        show_form();
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
```

```
    $defaults = array('num1' => 2,
                      'op' => 2, // индекс операции '*'
                                 // в массиве $ops
                      'num2' => 8);
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper($defaults);

    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'math-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();

    // требуется ввести операцию
    $input['op'] = $GLOBALS['ops'][$_POST['op']] ?? '';
    if(! in_array($input['op'], $GLOBALS['ops'])) {
        $errors[] = 'Please select a valid operation.';
    }
    // ключи num1 и num2 должны быть числовыми

    $input['num1'] = filter_input(INPUT_POST, 'num1',
                                  FILTER_VALIDATE_FLOAT);
    if (is_null($input['num1']) || ($input['num1'] === false)) {
        $errors[] = 'Please enter a valid first number.';
    }
    $input['num2'] = filter_input(INPDT_POST, 'num2',
                                  FILTER_VALIDATE_FLOAT);
    if (is_null($input['num2']) || ($input['num2'] === false)) {
        $errors[] = 'Please enter a valid second number.';
    }

    // Деление на нуль недопустимо!
    if (($input['op'] == '/') && ($input['num2'] == 0)) {
        $errors[] = 'Division by zero is not allowed.';
    }
    return array($errors, $input);
}

function process_form($input) {
    $result = 0;
    if ($input['op'] == '+') {
        $result = $input['num1'] + $input['num2'];
    }
    elseif ($input['op'] == '-') {
        $result = $input['num1'] - $input['num2'];
    }
    elseif ($input['op'] == '*') {
        $result = $input['num1'] * $input['num2'];
    }
```

```
    elseif($input['op'] =='/') {
        $result = $input['num1'] / $input['num2'];
    }
    $message = "{$input['num1']} {$input['op']}
                {$input['num2']} = $result";
    print "<h3>$message</h3>";
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `math-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>

  <tr><td>First Number :</td>
    <td><?= $form->input('text', ['name' => 'num1']) ?></td>
  </tr>
  <tr><td>Operation: </td>
    <td><?= $form->select($GLOBALS['ops'], ['name' => 'op']) ?></td>
  </tr>
  <tr><td>Second Number:</td>
    <td><?= $form->input('text', ['name' => 'num2']) ?></td>
  </tr>

  <tr><td colspan="2" align="center">
      <?= $form->input('submit', ['value' => 'Calculate']) ?>
  </td></tr>
</table>
</form>
```

Упражнение 4

```
<?php

// Здесь предполагается, что исходный файл FormHelper.php
// находится в том же каталоге, где и данный файл
require 'FormHelper.php';

// Установить массив с вариантами выбора из списка,
// размечаемого дескриптором <select>. Следующий массив
// требуется в функциях display_form(), validate_form()
```

```
// и process_form(), и поэтому он объявляется в глобальной
// области действия
$states = [ 'AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DC',
            'DE', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN', 'IA',
            'KS', 'KY', 'LA', 'ME', 'MD', 'MA', 'MI', 'MN',
            'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM',
            'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI',
            'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA',
            'WV', 'WI', 'WY' ];

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}
function show_form($errors = array()) {
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'shipping-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();
    foreach (['from','to'] as $addr) {
        // проверить обязательные поля
        foreach(['Name' => 'name', 'Address 1' => 'address1',
                 'City' => 'city', 'State' => 'state']
                 as $label => $field) {
            $input[$addr.'_'.$field] =
                            $_POST[$addr.'_'.$field] ?? '';
            if (strlen($input[$addr.'_'.$field]) == 0) {
                $errors[] = "Please enter a value for $addr $label.";
            }
        }
        // проверить штат
        $input[$addr.'_state'] =
```

```
        $GLOBALS['states'][$input[$addr.'_state']] ?? '';
        if (! in_array($input[$addr.'_state'], $GLOBALS['states'])) {
            $errors[] = "Please select a valid $addr state.";
        }
        // проверить почтовый индекс
        $input[$addr.'_zip'] = filter_input(INPUT_POST, $addr.'_zip',
                        FILTER_VALIDATE_INT, ['options' =>
                                             ['min_range'=>10000,
                                              'max_range'=>99999]]);
        if (is_null($input[$addr.'_zip']) ||
                    ($input[$addr.'_zip']===false)) {
            $errors[] = "Please enter a valid $addr ZIP";
        }
        // Не забыть о втором адресе address2!
        $input[$addr.'_address2'] = $_POST[$addr.'_address2'] ?? '';
    }

    // высота, ширина, глубина, вес посылки должны быть
    // выражены числовыми положительными значениями
    foreach(['height','width','depth','weight'] as $field) {
        $input[$field] =filter_input(INPUT_POST, $field,
                                     FILTER_VALIDATE_FLOAT);
        // Нулевое значение является недействительным, поэтому
        // проверить следующее условие только на истинность вместо
        // того, чтобы проверять на пустое или ложное значение
        if (! ($input[$field] && ($input[$field] > 0))) {
            $errors[] = "Please enter a valid $field.";
        }
    }

    // проверить вес посылки
    if ($input['weight'] > 150) {
        $errors[] = "The package must weigh no more than 150 lbs.";
    }
    // проверить размеры посылки
    foreach(['height'width','depth'] as $dim) {
        if ($input[$dim] > 36) {
            $errors[] = "The package $dim must be no more
                         than 36 inches.";
        }
    }

    return array($errors, $input);
}

function process_form($input) {
    // создать шаблон для отчета
    $tpl=<<<HTML
    <p>Your package is {height}" x {width}" x {depth}"
            and weighs {weight} lbs.</p>
    <p>It is coming from:</p>
    <pre>
```

```
    {from_name}
    {from_address}
    {from_city}, {from_state} {from_zip}
    </pre>

    <p>It is going to:</p>
    <pre>
    {to_name}
    {to_address}
    {to_city}, {to_state} {to_zip}
    </pre>
    HTML;

    // откорректировать адреса в массиве $input,
    // чтобы упростить вывод отчета
    foreach(['from', 'to'] as $addr) {
        $input[$addr.'_address'] = $input[$addr.'_address1'];
        if (strlen($input[$addr.'_address2'])) {
            $input[$addr.'_address'] .= "\n"
                    . $input[$addr.'_address2'];
        }
    }

    // Заменить каждую переменную из шаблона соответствующим
    // значением из массива $input
    $html = $tpl;
    foreach($input as $k => $v) {
        $html = str_replace('{'.$k.'}', $v, $html);
    }

    // вывести отчет
    print $html;
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `shipping-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>

  <tr><th>From:</th><td></td></tr>
```

```
<tr><td>Name:</td>
  <td><?= $form->input('text', ['name' => 'from_name']) ?>
  </td></tr>
<tr><td>Address 1:</td>
  <td><?= $form->input('text', ['name' => 'from_address1']) ?>
  </td></tr>
<tr><td>Address 2:</td>
  <td><?= $form->input('text', ['name' => 'from_address2']) ?>
  </td></tr>
<tr><td>City: </td>
  <td><?= $form->input('text', ['name' => 'from_city']) ?>
  </td></tr>
<tr><td>State:</td>
  <td><?= $form->select($GLOBALS['states'],
      ['name' => 'from_state']) ?>
  </td></tr>
<tr><td>ZIP:</td>
  <td><?= $form->input('text', ['name' => 'from_zip',
      'size' => 5]) ?>
  </td></tr>

<tr><th>To: </th><td></td></tr>
<tr><td>Name: </td>
  <td><?= $form->input('text', ['name' => 'to_name']) ?>
  </td></tr>
<tr><td>Address 1:</td>
  <td><?= $form->input('text', ['name' => 'to_address1']) ?>
  </td></tr>
<tr><td>Address 2:</td>
  <td><?= $form->input('text', ['name' => 'to_address2']) ?>
  </td></tr>
<tr><td>City:</td>
  <td><?= $form->input('text', ['name' => 'to_city']) ?>
  </td></tr>
<tr><td>State: </td>
  <td><?= $form->select($GLOBALS['states'],
      ['name' => 'to_state']) ?>
  </td></tr>
<tr><td>ZIP:</td>
  <td><?= $form->input('text', ['name' => 'to_zip',
      'size' => 5]) ?>
  </td></tr>

<tr><th>Package:</th><td></td></tr>
<tr><td>Weight:</td>
  <td><?= $form->input('text', ['name' => 'weight']) ?></td></tr>
<tr><td>Height:</td>
  <td><?= $form->input('text', ['name' => 'height']) ?></td></tr>
<tr><td>Width:</td>
  <td><?= $form->input('text', ['name' => 'width']) ?></td></tr>
<tr><td>Depth:</td>
  <td><?= $form->input('text', ['name' => 'depth']) ?></td></tr>
```

```
  <tr><td colspan="2" align="center">
    <?= $form->input('submit', ['value' => 'Ship!']) ?>
  </td></tr>

</table>
</form>
```

Упражнение 5

```
function print_array($ar) {
    print '<ul>';
    foreach ($ar as $k => $v) {
        if (is_array($v)) {
            print '<li>' . htmlentities($k) .':</li>';
            print_array($v);
        } else {
            print '<li>' . htmlentities($k) . '='
                         . htmlentities($v) . '</li>';
        }
    }
    print '</ul>';
}

/* Эта функция оперирует данными из переданной на обработку
   формы, и поэтому она обращается непосредственно к массиву
   $_POST, а не к массиву $input проверенных на достоверность
   данных */
function process_form() {
    print_array($_POST);
}
```

Глава 8

Упражнение 1

```
try {
    // подключиться к базе данных
    $db = new PDO('sqlite:/tmp/restaurant.db');
    // установить исключения при возникновении ошибок в БД
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $db->query('SELECT * FROM dishes ORDER BY price');
    $dishes = $stmt->fetchAll();
    if (count($dishes) == 0) {
        $html = '<p>No dishes to display</p>';
    } else {
        $html = "<table>\n";
        $html .= "<tr><th>Dish Name</th><th>Price</th><th>Spicy?
                  </th></tr>\n";
        foreach ($dishes as $dish) {
            $html .= '<tr><td>' .
            htmlentities($dish['dish_name']) . '</td><td>$' .
```

```
            sprintf('%.02f', $dish['price']) . '</td><td>'
            ($dish['is_spicy'] ? 'Yes' : 'No') . "</td></tr>\n";
        }
        $html .= "</table>";
    }
} catch (PDOException $e) {
    $html = "Can't show dishes: " . $e->getMessage();
}
print $html;
```

Упражнение 2

```
<?php

// загрузить вспомогательный класс для составления формы
require 'FormHelper.php';

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Can't connect: " . $e->getMessage();
    exit();
}
// установить исключения при возникновении ошибок в базе данных
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// установить режим извлечения строк из таблицы в виде объектов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
```

```
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'price-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();
    // Минимальная цена должна быть представлена достоверным
    // числом с плавающей точкой
    $input['min_price'] = filter_input(INPOT_POST,'min_price',
                                       FILTER_VALIDATE_FLOAT);
    if ($input['min_price'] === null ||
                                     $input['min_price'] === false) {
        $errors[] = 'Please enter a valid minimum price.';
    }
    return array($errors, $input);
}

function process_form($input) {
    // получить в этой функции доступ к глобальной переменной $db
    global $db;
    // составить запрос к базе данных
    $sql = 'SELECT dish_name, price, is_spicy FROM dishes
            WHERE price >= ?';
    // отправить запрос программе базы данных и получить в ответ
    // все строки из таблицы базы данных
    $stmt = $db->prepare($sql);
    $stmt->execute(array($input['min_price']));
    $dishes = $stmt->fetchAll();
    if (count($dishes) == 0) {
        print 'No dishes matched.';
    } else {
        print '<table>';
        print ' <tr><th>Dish Name</th><th>Price</th><th>Spicy?
                                    </th></tr>';
        foreach ($dishes as $dish) {
            if ($dish->is_spicy == 1) {
                $spicy = 'Yes';
            } else {
                $spicy = 'No';
            }
            printf('<tr><td>%s</td><td>$%.02f</td><td>%s
                                </td></tr>',
                   htmlentities($dish->dish_name),
                   $dish->price, $spicy);
        }
        print '</table>';
    }
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся

в главе 7. В нем делается также ссылка на исходный файл `price-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>
  <tr>
    <td>Minimum Price:</td>
    <td><?= $form->input('text',['name' => 'min_price']) ?></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
        <?= $form->input('submit', ['name' => 'search',
                                    'value' => 'Search']) ?>
    </td>
  </tr >
</table>
</form>
```

Упражнение 3

```
<?php

// загрузить вспомогательный класс для составления форм
require 'FormHelper.php';

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Can't connect: " . $e->getMessage();
    exit();
}
// установить исключения при возникновении ошибок в базе данных
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// установить режим извлечения строк из таблицы в виде объектов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] = 'POST') {
```

```
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    global $db;
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
    // извлечь список наименований блюд из базы данных
    // для последующего применения
    $sql = 'SELECT dish_id, dish_name FROM dishes
                                        ORDER BY dish_name';
    $stmt = $db->query($sql);
    $dishes = array();
    while ($row = $stmt->fetch()) {
        $dishes[$row->dish_id] = $row->dish_name;
    }
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'dish-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();
    // Если передан идентификатор блюда (dish_id value),
    // считать его достоверным. Если же он не совпадает ни
    // с одним из блюд в базе данных, об этом может сообщить
    // функция process_form()
    if (isset($_POST['dish_id'])) {
        $input['dish_id'] = $_POST['dish_id'];
    } else {
        $errors[] = 'Please select a dish.';
    }
    return array($errors, $input);
}

function process_form($input) {
    // получить в этой функции доступ к глобальной переменной $db
    global $db;
    // составить запрос к базе данных
    $sql = 'SELECT dish_id, dish_name, price, is_spicy FROM dishes
```

```
                    WHERE dish_id = ?';
    // отправить запрос программе базы данных и получить в ответ
    // все строки из таблицы базы данных
    $stmt = $db->prepare($sql);
    $stmt->execute(array($input['dish_id']));
    $dish = $stmt->fetch();
    if (count($dish) == 0) {
      print 'No dishes matched.';
    } else {
      print '<table>';
      print ' <tr><th>ID</th><th>Dish Name</th><th>Price</th>';
      print '<th>Spicy?</th></tr>';
      if ($dish->is_spicy == 1) {
          $spicy = 'Yes';
      } else {
          $spicy = 'No';
      }
      printf('<tr><td>%d</td><td>%s</td><td>$%.02f</td>
                  <td>%s</td></tr>',
             $dish->dish_id, htmlentities($dish->dish_name),
             $dish->price, $spicy);
      print '</table>';
    }
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `dish-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
       <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
    <?php } ?>

    <tr>
      <td>Dish:</td>
      <td><?= $form->select($dishes, ['name' => 'dish_id']) ?></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
          <?= $form->input('submit', ['name' => 'info',
          'value' => 'Get Dish Info']) ?></td>
    </tr>
</table>
```

```
</form>
```

Упражнение 4

```
<?php

// загрузить вспомогательный класс для составления формы
require 'FormHelper.php';

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (PDOException $e) {
    print "Can't connect: " . $e->getMessage();
    exit();
}

// установить исключения при возникновении ошибок в базе данных
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// установить режим извлечения строк из таблицы в виде объектов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);

// ввести список идентификаторов и наименований блюд в
// глобальный массив, поскольку он потребуется в функциях
// show_form() и validate_form()
$dishes = array();
$sql = 'SELECT dish_id, dish_name FROM dishes ORDER BY dish_name';
$stmt = $db->query($sql);
while ($row = $stmt->fetch()) {
    $dishes[$row->dish_id] = $row->dish_name;
}

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] = 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}
```

```
function show_form($errors = array()) {
    global $db, $dishes;
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'customer-form.php';
}

function validate_form() {
    global $dishes;
    $input = array();
    $errors = array();
    // Убедиться в достоверности переданного идентификатора
    // блюда (dish_id) и его наличии в массиве $dishes.
    // Если передан идентификатор блюда (dish_id value),
    // считать его достоверным. Если же он не совпадает ни
    // с одним из блюд в базе данных, об этом может сообщить
    // функция process_form()
    $input['dish_id'] = $_POST['dish_id'] ?? '';
    if (! array_key_exists($input['dish_id'], $dishes)) {
        $errors[] = 'Please select a valid dish.';
    }

    // требуется ввести имя
    $input['name'] = trim($_POST['name'] ?? '');
    if (0 == strlen($input['name'])) {
        $errors[] = 'Please enter a name.';
    }

    // требуется ввести номер телефона
    $input['phone'] = trim($_POST['phone'] ?? '');
    if (0 == strlen($input['phone'])) {
        $errors[] = 'Please enter a phone number.';
    } else {
        // В США номер телефона состоит, как минимум, из
        // 10 цифр. И хотя проверка достоверности телефонного
        // номера с помощью функции ctype_digit() оказывается
        // не самой эффективной, тем не менее, она логически
        // проста и не требует употребления регулярных выражений
        $digits = 0;
        for ($i = 0; $i < strlen($input['phone']); $i++) {
            if (ctype_digit($input['phone'][$i])) {
                $digits++;
            }
        }
        if ($digits < 10) {
           $errors[] = 'Phone number needs at least ten digits.';
        }
    }
    return array($errors, $input);
}
```

```
function process_form($input) {
    // получить в этой функции доступ к глобальной переменной $db
    global $db;
    // Составить запрос к базе данных. Указывать идентификатор
    // абонента (customer_id) необязательно, поскольку он
    // автоматически присваивается в базе данных
    $sql = 'INSERT INTO customers (name,phone,favorite_dish_id) '
           . 'VALUES (?,?,?)';
    // отправить запрос программе базы данных и получить в ответ
    // все строки из таблицы базы данных
    try {
        $stmt = $db->prepare($sql);
        $stmt->execute(array($input['name'],$input['phone'],
                       $input['dish_id']));
        print '<p>Inserted new customer.</p>';
    } catch (Exception $e) {
        print "<p>Couldn't insert customer:
                  {$e->getMessage()},</p>";
    }
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `customer-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>
  <tr>
  <tr><td>Name:</td><td><?= $form->input('text',
                ['name' => 'name']) ?>
  </td></tr>
  <tr><td>Phone Number:</td>
  <td><?= $form->input('text', ['name' => 'phone']) ?>
  </td></tr>
  <tr><td>Favorite Dish:</td>
  <td><?= $form->select($dishes,['name' => 'dish_id']) ?></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
        <?= $form->input('submit', ['name' => 'add',
                                    'value' => 'Add Customer']) ?>
```

```
    </td>
  </tr>
</table>
</form>
```

Глава 9

Упражнение 1

HTML-файл шаблона template.html выглядит следующим образом:

```
<html>
  <head><title>{title}</title></head>
  <body>
    <h1>{headline}</h1>
    <h2>By {byline}</h2>
    <div class="article">{article}</div>
    <p><small>Page generated: {date}</small></p>
  </body>
</html>
```

А программа на PHP для замены переменных из шаблона выглядит таким образом:

```
$now = new DateTime();
// выразить переменные как можно проще следующим образом:
// ключ => значение
$vars = array('title' => 'Man Bites Dog',
              'headline' => 'Man and Dog Trapped in Biting Fiasco',
                            'byline' => 'Ireneo Funes',
                            'article' => <<<_HTML_
<p>While walking in the park today, Bioy Casares took a big juicy
bite out of his dog, Santa's Little Helper. When asked why he did
it, Mr. Casares said, "I was hungry."</p>
_HTML_
                            '
                            'date' => $now->format('1, F j, Y'));

// создать вариант массива $vars, чтобы соответствовать синтаксису
// шаблонной обработки, заключив ключи в фигурные скобки
$template_vars = array();
foreach ($vars as $k => $v) {
    $template_vars['{'.$k.'}'] = $v;
}
// загрузить шаблон
$template = file_get_contents('template.html');
if ($template === false) {
    die("Can't read template.html: $php_errormsg");
}
// Если задан один массив символьных строк для поиска и другой
// массив для замены, то все замены автоматически выполняются
// функцией str_replace()
$html = str_replace(array_keys($template_vars),
```

```
                    array_values($template_vars),
                    $template);
// вывести новую HTML-страницу
$result = file_put_contents('article.html', $html);
if ($result === false) {
    die("Can't write article.html: $php_errormsg");
}
```

Упражнение 2

```
// Массив для накапливания подсчитываемых адресов
$addresses = array();
$fh = fopen('addresses.txt','rb');
if (! $fh) {
    die("Can't open addresses.txt: $php_errormsg");
}
while ((! feof($fh)) && ($line = fgets($fh))) {
    $line = trim($line);
    // воспользоваться адресом в качестве ключа в
    // массиве $addresses. Значение обозначает, сколько
    // раз появляется адрес
    if (! isset($addresses[$line])) {
        $addresses[$line] = 0;
    }
    $addresses[$line] = $addresses[$line] + 1;
}
if (! fclose($fh)) {
    die("Can't close addresses.txt: $php_errormsg");
}

// отсортировать элементы массива $addresses в обратном порядке,
// когда сначала следуют самые большие значения элементов массива

arsort($addresses);
$fh = fopen('addresses-count.txt','wb');
if (! $fh) {
    die("Can't open addresses-count.txt: $php_errormsg");
}
foreach ($addresses as $address => $count) {
    // не забыть о знаке перевода строки в ее конце
    if (fwrite($fh, "$count,$address\n") === false) {
        die("Can't write $count,$address: $php_errormsg");
    }
}
if (! fclose($fh)) {
    die("Can't close addresses-count.txt: $php_errormsg");
}
```

Ниже приведен текстовый файл `addresses.txt` с образцами адресов электронной почты.

```
brilling@tweedledee.example.com
slithy@unicorn.example.com
uffish@knight.example.net
```

```
slithy@unicorn.example.com
jubjub@sheep.example.com
tumtum@queen.example.org
slithy@unicorn.example.com
uffish@knight.example.net
manxome@king.example.net
beamish@lion.example.org
uffish@knight.example.net
frumious@tweedledum.example.com
tulgey@carpenter.example.com
vorpal@crow.example.org
beamish@lion.example.org
mimsy@walrus.example.com
frumious@tweedledum.example.com
raths@owl.example.net
frumious@tweedledum.example.com
```

Упражнение 3

```
$fh = fopen('dishes.csv','rb');
if (! $fh) {
    die("Can't open dishes.csv: $php_errormsg");
}
print "<table>\n";
while ((! feof($fh)) && ($line = fgetcsv($fh))) {
    // воспользоваться функцией implode(), как и главе 4
    print "<tr><td>" . implode("</td><td>", $line)
                     . "</td></tr>\n";
}
print "</table>";
```

Упражнение 4

```
<?php

// загрузить вспомогательный класс для составления формы
require 'FormHelper.php';

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
```

```
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'filename-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();
    // убедиться, что имя файла указано
    $input['file'] = trim($_POST['file'] ?? '');
    if (0 == strlen($input['file'])) {
        $errors[] = 'Please enter a filename.';
    } else {
        // убедиться, что полное имя файла находится по пути
        // к корневому каталогу документов на веб-сервере
        $full = $_SERVER['DOCUMENT_ROOT'] . '/' . $input['file'];
        // воспользоваться функцией realpath() для разрешения
        // любых последовательностей знаков .. или символических
        // ссылок
        $full = realpath($full);
        if ($full === false) {
            $errors[] = "Please enter a valid filename.";
        } else {
            // убедиться, что полный путь к файлу в переменной
            // $full начинается с корневого каталога документов
            $docroot_len = strlen($_SERVER['DOCUMENT_ROOT']);
            if (substr($full, 0, $docroot_len) !=
                                $_SERVER['DOCUMENT_ROOT']) {
                $errors[] = 'File must be under document root.';
            } else {
                // Если это так, то сохранить полный путь к файлу
                // в переменной $full, чтобы воспользоваться ею в
                // функции process_form()
                $input['full'] = $full;
            }
        }
   }

    return array($errors, $input);
}

function process_form($input) {
    if (is_readable($input['full'])) {
        print htmlentities(file_get_contents($input['full']));
```

```
    } else {
        print "Can't read {$input['file']}.";
    }
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `filename-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>

  <tr><td>File:</td>
    <td><?= $form->input('text', ['name' => 'file']) ?></td></tr>
  <tr><td colspan="2"
      align="center"><?= $form->input('submit',
      ['value' => 'Display']) ?>
  </td></tr>

</table>
</form>
```

Упражнение 5

Ниже приведен новый вариант функции `validate_form()`, где реализована дополнительная проверка с помощью функции `strcasecmp()`.

```
function validate_form() {
    $input = array();
    $errors = array();

    // убедиться, что имя файла указано
    $input['file'] = trim($_POST['file'] ?? '');
    if (0 == strlen($input['file'])) {
        $errors[] = 'Please enter a filename.';
    } else {
        // убедиться, что полное имя файла находится по пути
        // к корневому каталогу документов на веб-сервере
        $full = $_SERVER['DOCUMENT_ROOT'] . '/' . $input['file']
        // воспользоваться функцией realpath() для разрешения
        // любых последовательностей знаков .. или символических
        // ссылок
```

```
        $full = realpath($full);
        if ($full = false) {
            $errors[] = "Please enter a valid filename.";
        } else {
            // убедиться, что полный путь к файлу в переменной
            // $full начинается с корневого каталога документов
            $docroot_len = strlen($_SERVER['DOCUMENT_ROOT']);
            if (substr($full, 0, $docroot_len) !=
                        $_SERVER['DOCUMENT_ROOT']) {
                $errors[] = 'File must be under document root.';
            } elseif (strcasecmp(substr($full, -5),
                        '.html') != 0) {
                  $errors[] = 'File name must end in .html';
            } else {
                // Если это так, то сохранить полный путь к файлу
                // в переменной $full, чтобы воспользоваться ею
                // в функции process_form()
                $input['full'] = $full;
            }
        }
    }

    return array($errors, $input);
}
```

Глава 10

Упражнение 1

```
$view_count = 1 + ($_COOKIE['view_count'] ?? 0);
setcookie('view_count', $view_count);
print "<p>Hi! Number of times you've viewed this page:
          $view_count.</p>";
```

Упражнение 2

```
$view_count = 1 + ($_COOKIE['view_count'] ?? 0);

if ($view_count == 20) {
    // Если функции setcookie() в качестве второго аргумента
    // передано пустое строковое значение, то cookie-файл удаляется
    setcookie('view_count', '');
    $msg = "<p>Time to start over.</p>";
} else {
    setcookie('view_count', $view_count);
    $msg = "<p>Hi! Number of times you've viewed this page:
                $view_count.</p>" ;
    if ($view_count == 5) {
        $msg .= "<p>This is your fifth visit.</p>";
    } elseif ($view_count == 10) {
        $msg .= "<p>This is your tenth visit.
                    You must like this page.</p>";
```

```
    } elseif ($view_count == 15) {
        $msg .= "<p>This is your fifteenth visit. " .
                "Don't you have anything else to do?</p>";
    }
}
print $msg;
```

Упражнение 3

Страница с формой для выбора цвета выглядит следующим образом:

```
<?php

// начать сеансы, прежде всего, с того, чтобы свободно
// воспользоваться в дальнейшем глобальным массивом $_SESSION
session_start();

// загрузить вспомогательный класс для составления формы
require 'FormHelper.php';

$colors = array('ff0000' => 'Red',
                'ffa500' => 'Orange',
                'ffffff' => 'Yellow',
                '008000' => 'Green',
                '0000ff' => 'Blue',
                '4b0082' => 'Indigo',
                '663399' => 'Rebecca Purple');

// Основная логика функционирования страницы:
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form($errors = array()) {
    global $colors;
    // создать объект $form с надлежащими свойствами по умолчанию
    $form = new FormHelper();
    // Ради ясности весь код HTML-разметки и отображения
```

```
    // формы вынесен в отдельный файл
    include 'color-form.php';
}

function validate_form() {
    $input = array();
    $errors = array();

    // цвет должен быть достоверным
    $input['color'] = $_POST['color'] ?? '';
    if (! array_key_exists($input['color'],
                           $GLOBALS['colors'])) {
        $errors[] = 'Please select a valid color.';
    }
    return array($errors, $input);
}

function process_form($input) {
    global $colors;
    $_SESSION['background_color'] = $input['color'];
    print '<p>Your color has been set.</p>';
}
?>
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `color-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>
  <tr>
    <td>Favorite Color:</td>
    <td><?= $form->select($colors,['name' => 'color']) ?></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
        <?= $form->input('submit', ['name' => 'set',
                                    'value' => 'Set Color']) ?>
    </td>
  </tr>
</table>
</form>
```

Ниже приведена страница, на которой установлен цвет фона.

```
<?php
// начать сеансы, прежде всего, с того, чтобы свободно
// воспользоваться в дальнейшем глобальным массивом $_SESSION
session_start();
?>
<html>
  <head><title>Background Color Example</title>
  <body style="background-color:<?=
               $_SESSION['background_color'] ?>">
    <p>What color did you pick?</p>
  </body>
</html>
```

Упражнение 4

Страница с формой заказа выглядит следующим образом:

```
session_start();

// Здесь предполагается, что исходный файл FormHelper.php
// находится в том же самом каталоге, где данный файл
require 'FormHelper.php';

// Установить массив с вариантами выбора из списка,
// размечаемого дескриптором <select>. Этот массив
// требуется в функциях display_form(), validate_form()
// и process_form(), и поэтому он объявляется в глобальной
// области действия
$products = [ 'cuke' => 'Braised Sea Cucumber',
              'stomach' => "Sauteed Pig's Stomach",
              'tripe' => 'Sauteed Tripe with Wine Sauce',
              'taro' => 'Stewed Pork with Taro',
              'giblets' => 'Baked Giblets with Salt',
              'abalone' => 'Abalone with Marrow and Duck Feet'];

// Основная логика функционирования страницы;
// - Если форма передана на обработку, проверить достоверность
// данных, обработать их и снова отобразить форму.
// - Если форма не передана на обработку, отобразить ее снова
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Если функция validate_form() возвратит ошибки,
    // передать их функции show_form()
    list($errors, $input) = validate_form();
    if ($errors) {
        show_form($errors);
    } else {
        // Переданные данные из формы достоверны, обработать их
        process_form($input);
    }
} else {
   // Данные из формы не переданы, отобразить ее снова
   show_form();
```

```
}

function show_form($errors = array()) {
    global $products;
    $defaults = array();
    // начать заполнение массива с нулевого значения,
    // устанавливаемого по умолчанию
    foreach($products as $code => $label) {
        $defaults["quantity_$code"] = 0;
    }
    // Если количества товаров сохраняются в сеансе,
    // использовать именно их
    if (isset($_SESSION['quantities'])) {
        foreach ($_SESSION['quantities'] as
                 $field => $quantity) {
            $defaults[$field] = $quantity;
        }
    }
    $form = new FormHelper($defaults);
    // Ради ясности весь код HTML-разметки и отображения
    // формы вынесен в отдельный файл
    include 'order-form.php';
}

function validate_form() {
    global $products;

    $input = array();
    $errors = array();
    // убедиться, что в каждом поле ввода количества товаров
    // указаны достоверные целочисленные положительные значения
    foreach ($products as $code => $name) {
        $field = "quantity_$code";
        $input[$field] = filter_input(INPUT_POST, $field,
                                      FILTER_VALIDATE_INT,
                          ['options' => ['min_range'=>0]]);
        if (is_null($input[$field]) ||
                   ($input[$field] === false)) {
            $errors[] = "Please enter a valid quantity for $name.";
        }
    }

    return array($errors, $input);
}

function process_form($input) {
    $_SESSION['quantities'] = $input;
    print "Thank you for your order.";
}
```

Исходный код данной программы основан на исходном файле `FormHelper.php`, упоминавшемся в главе 7. В нем делается также ссылка на исходный файл `order-form.php`, предназначенный для отображения формы, размеченной в формате HTML. Ниже приведено содержимое этого файла.

```
<form method="POST" action=
                    "<?= $form->encode($_SERVER['PHP_SELF']) ?>">
<table>
  <?php if ($errors) { ?>
    <tr>
      <td>You need to correct the following errors:</td>
      <td><ul>
        <?php foreach ($errors as $error) { ?>
          <li><?= $form->encode($error) ?></li>
        <?php } ?>
      </ul></td>
  <?php } ?>

  <tr><th>Product</thxtd>Quantity</td></tr>
<?php foreach ($products as $code => $name) { ?>
  <tr><td><?= htmlentities ($name) ?>:</td>
    <td><?= $form->input('text', ['name' => "quantity_$code"]) ?>
    </td></tr>
<?php } ?>
  <tr><td colspan="2"
      align="center"><?= $form->input('submit',
      ['value' => 'Order']) ?>
  </td></tr>
</table>
</form>
```

Страница оформления заказа выглядит следующим образом:

```
session_start();

// Те же самые товары, что и на странице с формой заказа
$products = ['cuke' => 'Braised Sea Cucumber',
             'stomach' => "Sauteed Pig's Stomach",
             'tripe' => 'Sauteed Tripe with Wine Sauce',
             'taro' => 'Stewed Pork with Taro',
             'giblets' => 'Baked Giblets with Salt',
             'abalone' => 'Abalone with Marrow and Duck Feet'];

// упрощенная основная логика функционирования страницы
// без проверки достоверности данных, введенных в форме
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    process_form();
} else {
    // Данные из формы не переданы, отобразить ее снова
    show_form();
}

function show_form() {
    global $products;

    // Форма состоит из единственной кнопки передачи формы
    // на обработку, и поэтому для ее отображения вместо
    // вспомогательного класса FormHelper здесь просто
```

```
    // вставляется HTML-разметка
    if (isset($_SESSION['quantities']) &&
          (count($_SESSION['quantities'])>0)) {
        print "<p>Your order: </p><ul>";
        foreach ($_SESSION['quantities'] as $field => $amount) {
            list($junk, $code) = explode('_', $field);
            $product = $products[$code];
            print "<li>$amount $product</li>";
        }
        print "</ul>";
        print '<form method="POST" action=' .
                htmlentities($_SERVER['PHP_SELF']) . '>';
        print '<input type="submit" value="Check Out" />';
        print '</form>';
    } else {
        print "<p>You don't have a saved order.</p>";
    }
    // Здесь предполагается, что страница с формой заказа
    // сохранена в исходном файле order.php
    print '<a href="order.php">Return to Order page</a>';
}

function process_form() {
    // Здесь данные удаляются из сеанса
    unset($_SESSION['quantities']);
    print "<p>Thanks for your order.</p>";
}
```

Глава 11

Упражнение 1

```
$json = file_get_contents("http://php.net/releases/?json");
if ($json === false) {
    print "Can't retrieve feed.";
}
else {
    $feed = json_decode($json, true);
    // Переменная $feed содержит массив, ключи которого упорядочены
    // по номерам основных версий. Поэтому сначала необходимо
    // извлечь самый большой номер версии
    $major_numbers = array_keys($feed);
    rsort($major_numbers);
    $biggest_major_number = $major_numbers[0];
    // Элемент "version", находящийся в массиве по ключу
    //с номером основной версии, обозначает самый последний
    // выпуск с номером этой основной версии
    $version = $feed[$biggest_major_number]['version'];
    print "The latest version of PHP released is $version.";
}
```

Упражнение 2

```
$c = curl_init("http://php.net/releases/?json");
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$json = curl_exec($c);
if ($json === false) {
    print "Can't retrieve feed.";
}
else {
    $feed = json_decode($json, true);
    // Переменная $feed содержит массив, ключи которого упорядочены
    // по номерам основных версий. Поэтому сначала необходимо
    // извлечь самый большой номер версии
    $major_numbers = array_keys($feed);
    rsort($major_numbers);
    $biggest_major_number = $major_numbers[0];
    // Элемент "version", находящийся в массиве по ключу
    // с номером основной версии, обозначает самый последний
    // выпуск с номером этой основной версии
    $version = $feed[$biggest_major_number]['version'];
    print "The latest version of PHP released is $version.";
}
```

Упражнение 3

```
// Количество секунд, истекших с 1 января 1970 г.
// и до текущего момента времени
$now = timed;
setcookie('last_access', $now);
if (isset($_COOKIE['last_access'])) {
    // чтобы создать объект типа DateTime из значения,
    // обозначающего количество секунд, истекших с 1 января
    // 1970 г., предварить это значение префиксом @
    $d = new DateTime('@' . $_COOKIE['last_access']);
    $msg = '<p>You last visited this page at ' .
                $d->format('g:i a') . ' on ' .
                $d->format('F j, Y') . '</p>';
} else {
    $msg = '<p>This is your first visit to this page.</p>';
}
print $msg;
```

Упражнение 4

```
$url = 'https://api.github.com/gists';
$data = ['public' => true,
         'description' => "This program a gist of itself.",
         // Как следует из документации на прикладной
         // программный интерфейс API, ключи в объекте,
         // представляющем файлы, обозначают имя файла,
         // а значения — содержимое файла
         'files' => [ basename (__FILE__) =>
```

```
                [ 'content' => file_get_contents(__FILE__) ] ] ];
$c = curl_init($url);
curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
curl_setopt($c, CURLOPT_POST, true);
curl_setopt($c, CURLOPT_HTTPHEADER, array(
                        'Content-Type: application/json'));
curl_setopt($c, CURLOPT_POSTFIELDS, json_encode($data));
curl_setopt($c, CURLOPT_USERAGENT, 'learning-php-7/exercise');

$response = curl_exec($c);
if ($response === false) {
    print "Couldn't make request.";
} else {
    $info = curl_getinfo($c);
    if ($info['http_code'] != 201) {
        print "Couldn't create gist, got {$info['http_code']}\n";
        print $response;
    } else {
        $body = json_decode($response);
        print "Created gist at {$body->html_url}\n";
    }
}
```

Глава 12

Упражнение 1

Ключевое слово `global` должно отсутствовать в строке кода 5. В связи с этим должно быть выдано сообщение о синтаксической ошибке, где указано на неожиданно встретившееся ключевое слово:

```
PHP Parse error: syntax error, unexpected 'global' (T_GLOBAL)
in debugging-12.php on line 5
```

Чтобы данная программа выполнялась нормально, следующую строку кода:

```
print global $name;
```

нужно заменить такой строкой:

```
print $GLOBALS['name'];
```

С другой стороны, в первой строке кода функции можно ввести следующее:

```
global name;
```

а затем заменить строку кода

```
print global $name;
```

такой строкой:

```
print $name;
```

Упражнение 2

```
function validate_form() {
    $input = array();
    $errors = array();

    // включить буферизацию вывода
    ob_start();
    // вывести все переданные на обработку данные
    var_dump($_POST);
    // зафиксировать сформированные для вывода данные
    $output = ob_get_contents();
    // выключить буферизацию вывода
    ob_end_clean();
    // направить данные, выводимые из переменной,
    // в журнал регистрации
    error_log($output);

    // требуется ввести операцию
    $input['op'] = $GLOBALS['ops'] [$_POST['op']] ?? '';
    if (! in_array($input['op'], $GLOBALS['ops'])) {
        $errors[] = 'Please select a valid operation.';
    }
    // ключи num1 и num2 должны быть числовыми
    $input['num1'] = filter_input(INPOT_POST, 'num1',
                                  FILTER_VALIDATE_FLOAT);
    if (is_null($input['num1']) || ($input['num1'] === false)) {
        $errors[] = 'Please enter a valid first number.';
    }

    $input['num2'] = filter_input(INPUT_POST, 'num2',
                                  FILTER_VALIDATE_FLOAT);
    if (is_null($input['num2']) || ($input['num2'] === false)) {
        $errors[] = 'Please enter a valid second number.';
    }

    // Деление на нуль недопустимо!
    if {($input['op'] == '/') && {$input['num2'] == 0)) {
        $errors[] = 'Division by zero is not allowed.';
    }

    return array($errors, $input);
}
```

Упражнение 3

Приведенный ниже код располагается в самом начале программы. В нем определяется обработчик исключений и организуется его вызов при возникновении неперехватываемых исключений.

```
function exceptionHandler($ex) {
    // вывести подробности возникшей ошибки в журнал
    // регистрации ошибок
    error_log("ERROR: " . $ex->getMessage());
```

```
    // вывести причину возникшей ошибки в удобной для
    // пользователя форме и выйти из программы
    die("<p>Sorry, something went wrong.</p>");
}
set_exception_handler('exceptionHandler');
```

А далее блоки операторов try/catch можно удалить в тех местах программы, где они употребляются: один раз — при создании объекта типа PDO, а второй раз — в функции process_form(). Ведь теперь исключения будут обрабатываться специально предназначенным для этой цели обработчиком.

Упражнение 4

- Строка кода 4. Заменить **::** на **:** в имени источника данных (DSN).
- Строка кода 5. Заменить `catch ($e)` на `catch (Exception $e)`.
- Строка кода 16. Заменить `$row['dish_id']]` на `$row['dish_id']` в качестве ключа для поиска в массиве `$dish_names`.
- Строка кода 18. Заменить ****** на ***** в запросе SQL.
- Строка кода 20. Заменить **=** на **==**.
- Строка кода 26. Заменить третий спецификатор формата с **%f** на **%s**, поскольку `$customer['phone']` — это символьная строка.
- Строка кода 30. Заменить `$customer['favorite\_dish\_id']` на `$dish_names[$customer['favorite\_dish\_id']]`, чтобы преобразовать идентификатор блюда в наименование соответствующего блюда.
- Строка кода 33. Вставить закрывающую фигурную скобку (**}**), чтобы согласовать ее с открывающей фигурной скобкой (**{**) в строке кода 22.

Ниже приведен весь исходный код исправленной программы.

```
<?php

// подключиться к базе данных
try {
    $db = new PDO('sqlite:/tmp/restaurant.db');
} catch (Exception $e) {
    die("Can't connect: " . $e->getMessage());
}
// организовать обработку исключений
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// установить режим извлечения строк из таблицы в виде массивов
$db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
// получить массив наименований блюд из базы данных
$dish_names = array();
$res = $db->guery('SELECT dish_id,dish_name FROM dishes');
foreach ($res->fetchAll()as $row) {
    $dish_names[ $row['dish_id'] ] = $row['dish_name'];
}
$res = $db->query('SELECT * FROM customers ORDER BY phone DESC');
```

```
$customers = $res->fetchAll();
if (count($customers) == 0) {
    print "No customers.";
} else {
    print '<table>';
    print '<tr><th>ID</th><th>Name</th>
           <th>Phone</th><th>Favorite Dish</th></tr>';
    foreach($customers as $customer) {
        printf("<tr><td>%d</td><td>%s</td><td>%s</td><td>%s</td>
                </tr>\n",
               $customer['customer_id'],
               htmlentities($customer['customer_name']),
               $customer['phone'],
               $dish_names[$customer['favorite_dish_id']]);
    }
    print '</table>';
}
?>
```

Глава 13

Упражнение 2

```
public function testNameMustBeSubmitted() {
    $submitted = array('age' => '15',
                       'price' => '39.95');
    list($errors, $input) = validate_form($submitted);
    $this->assertContains('Your name is required.', $errors);
    $this->assertCount(1, $errors);
}
```

Упражнение 3

```
include 'FormHelper.php';

class FormHelperTest extends PHPUnit_Framework_TestCase {
    public $products = [ 'cu&ke' => 'Braised <Sea> Cucumber',
                         'stomach' => "Sauteed Pig's Stomach",
                         'tripe' =>
                             'Sauteed Tripe with Wine Sauce',
                         'taro' => 'Stewed Pork with Taro',
                         'giblets' => 'Baked Giblets with Salt',
                         'abalone' =>
                             'Abalone with Marrow and Duck Feet'];

    public $stooges = ['Larry','Moe','Curly','Shemp'];
    // Этот код выполняется перед каждым тестом. Разместив его в
    // специальном методе setUp(), можно сократить исходный код,
    // чтобы не повторять его в каждом тестовом методе
    public function setup() {
        $_SERVER['REQUEST_METHOD' ] = 'GET';
    }
```

```
    public function testAssociativeOptions() {
        $form = new FormHelper();
        $html = $form->select($this->products);
        $this->assertEquals($html,<<<_HTML_
<select ><option value="cu&amp;ke">Braised &lt;Sea&gt; Cucumber
         </option>
<option value="stomach">Sauteed Pig's Stomach</option>
<option value="tripe">Sauteed Tripe with Wine Sauce</option>
<option value="taro">Stewed Pork with Taro</option>
<option value="giblets">Baked Giblets with Salt</option>
<option value="abalone">Abalone with Marrow and Duck Feet</option></select>
_HTML_
        );
    }

    public function testNumericOptions() {
        $form = new FormHelper();
        $html = $form->select($this->stooges);
        $this->assertEquals ($html, <<<_HTML_
<select ><option value="0">Larry</option>
<option value="1">Moe</option>
<option value="2">Curly</option>
<option value="3">Shemp</option></select>
_HTML_
        );
    }

   public function testNoOptions() {
        $form = new FormHelper();
        $html = $form->select([]);
        $this->assertEquals('<select ></select>', $html);
    }

    public function testBooleanTrueAttributes() {
        $form = new FormHelper();
        $html = $form->select([],['np' => true]);
        $this->assertEquals('<select np></select>', $html);
    }

    public function testBooleanFalseAttributes() {
        $form = new FormHelper();
        $html = $form->select([],['np' => false,
                                  'onion' => 'red']);
        $this->assertEquals('<select onion="red">
                             </select>', $html);
    }

    public function testNonBooleanAttributes() {
        $form = new FormHelper();
        $html = $form->select([],['spaceship'=>'<=>']);
        $this->assertEquals('<select spaceships"&lt;=&gt;">
```

```
                            </select>', $html);
    }

    public function testMultipleAttribute() {
        $form = new FormHelper();
        $html = $form->select([],["name" => "menu",
                "q" => 1, "multiple" => true]);
        $this->assertEquals('<select name="menu[]" q="1" multiple>
                            </select>', $html);
    }
}
```

Упражнение 4

Ниже приведены дополнительные тестовые методы для класса `FormHelperTest`.

```
public function testButtonNoTypeOK() {
    $form = new FormHelper();
    $html = $form->tag('button');
    $this->assertEquals('<button />',$html);
}

public function testButtonTypeSubmitOK() {
    $form = new FormHelper();
    $html = $form->tag('button',['type'=>'submit']);
    $this->assertEquals('<button type="submit" />',$html);
}

public function testButtonTypeResetOK() {
    $form = new FormHelper();
    $html = $form->tag('button',['type'=>'reset']);
    $this->assertEquals('<button type="reset" />',$html);
}

public function testButtonTypeButtonOK() {
    $form = new FormHelper();
    $html = $form->tag('button',['type'=>'button']);
    $this->assertEquals('<button type="button" />',$html);
}

public function testButtonTypeOtherFails() {
    $form = new FormHelper();
    // В классе FormHelper должно генерироваться
    // исключение типа InvalidArgumentException,
    // когда предоставляется недостоверный атрибут
    $this->setExpectedException('InvalidArgumentException');
    $html = $form->tag('button', ['type'=>'other']);
}
```

Ниже приведены коррективы, которые требуется внести в класс `FormHelper`, чтобы прошли все тесты.

```
// Этот код следует лишь после объявления class FormHelper.
// Следующий массив выражает с помощью заданных в нем элементов
// те имена атрибутов, которые имеют допустимые значения
protected $allowedAttributes = ['button' =>
                                ['type' => ['submit',
                                            'reset',
                                            'button' ] ] ];

// Метод tag() изменен таким образом, чтобы переменную $tag
// можно было передать в качестве первого аргумента при вызове
// метода $this->attributes()
public function tag($tag, $attributes = array(),
                    $isMultiple = false) {
    return "<$tag {$this->attributes($tag, $attributes,
                                     $isMultiple)} />";
}

// Метод start() также изменен, чтобы переменную $tag
// можно было передать в качестве первого аргумента при вызове
// метода $this->attributes()
public function start($tag, $attributes = array(),
                      $isMultiple = false) {
    // Дескрипторы <select> и <textarea> не получают
    // атрибуты value
    $valueAttribute = (! (($tag == 'select') ||
                          ($tag == 'textarea')));
    $attrs = $this->attributes($tag, $attributes, $isMultiple,
                               $valueAttribute);
    return "<$tag $attrs>";
}

// Метод attributes() изменен, чтобы принимать переменную $tag
// в качестве первого аргумента. Установить переменную
// $attributeCheck, если при вызове метода $this-attributes()
// были определены атрибуты, допустимые для дескриптора, а затем
// проверить, допустимо ли предоставленное значение, а иначе —
// сгенерировать исключение
protected function attributes($tag, $attributes, $isMultiple,
                              $valueAttribute = true) {
    $tmp = array();
    // Если данный дескриптор может содержать атрибут value,
    // а его имени соответствует элемент в массиве значений,
    // то установить этот атрибут
    if ($valueAttribute && isset($attributes['name']) &&
           array_key_exists($attributes['name'], $this->values)) {
        $attributes['value'] = $this->values[$attributes['name']];
    }
    if (isset($this->allowedAttributes[$tag])) {
        $attributeCheck = $this->allowedAttributes[$tag];
    } else {
        $attributeCheck = array();
    }
```

```
    foreach ($attributes as $k => $v) {
        // проверить, допустимо ли значение атрибута
        if (isset($attributeCheck[$k]) &&
           (! in_array($v, $attributeCheck[$k]))) {
           throw new InvalidArgumentException(
                 "$v is not allowed as value for $k");
        }
        // Истинное логическое значение означает
        // логический атрибут
        if (is_bool($v)) {
            if ($v) { $tmp[] = $this->encode ($k); }
        }
        // иначе k = v
        else {
            $value = $this->encode($v);
            // Если это многозначный элемент, присоединить
            // квадратные скобки ([]) к его имени
            if ($isMultiple && ($k == 'name')) {
                $value .= '[]';
            }
            $tmp[] = "$k=\"$value\"";
        }
    }
    return implode('', $tmp);
}
```

Предметный указатель

О

П

У

Ф

Ц

Э

Я