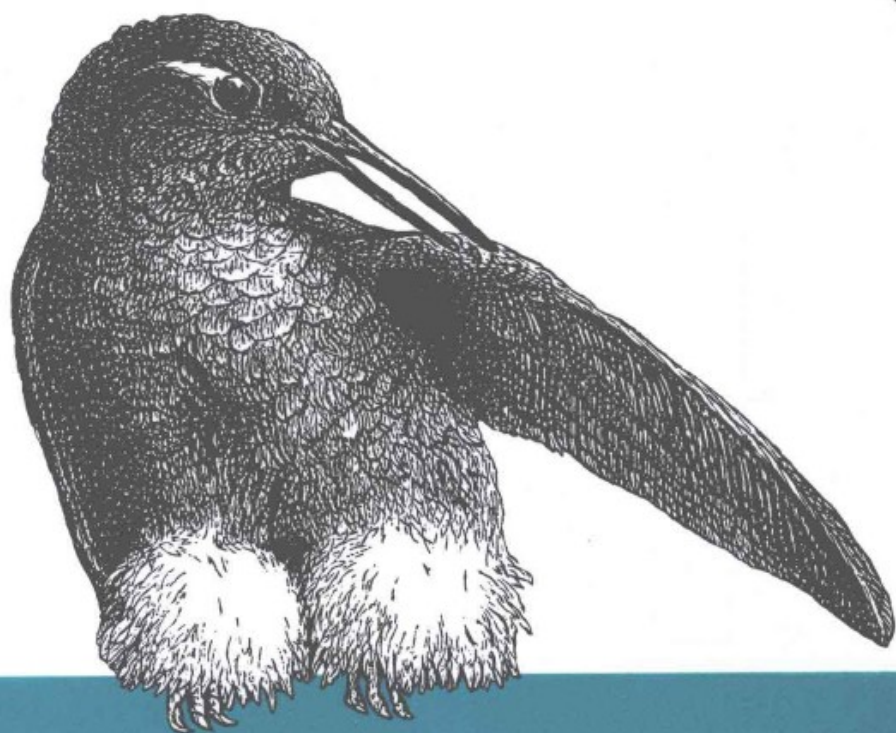


PHP: The Good Parts  
揭示PHP的精华

涵盖  
PHP 5.3 的精华



# PHP

## 语言精粹

Peter B. MacIntyre 著  
Susie Sedlacek 序  
刘涛 丁静 译

O'REILLY®



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>



# PHP语言精粹

让我们忽略所有关于PHP的炒作，只深挖这门语言的真正力量。这本书探索PHP所有最有用的特征，向你展示它们究竟如何为Web开发提速，并且向你解释为什么最常用的PHP要素会被误用。你将会学到究竟是什么为面向对象编程添加力量和如何使用某些特征来让你的应用与数据库集成。

本书由PHP社区的一个资深成员完成，非常适合新的PHP程序员和由其他语言转向PHP的程序员阅读。

- 熟悉PHP的基本语法、变量和数据类型
- 学会如何让这门语言与网页相集成
- 理解如何使用字符串、数组和PHP内建函数
- 发现把PHP作为面向对象语言使用的益处
- 探索PHP如何与数据库交互，如SQLite和MySQL
- 学会与消除安全隐患相关的输入输出处理

Peter B. MacIntyre有超过20年的软件开发经验，在此期间，他使用客户端/服务器工具以及相关的数据库系统如PHP、PowerBuilder、Visual Basic和ASP。他也是phplarchitect的前任特约编辑和作者。

无须具备PHP经验。

图书分类：编程语言/Web开发

策划编辑：张春雨

责任编辑：徐津平



**Broadview**  
WWW.BROADVIEW.COM.CN

www.phei.com.cn

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

O'Reilly Media, Inc. 授权电子工业出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行

This Authorized Edition for sale in the mainland of China (excluding Hong Kong, Macao and Taiwan)



“涵盖了PHP最新特征使本书具有实时性并非常有价值。它甚至向你展示如何自动发送SMS的文本消息！由于PHP的调整正变得越来越宽泛，这本书将帮助你赶上这种变革的步伐！”

——Andi Gutmans

CEO, Zend Technologies

“当我学习PHP的时候，这本书在哪里呢？它对于任何刚开始学习PHP的人都是非常好的资源。彼得为大家提供了一本开始使用PHP的最佳工具书。”

——Cal Evans

Chief Marketing Officer,  
Blue Parabola

**O'REILLY**<sup>®</sup>  
oreilly.com.cn

ISBN 978-7-121-15385-3



9 787121 153853 >

定价：39.00元

O'REILLY®

# PHP语言精粹

---

PHP: The Good Parts

【美】Peter B. MacIntyre 著  
Susie Sedlacek 序  
刘涛 丁静 译

电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING

## 内 容 简 介

这是一本可以带你迈入 PHP 殿堂的书。

PHP 作为当今主流的服务器端开发语言，广泛应用于世界上各种排名比较靠前的网站，如 Facebook、Flickr 和 Wikipedia 等。其广泛的应用与其强大的功能相辅相成，密不可分。

在这本书中，你将看到 PHP 中最为精华的特征，包括类型系统、面向对象机制、数据库交互、安全性保证、内建函数库等。

通过书中极为实用的代码，上述特征的学习和应用将被无缝连接在一起。

作者 Peter B. MacIntyre 在软件开发领域已有超过 20 年的经验，曾是 PHP|Architect 杂志的特约编辑和作者。长期从事 PHP 相关的工作使作者对 PHP 的发展历程非常了解。这也使本书不单可以让人了解 PHP 当前是什么样子，也可以让人了解到它为什么是现在这个样子。

978-0-596-80437-4 PHP: The Good Parts © 2010 by O'Reilly Media, Inc. Simplified Chinese edition, jointly published by O'Reilly Media, Inc. and Publishing House of Electronics Industry, 2010. Authorized translation of the English edition, 2010 O'Reilly Media, Inc., the owner of all rights to publish and sell the same. All rights reserved including the rights of reproduction in whole or in part in any form.

本书中文简体版专有出版权由 O'Reilly Media, Inc. 授予电子工业出版社，未经许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易登记号 图字：01-2011-7092

### 图书在版编目 (CIP) 数据

PHP 语言精粹 / (美) 麦因泰 (MacIntyre, P.B.) 著; 刘涛, 丁静译. —北京: 电子工业出版社, 2012.3

书名原文: PHP: The Good Parts

ISBN 978-7-121-15385-3

I. ①P… II. ①麦… ②刘… ③丁… III. ①PHP 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2011)第 252567 号

责任编辑: 徐津平 文字编辑: 江 立

封面设计: Karen Montgomery 张 健

印 刷: 北京市顺义兴华印刷厂

装 订: 三河市双峰印刷装订有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 720×1000 1/16 印张: 11.25 字数: 275.2 千字

印 次: 2012 年 3 月第 1 次印刷

定 价: 39.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线: (010) 88258888。



# O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始, O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来, 而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者, O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”; 创建第一个商业网站 (GNN); 组织了影响深远的开放源代码峰会, 以至于开源软件运动以此命名; 创立了 Make 杂志, 从而成为 DIY 革命的主要先锋; 公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖, 共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择, O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版, 在线服务或者面授课程, 每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O'Reilly Radar 博客有口皆碑。”

——Wired

“O'Reilly 凭借一系列 (真希望当初我也想到了) 非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人, 他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了: ‘如果你在路口遇到岔路口, 走小路 (岔路) 。’ 回顾过去 Tim 似乎每一次都选择了小路, 而且有几次都是一闪即逝的机会, 尽管大路也不错。”

——Linux Journal

---

# 目录

*Table of Contents*

序 .....	ix
第1章 精粹 .....	1
为什么是PHP .....	1
PHP历史摘要 .....	1
PHP的地位 .....	2
PHP是什么 .....	2
PHP有哪些成就 .....	2
PHP基本设置 .....	3
第2章 “实地勘察” .....	7
空白、注释和基本语法 .....	7
变量：数据类型、弱类型和作用域 .....	9
定义常量 .....	11
表达式 .....	13
判断、选择（流程控制） .....	14
If...Else .....	14
Switch...Case .....	16
While .....	18
For .....	19
Web页面交互 .....	20
客户端 Cookie .....	21

Sessions .....	22
\$_GET .....	23
\$_POST .....	23
\$_REQUEST .....	24
<b>第3章 函数（代码复用） .....</b>	<b>27</b>
参数传递 .....	28
参数默认值 .....	29
传值和传引用 .....	30
include 和 require .....	31
内置函数和用户定义函数 .....	32
<b>第4章 字符串 .....</b>	<b>33</b>
什么是字符串 .....	33
你能引用我 .....	34
字符串函数（精选） .....	36
字符串截取 .....	36
管理字符的大小写 .....	37
字符串查找 .....	38
字符串编辑 .....	40
<b>第5章 数组 .....</b>	<b>43</b>
索引数组 .....	43
关联数组 .....	44
多维数组 .....	45
数组可以动态构建 .....	46
遍历数组 .....	48
数组函数精选 .....	49
数组排序 .....	49
数学类函数 .....	51
其他数组函数 .....	52



<b>第6章 对象 .....</b>	<b>57</b>
付诸实践 .....	58
魔术方法 .....	64
变量 \$this .....	65
对象实战 .....	65
公开的、保护的和私有的 .....	66
get和set访问器 .....	67
<b>第7章 数据库交互 .....</b>	<b>69</b>
MySQLi 对象接口 .....	69
取得数据并显示 .....	71
PHP数据对象 (PDO) .....	72
PDO 预处理对象 .....	74
低成本数据管理方案 .....	75
SQLite .....	75
用文件替代数据库 .....	77
<b>第8章 PHP周边 .....</b>	<b>87</b>
电子邮件/短信生成 .....	87
PDF生成 .....	90
构造方法和基本选项 .....	94
添加页眉和页脚 .....	94
添加图片和链接 .....	96
添加水印 .....	99
显示动态 PDF 文件和表格 .....	101
图形报表生成 .....	103
饼图 .....	103
柱状图 .....	106
图形验证码 .....	107
<b>第9章 PHP的安全性 .....</b>	<b>109</b>
数据验证 .....	109

转义输出 .....	111
跨站脚本 (XSS) 和SQL注入 .....	113
密码加密安全 .....	114
安全技巧 .....	115
<b>第10章 PHP 5.3 精粹 .....</b>	<b>117</b>
命名空间 .....	117
闭包 (匿名函数) .....	120
NOWDOC .....	121
goto 操作符 .....	122
DateTime 和DateTimeZone类 .....	124
额外的5.3特征 .....	129
<b>第11章 高级优势 .....</b>	<b>131</b>
正则表达式 .....	131
字符串匹配 .....	131
字符串替换 .....	133
字符串分割 .....	133
SimpleXML .....	134
集成开发环境 .....	137
ActiveState的Komodo .....	137
用于Eclipse的Zend Studio .....	137
NuSphere的PhpED .....	138
主要网站 .....	138
php.net .....	138
zend.com .....	139
devzone.zend.com .....	139
phparch.com .....	141
PHP/Web 会议 .....	141
<b>附录 缺点 .....</b>	<b>143</b>
<b>索引 .....</b>	<b>147</b>

酝酿这本书有相当长一段时间了。这么多年来，我一直在用 PHP，因为喜欢它易于上手、灵活和功能强大而投入越来越多的感情。在我 20 多年的职业生涯里，PHP 是我用过的所有语言中最喜欢和最拿手的。在此期间，PHP 也从一个小型函数集变成一个体积庞大、模块众多和扩展丰富的工具。有些程序员可能一开始会淹没在它浩如烟海的函数之中，但我希望通过这本书能帮助你真正了解 PHP 的行之有效。这本书并不厚，你将看到 PHP 开发中最精华的部分。当看到本书最后一页时，你会更进一步理解在 Web 开发领域里 PHP 是多么强大。

## 为什么是 PHP

市场上有那么多编程的书——包括大量关于 PHP 的书——你甚至都不知道也许正有本书即将完成。PHP 是被广泛使用的语言，并且近几年在企业应用方面也有较大的增长。Web 应用如 Facebook<sup>注1</sup>、Flickr<sup>注2</sup>、雅虎的部分网站、维基百科的核心实现，以及网站内容管理系统如 Drupal、Joomla 和 WordPress 也都是采用 PHP 构建的。IBM 还展示了很多自身技术与 PHP 相结合的有趣实例。基于这些原因，帮助技术业界的初中级程序员熟悉这个语言中最精华的部分是件有意义的事情。

## PHP 历史摘要

我们简单地回顾一下 PHP 的发展历史。PHP (Personal Home Page) 由 Rasmus Lerdorf 在 1995 年发布，最早的名字叫个人主页工具 (PHP Tools)。从推出到现在，一直是作为

注1：社交网站，部分地区访问可能有障碍。——译者注

注2：图片分享网站。——译者注



开源软件出现。数据库操作集成于 1996 的 2.0 版，之后其发展和变化可谓日新月异、翻天覆地。它成为当今世界上使用率最高的网站开发语言。在本书撰写之际，最新版本是 2009 年 6 月 30 日发布的 5.3 版。

## PHP的地位

PHP是一种使用最广泛的编程语言。想想看，在这么短的时间内有如此显著的成长，仅仅15年左右，它已经成为网站开发世界中的一个主要参与者。在最近几年，许多PHP社区的人都在争论它能否适用于企业开发：可信任否？可否用于大项目？够强壮否？鉴于近期也有如IBM和微软这样的公司在关注PHP，而且事实上，它可以构建大型网站（如Facebook和雅虎等），有人认为它也可用于企业开发。这些争论会随着时间的逝去而最终尘埃落定。对于最近发布的5.3版本，你可以十拿九稳地说，行或不行，将很快见分晓。

## PHP是什么

那么，什么是 PHP 呢？它是一种脚本语言，主要用于服务器端开发，可以被用来动态生成超文本标记语言（HTML）内容。PHP 和 Web 服务器集成在一起，较常见的是 Apache 或 IIS<sup>注3</sup>，一旦 PHP 完成 HTML 的生成，将交由 Web 服务负责向发起请求的客户端返回结果页面。

我说“主要用于”服务器端，是指你也可以将其用于其他领域，包括命令行、桌面开发和客户端服务环境，我只是举几个例子。但它最常用于 Web 服务器环境。

PHP 开发人员通常会将 PHP 和许多不同的数据库操作工具集成在一起，例如 MySQL、SQLite、PostgreSQL、DB2、MS SQL、Oracle 等。它们使动态内容成为可能。实际上，最终结果页面还是一个静态 HTML 文件，但它是在程序运行中产生的，因此是动态的。其实，你完全可以认为，由于内容是从数据库或其他来源读取并产生的，PHP 实际上是可以产生动态内容的。

## PHP有哪些成就

说了这么多 PHP 的优势，如果没有论据来证明，那显然没说服力。所以，让我们展现几

注3：这是两种较流行的 Web 服务器软件。——译者注

个用 PHP 搭建和实现的实例吧！世界排名较靠前的网站中有一些是部分基于 PHP 构建的。表 1-1 是使用 PHP 的流行网站简表，包括网站地址和简要介绍。

表1-1 使用PHP的流行网站

网站名称	介绍	网址
Facebook	社交网站	<a href="http://www.facebook.com">http://www.facebook.com</a>
Flickr	照片存储及分享	<a href="http://www.flickr.com">http://www.flickr.com</a>
Wikipedia	在线百科全书	<a href="http://www.wikipedia.org">http://www.wikipedia.org</a>
SugarCRM	客户关系管理	<a href="http://www.sugarcrm.com">http://www.sugarcrm.com</a>
Dotproject	项目管理工具	<a href="http://www.dotproject.net">http://www.dotproject.net</a>
Drupal	网站搭建模板引擎	<a href="http://drupal.org">http://drupal.org</a>
Interspire	资讯和邮件营销产品	<a href="http://www.interspire.com">http://www.interspire.com</a>

众所周知，这张表只是冰山一角，它肯定无法完整列出用 PHP 建造的网站，只是举出了几个代表而已。如果你还知道更多这样的网站，那你应该更加清楚 PHP 这门功能强大的语言究竟都能实现什么！

## PHP基本设置

这会儿，你或许跃跃欲试想要亲自体验 PHP 了。因此我们通过一个快速安装来实现一个随时运行都会显示“世界你好”的简单程序。

运行 PHP 代码的基本环境是使用最常用的 Web 服务软件，比如 Apache 和 IIS。有一种包含功能齐全的开发环境的软件包：LAMP 和 WAMP。LAMP 表示 Linux/Apache/MySQL/PHP，但这不是一成不变的，你也可以用 PostgreSQL，而不一定非要是 MySQL 数据库，甚至可以叫它 LAPP。另一副首字母组合——WAMP——是指 Windows/Apache/MySQL/PHP。



通常来讲，你编写的是和操作系统无关的代码。在 Windows 下编写的程序复制到 Linux 一样可以运行良好，反之亦然。但需要注意的是，对于操作系统级别的函数，如 CHMOD（更改文件权限）或 CHOWN（更改文件所有人），类似这样的代码在 Linux 和其他操作系统中可能会出现不同的结果，你一定要在不同环境中充分测试所有的代码实例。

既然有这么多的不同平台和组件可以建立 PHP 开发环境，我们不用再详细讨论这个问题了。总之你要知道，到 <http://www.php.net/downloads.php> 这个地址下载所有平台的 PHP 稳定版本。也有专为 Windows 定制的集多种软件于一身的安装包。一个是 XAMPP (X

4

表示多种平台，A 是 Apache，M 指 MySQL，第一个 P 表示 PHP，后一个 P 表示 Perl)，你可以在 <http://www.apachefriends.org/en/xampp-windows.html> 下载到。当下载合适平台的软件包之后，可以在里面找到一个叫 INSTALL.txt 的文件，其中包含了下载及安装过程指南。

一旦你完成了 PHP 的安装，你就可以通过编译一小段脚本来显示出你的 `php.ini` 配置文件所指定的配置信息，这样的代码只需一行，如下所示：

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

启动和停止位于 `<?PHP` 和 `?>` 之间的 PHP 代码内容的更多方法将在本书稍后几章中分别讲述。现在，将这行代码的文件取名为 `phpinfo.php` 并保存在你的 Web 站点根目录（一般叫 `www` 或 `htdocs`）。当你在浏览器地址栏中输入 `http://localhost/phpinfo.php` 时，输出的页面应如图 1-1 所示。

5

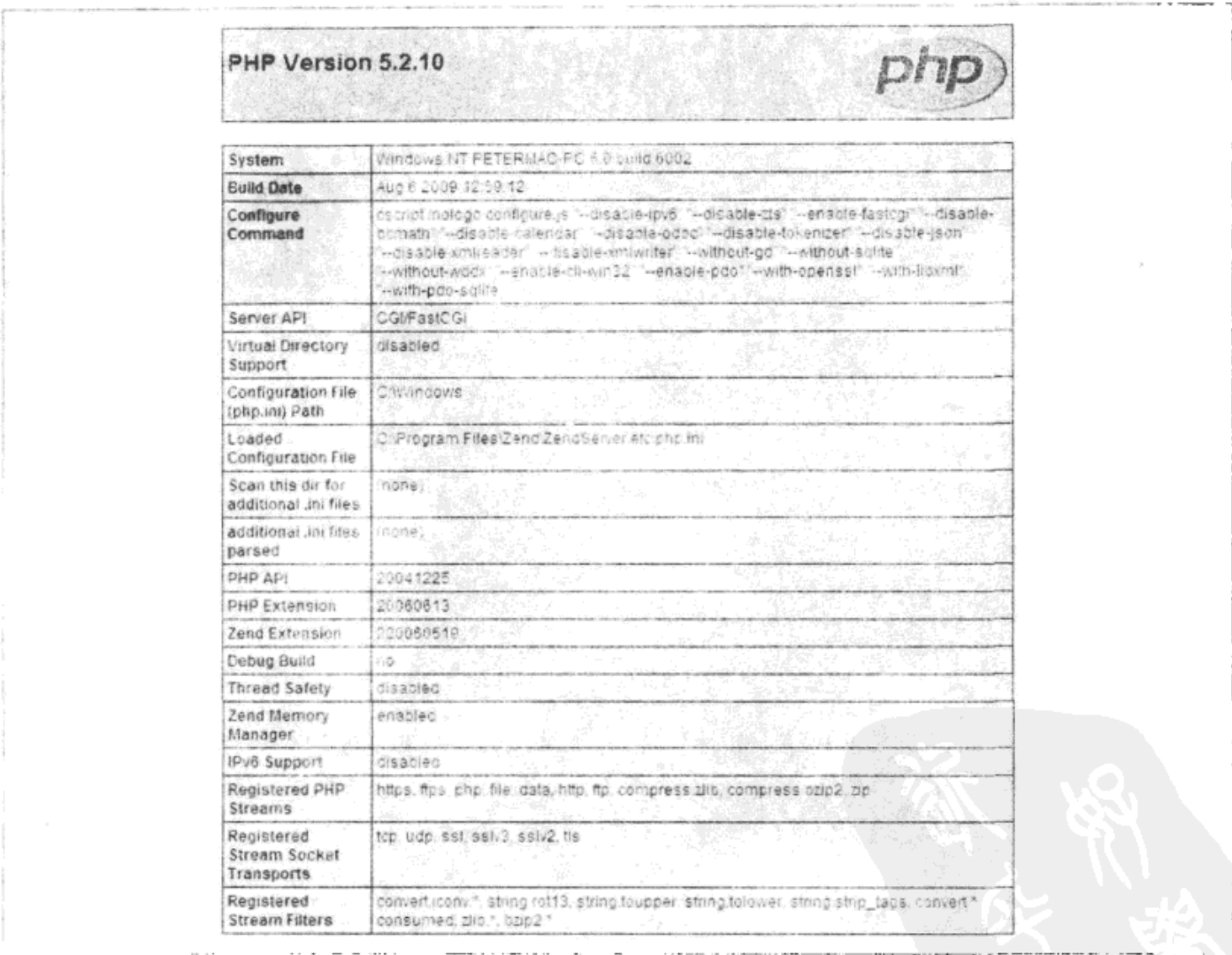


图1-1 `phpinfo()`函数的输入结果



请花一点儿时间来熟悉这些配置信息，如果你不确定其中的大部分信息，也别担心。只要页面显示的内容大致如图 1-1 所示那样，就足以表明 PHP 已经安装正确并通过你本地的 Web 服务器开始运行了。



localhost 是你本地电脑上用于 PHP 运行的 Web 服务器主机名。如果你有程序运行在一个远程服务器，则需要一个适当的域名或特定的 IP 地址来访问它。

现在我们就来写一句问候语。在网站根目录——一般在 Linux 下是 `/var/www`，在 Windows 下是 `./apache2/htdocs`——新建一个文本文件叫 `HelloOutThere.php`，用编辑器打开并输入下面这行代码：

```
<?php echo "Hello, is there anybody out there?" ; ?>
```

在浏览器地址栏中输入 `http://localhost/HelloOutThere.php`，显示的页面如图 1-2 所示。

6

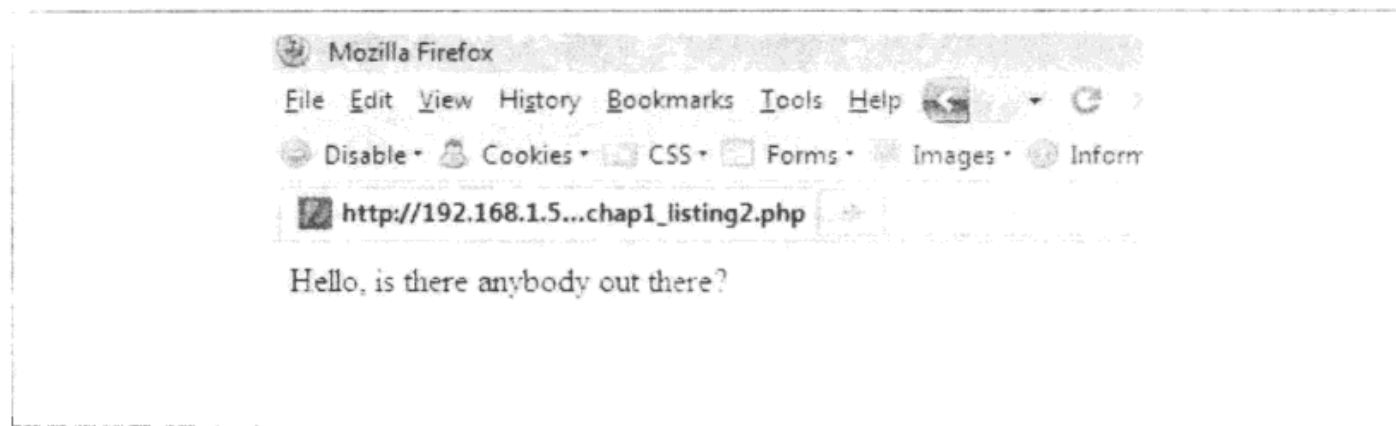


图1-2 实例输出结果

我们通过 `echo` 命令发送了一个字符串，要求服务器在浏览器的显示区域显示一些内容。在本书稍后几章中你将看到，我们可以在 Web 环境中处理很多东西。你刚刚创建了第一个 PHP 网页，真的就是这么简单。





现在，你已经知道了 PHP 文件的基本概念，以及如何通过 Web 服务器运行它并且将运行结果显示在浏览器中。接下来让我们继续研究如何用这门语言像搭积木那样来构建更大、更复杂的网站和网络应用程序。我称这个过程为“实地考察”，因为为了理解如何从根本上更好地处理 PHP 代码文件，我们将会粗略地看一下 PHP 的环境。如果你之前有过其他语言的编程经历，应该会对本章的学习感到得心应手，你所要做的就是充分理解并知道如何以及在何时使用这些结构元素。一开始，我们只是展示一小段一小段的 PHP 代码（比如变量和各种数据），稍后，我们将讨论如何操作控制语句相关的代码，也叫流程控制。最后，我们将研究可以说明完整的 PHP 应用程序环境的一些概念：内存数据存储（服务器与客户端）以及如何从内存中检索数据。

## 空白、注释和基本语法

对于 PHP 而言，解释器会忽略代码中所有的空白字符，所有的注释和空行在运行时都会被剔除。



如果你想用去除空白的办法来少许优化 PHP 代码或者想得到简洁的代码文件，其实不用花太多时间在整理代码格式上，PHP 中有一个函数叫 `php_strip_whitespace`，专门用来干这个。给这个函数传递一个文件名，它会返回去除所有注释和空行的干净代码，然后你就可以保存一个整洁的 PHP 代码文件。

看看下面这段代码，其中包含一些空白的部分：

```
1<?php
2 # 这是一行 PHP 注释
3
```



```

4 /*
5 * 这是多行 PHP
6 * 注释语句
7 */
8
9 echo " Hello my good web browser "; // 这是内联注释语句
10
11
12
13 ?>

```

第 1 行允许有空白，如果你在 PHP 开始标记（即 `<?php`）的后面添加空格（按空格键），将被视为空白。

第 2 行是个注释行，所以整行内容都将被视为空白。注释行表示代码注解，因而不会被解释器执行。第 4 行至第 7 行也是注释，但稍有不同，叫做多行注释，你能看到它以 `/*` 开始并以 `*/` 结束。PHP 解释器遇到这四行时会把它当成不需要执行的部分，即将其当成空白而完全跳过。

第 9 行会被执行，此行后面以 `//` 开头的部分是一个内联注释。PHP 解释器会对以 `//` 开头的部分视而不见。事实上，你完全可以把这样的注释加在语句末尾和分号之间，只不过这会让代码变得很难读懂（分号表示此行命令结束，如果漏加会报一个语法错误——即便你是一个经验老道的高手偶尔也会犯这样的错误）。

第 3 行、第 8 行和第 10~12 行的空行既不是注释也不是可运行代码。如果你清除所有的空白内容，那么上述代码就变成下面这个样子：

```
1<?php echo "Hello my good web browser"; ?>
```

正如你所看到的，在不同的命令或函数之间还有一个或多个的空格，和恰当的注释能帮助理解代码的道理一样，这也是为了使程序更易于阅读，毕竟你写出的代码要让自己或别人读得懂，所以，一定要养成在代码中适当留白的习惯。下面任意一种方法都可以在代码中添加注释：

```
#
```

用它来表示行内注释，此行内不允许再有任何可运行指令。

```
//
```

用它也表示行内注释，它可以自起一行，也可以附加在可执行代码的末尾。

9 `/* ... */`

表示多行注释块，块里的任何指令都不会被运行。

那么，一个基本的 PHP 语法包括一个 PHP 开始指示标签 [即 `<?php` 或 `<?`] 加上一个结束指示标签 [即 `?>`]。指示标签中间这部分内容会被 Web 服务器交给 PHP 解释器处理。



那种短一些的开始标签 `<?` 在 PHP 5 以后的版本中默认不允许使用，除非在 `php.ini` 的配置中将 `short_open_tag` 的值改为 `On`。这个约定是为了让人们尽可能使用更好且更完整的开始标签 `<? php`。

在这样一对标签里面<sup>注1</sup>，就可以开始 PHP 编程工作了。你可以使用四种不同的语言结构来编程：像前面 `echo` 那样的指令语句、函数调用（PHP 库里的或你自己编写的函数）、流程控制语句（如 `if...else...`）和注释。几乎所有的 PHP 程序都是由这四种简单结构组成的，当然了，一个完整的 Web 应用程序会大量使用这些语句，另外，PHP 还可以定义面向对象的类（详见本书第 6 章）。

为了构建更加健全的应用程序，往往还需要其他的元素，例如变量，下面我们就来看看变量及其使用规则。

## 变量：数据类型、弱类型和作用域

变量可以用来表示不同类型的数据，但它们建立的方式是一样的。下面就来说说 PHP 变量的定义规则：

**\$**

变量名一律以美元符号开头（\$）。

**大小写敏感**

变量名对大小写敏感，所以 `$firstname` 和 `$FirstName` 表示两个完全不同的变量。

**字母或下画线**

在美元符号 `$` 之后的第一个字符必须是字母或者下画线（`_`），其余的字符可以是字母、数字和下画线。

**`$this`**

除了在面向对象的 PHP 中，别处不会遇到使用 `$this` 的情况。

数据类型顾名思义就是指某种类型的数据。不同类型的数据在结构、可交互性及操作方式上都有不同的限制和约定。PHP 里有 8 种基本的（或叫原始的）变量类型。当然也可以自定义类型，但在这本书里只涉及 8 种。这些原始类型按数据分段存储方式又可分为

注 1：如果结束标签后无输出则可以省略。——译者注



三大类：标量类型、复合类型和特殊类型。表 2-1 列举了变量类型和分段方式及部分示例。

表2-1 PHP数据类型

分段	类型	解释 / 示例
标量类型	布尔型	逻辑“真”或“假”
	整型	全部整数：例如 1、15、-122 和 967967
	浮点型（双精度）	有小数点的数字（在金融财会领域常见）：例如 12.56 和 345.456
	字符型	各类文字、字母和数字（通常用双引号括起来）：例如“你好”和“123AvR”
复合类型	数组	包含键值对的集合，数组中可再包含数组（多维数组）；详见第 5 章
	对象	面向对象编程中已定义类的实例；详见第 6 章
特殊类型	NULL	未赋值的变量；已存在的变量，但不包含任何数据（不是空字符也不是 0，而是什么都没有）
	资源	表示对函数、数据库资源、文件或其他 PHP 外部资源的引用指针

有两种方式给变量赋值：传值和传引用。一般是通过直接传值来定义变量，例如：`$firstname = "Peter"`，我们就给这个叫 `$firstname` 的变量分配了一个包含 5 个字符的字符串，这个变量将一直保留其内容，直到重新分配或脚本运行完毕。没有什么因素能影响这个变量的内容，除非程序直接和它发生交互。

而传引用就好比是用不同的变量名来表示同样的变量内容，通过这种方式可以让函数在其内部影响外部定义的变量。只有事先以传值方式定义过的变量才能以引用的方式传递和访问。当然，只要引用一次即始终是引用方式。如果被引用传递的变量内容有变化，则所有本地副本所指向的引用会自动变更为新内容。实现一个引用传递很简单，只要在目标变量名前面加个 `&` 符号即可。下面的代码即可演示这个效果：

```
<?php
    $firstname = "Peter" ; // 分配值给变量
    $fname = &$firstname ; // $fname 是对 $firstname 的引用
    echo $fname . "<br/>"; // 显示内容 Peter
    $fname = "Dawn";      // 更改引用变量的内容
    echo $firstname . "<br/>"; // 显示为 Dawn 而不是 Peter,
                           // 因为是“引用”
?>
```

11

和其他编程语言不同的是，PHP 是个弱类型和动态类型语言。也就是说，PHP 可以很聪明地识别出分配变量时其存储的数据类型，像日期、字符串和数字等。因此在前面进行赋值的例子中：`$firstname = "Peter"`，PHP 可以判别 `$firstname` 为字符串类型。然而



预先定义各个变量的类型确实也是一种很好的实践，这种方法可以减少混淆。

变量的作用域涉及变量在一段代码中被另一段代码发现并操作的问题。在默认情况下，一个变量是在整个 PHP 代码中（整个 PHP 文件范围）可见的，但也有例外。如果一段代码被某个函数引用或包含之后，则这段代码就不能访问此函数外部的任何部分变量。下面我们通过具体实例说明一下：

```
<?php
function show_stuff() {
    $secondName = "Beck" ;
    echo "inside show_stuff: " . $firstname . " " . $secondName ;
}

$firstname = "Peter" ;           // 除了函数内部，此变量全局可见
//
echo $firstname . "<br/>"; // 显示 Peter
show_stuff () ;                 // 只会显示 Beck 因为 $firstname
                                // 不在函数的作用域里
echo "Outside function " . $secondName ;
                                // 仅仅是在函数 show_stuff 中有定义，
                                // 所以此处无法访问，
                                // 这里不会显示它的内容
?>
```

你也看到了，同样都叫 `$firstname`，但因为是在不同的作用域里，所以 `show_stuff` 函数不能访问到外部的变量，而它定义的 `$secondName` 也同样不能被外部访问。有关函数及如何改变这些行为的更多详情，请参阅第 3 章。现在，你只要记住：由于作用域的原因，有些代码可以自然地访问某些变量，而换个地方就不能访问。

## 定义常量

定义的常量就像是 PHP 变量的近亲。常量可以定义在代码文件的任意位置，但大多数人都是在代码文件和函数的开头去定义它。一旦定义之后会保持到代码运行结束，而且是全局有效，也就是说无论在函数或类里，甚至在任意包含的文件及函数里，常量都是可以全局访问的。定义一个常量的方法很简单，有点像分配变量，但又不完全一样。最大的差别是常量定义需要使用一个 PHP 内置的函数 `define()`。定义常量时，需要遵循以下规则：

### *define()*

要用这个 PHP 内置函数来定义常量。

## 字母或下画线

常量名称必须以字母或下画线开头，首字符以后的部分也只能由字母、数字或下画线组成。

## 大小写敏感

按照默认的约定，常量名应全部大写，当然你也可以不遵守这个约定，用 `define()` 函数中的第三个参数来定义大小写不敏感的常量。

## 限制

只有标量数据（参考“变量：数据类型、弱类型和作用域”部分）才能定义在常量里。所以，定义一个常量的语法如下所示：

```
define("常量名称", 常量值, [是否大小写不敏感])
```

此函数中最后一个参数大小写不敏感是可选项，默认为假，即定义的常量名称大小写敏感（这是个众所周知的标准规范）。获取一个已定义常量的值，只要引用常量名称即可。下面的代码将向您演示定义两个常量以及试图重复定义常量的问题：

```
define("SYS_OWNER", "Peter");
define("SYS_MGR", "Simon", true);
echo "System owner is:" . SYS_OWNER . "<br/>" ;
define("SYS_OWNER", "Michael");
echo "System owner is:" . SYS_OWNER . "<br/>" ;
echo "System manager is:" . SYS_MGR . "<br/>" ;
echo "System manager is:" . SYS_mgr . "<br/>" ;
```

上述代码输出（包含一个错误警告）如图 2-1 所示。

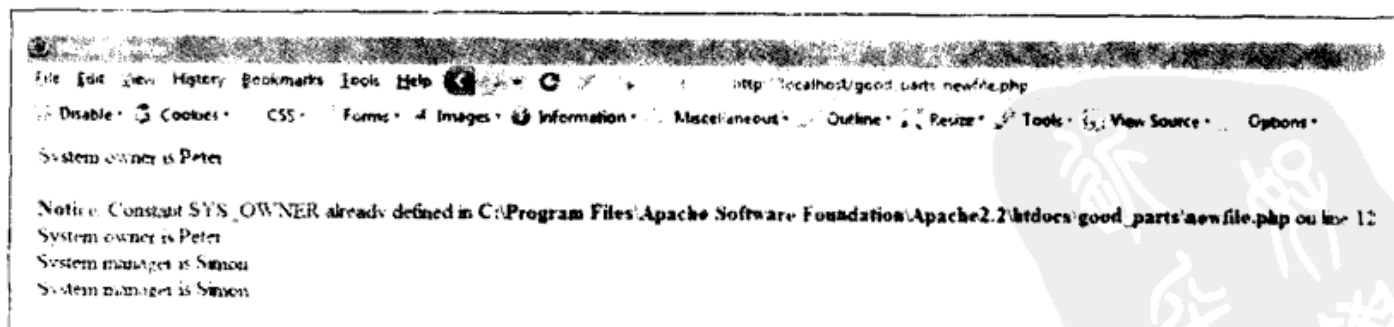


图2-1 定义常量示例代码的输出





如果 PHP 配置中错误报告被关闭，你可能看不到图 2-1 中的警告信息，可能会有意外的或不想要的结果，所以，你在将代码上传到产品环境以前，要确保代码经过测试。

毫无疑问，定义常量在 PHP 编程实践中占有一席之地，常量的价值是显而易见的，尤其是当你需要实现一个在代码进程中不被改变的值的时侯，如一个存储 PDF 文件的路径、计算需要的税率等。当你确实有这样的应用场景时，你会考虑用常量。但是一定要确保经过测试再上传到生产环境，只有这样你才能实现预期的效果。

## 表达式

PHP 中的表达式（不是正则表达式，那个是特例）是代码语句的统称。

```
$name = "Peter";
```

这行代码是个赋值表达式，它描述的意思是：分配一个字符串“*Peter*”给一个名叫 `$name` 的变量。从技术上讲，这行代码是一个由两部分表达式组成的声明（以分号为结束符）：左边的部分是一个存储的定义，右边的部分是要将什么值分配给那个存储定义。这两部分组成了一个赋值表达式，所以是个完整的声明。



作为一般规则，任何一个赋值语句的表达式，都是作为指令语句处理。

另外一些表达式包括函数和条件三元运算符（相关说明请见下一节条件判断代码），函数的返回值也包括在内。例如下面的代码就演示了这两种表达式：

```
function MyName () {  
    Return "Peter" ;  
}  
$name = MyName();  
$name ? $last = "MacIntyre" : $last = "" ;
```

函数 `MyName` 被当做一个表达式，因为它返回了一个值。你可能会觉得在赋值语句后面的那行代码有些奇怪，在以 `$name` 作为条件的前提下给 `$last` 赋值。我们将在本书中看到更多这样的表达式，现在你只要知道有这种语句存在并且在 PHP 中非常普遍就行了。

## 判断、选择（流程控制）

如果没有选择项就没有其他的可能性，人生将是多么枯燥无趣啊！下面我们来看看如何用流程控制语句来进行条件判断，也就是用它来根据预先规定的条件做出选择。

14

### If...Else...

基本 if 语句的测试目标可以建立在一个变量或执行其他简单（有时也会复杂）语句的运行结果的基础上。举一个使用了 if 语句的例子：

```
$today = date("l");
if ($today == "Wednesday") $tax_rate = $tax_rate + 4 ;
```



这里的比较判断使用的是双等号（看看是不是星期三）。我们已经知道赋值语句用的是一个等号，而判断是否相等要用双等号。

如果你需要进行数据类型级别的相等判断，可以使用 `===`，它会对比两边所有元素的内容及其类型，考虑下面的代码：

```
if (1 == '1') echo "true 1 等于 '1' <br/>";
if (1 === '1') echo "true 1 等于 '1'";
else echo "false 1 不等于 '1' " ;
```

它会产生下面的输出，显然，当用双等号比较字符串和数字时，字符串会先被转换成数字类型再进行比较，因此结果为真（true）。当用 `===`（三个等号）进行比较时，字符串不会被转换，它和数字类型不一致。上述代码的输出结果如下所示：

```
true 1 等于 '1'
false 1 不等于 '1'
```

在 if 语句中可以使用有效格式的数字。在前面例子的比较判断中，如果为真，则税率（变量 `$tax_rate`）加上 4，也就是说除了星期三以外的日子，税率保持不变。如果这个比较判断的结果为真你还要加入其他的语句，可以使用一对花括号。下面的代码是对前面例子的扩展，其中对日期判断进行了简化，去掉了变量赋值语句。

```
if (date("l") == "Wednesday") {
    $tax_rate = $tax_rate + 4;
    $wages = $salary * 0.4;
    $msg_color = "red";
}
```

该例实现的是如果判断条件为真则执行三条语句。请注意，这三条语句都包含在 if 后面的大括号里面，表明它们是这个判断结果中同等重要的部分。

你也可以添加当 if 语句中的条件不被满足时需要执行的代码，这个代码叫 else 子句，如下所示：

```
if (date("l") == "Wednesday") {  
    $tax_rate = $tax_rate + 4 ;  
    $wages = $salary * 0.4 ;  
    $msg_color = "red" ;  
} else {  
    $tax_rate = $tax_rate + 2 ;  
    $wages = $salary * 1.4 ;  
    $msg_color = "blue" ;  
}
```

15

如果 if 语句中的条件判断结果为假，则执行 else 后面花括号里的语句——具体这个例子来说，就是除周三以外的日期会执行这些代码。

你甚至可以在 if 语句中再嵌套别的条件判断语句。例如，如果今天是 6 月份的某个周三，进行判断的代码可以这样写：

```
if (date("l") == "Wednesday") {  
    $tax_rate = $tax_rate + 4 ;  
    $wages = $salary * 0.4 ;  
    $msg_color = "red" ;  
    if (date("F") == "June") {  
        $discount = $tax_rate * 0.15 ;  
    }  
}
```



嵌套会使代码显得笨重，为了防止出现晦涩难懂甚至无法运行的代码，你要慎用多层嵌套并控制嵌套层数。

下一步，可以用 elseif 子句来实现多层条件判断。举例来说，如果你需要进一步对一周的日期进行判断，以实现每天执行不同的程序任务，则会写出如下所示的代码：

```
$weekday = date("l") ;  
  
$tax_rate = 4 ;  
  
if ($weekday == "Monday") {  
    $discount = $tax_rate * 0.05 ;  
} elseif ($weekday == "Tuesday") {  
    $discount = $tax_rate * 0.06 ;  
} elseif ($weekday == "Wednesday") {
```

```

        $discount = $tax_rate * 0.07 ;
    } elseif ($weekday == "Thursday") {
        $discount = $tax_rate * 0.08 ;
    } elseif ($weekday == "Friday") {
        $discount = $tax_rate * 0.09 ;
    } elseif ($weekday == "Saturday" || $weekday == "Sunday") {
        $discount = $tax_rate * 0.10 ;
    }
    echo $weekday . "'s discount is: " . $discount ;

```

16

如果你在周四运行这段代码，会输出以下结果：

```
Thursday's discount is: 0.32
```

另外，注意在示例代码中使用了“或”（||）条件来判断周六和周日。

另一种实现条件判断的写法是使用三元运算符。这种格式虽然不使用明确的 if/else 语句，不过一旦你掌握了它，就可以写出很简洁的代码。暂时忘掉 if 语句，我们用这种方法来实现一个简单的税率判断操作，考虑下面的代码：

```
$tax_rate += date('l') == 'Wednesday' ? 4 : 2;
```

这行代码完整的意思是这样：如果当天是周三（?），则给税率变量加上 4，如果不是（else:），则只加 2。

你可能想了解例子中的 += 是怎么回事。PHP 允许在对数字变量做简单数学运算的同时做赋值操作<sup>注 2</sup>，像 ++、--、+= 等都属于这一类特性。要知道如何更好地利用这个特性，可具体查看 [php.net](http://php.net) 中的内容。

像这样的三元运算符通常仅限于处理多个条件判断中最终的一个结果（真或假），虽然也允许使用多层嵌套，但可能不会达到预期的目的，所以，最好保持这种简单直接的使用方式。

还有另外一些 if 语句的写法，它们也会运转得很好，但我们不用再探究更多的细枝末节。如果你有兴趣了解那些格式，请移步 [php.net](http://php.net)。

## Switch...Case...

针对单个值进行数个条件的判断，使用 if 语句来实现仍存在一定局限。在前面关于一周日期判断的例子中，只要我们愿意就可以让代码在 7 天里做 7 件不同的事。就这点来说，如果那天正好是周三，则只要在代码里定义一个特定的条件即可。诚然，如果我们针对

注 2：多数流行编程语言都有类似支持。——译者注



一周的每天都处理不同的税率，则可以写出类似这样的代码：

```
$today = date("l") ;
if ($today == "Monday")      { $tax_rate += 2 ; }
if ($today == "Tuesday")     { $tax_rate += 3 ; }
if ($today == "Wednesday")   { $tax_rate += 4 ; }
if ($today == "Thursday")    { $tax_rate += 5 ; }
if ($today == "Friday")      { $tax_rate += 6 ; }
if ($today == "Saturday")    { $tax_rate += 7 ; }
if ($today == "Sunday")      { $tax_rate += 8 ; }
```

17

但如果需要在每天的条件判断中执行更多的语句，这样的代码将会变得更加复杂和难以管理。这里我们引入 `switch...case...` 语句，你可以用这个控制结构操作更多种可能值的情况。语法如下：

```
switch ( 用来判断的值 ) {
    case 第一个可能的值 :
        // 一些代码
        [break;]
    case 第二个可能的值 :
        // 一些代码
        [break;]
    default:
        // 如果没有任何一个前面的条件为 true, 那么执行到这里
}
```

这个语法看起来有点儿令人难以理解，不过一旦你了解它的逻辑，你会觉得它确实很有用。括号内的值是用来判断的项，之后是对每种值的情况进行判断处理，最后，如果有其他可能，会放在 `default` 子句中处理。



`break` 语句是可选的，但如果不写上它，PHP 会在处理完某个值之后继续执行 `case` 语句的下一可能，因此，要在 `break` 起作用的地方小心使用它。

好了，对一周中的每一天进行处理的例子，最终就会变成这个样子：

```
switch ($today) {
    case "Monday" :
        $tax_rate += 2 ;
        $wages = $salary * 0.2 ;
        $msg_color = "red" ;
        break;
    case "Tuesday" :
        $tax_rate += 3 ;
        $wages = $salary * 0.3 ;
        $msg_color = "yellow" ;
        break;
```



```

case "Wednesday" :
    $tax_rate += 4 ;
    $wages = $salary * 0.4 ;
    $msg_color = "black" ;
    break;
case "Thursday" :
    $tax_rate += 5 ;
    $wages = $salary * 0.5 ;
    $msg_color = "green" ;
    break;
case "Friday" :
    $tax_rate += 6 ;
    $wages = $salary * 0.6 ;
    $msg_color = "orange" ;
    break;
case "Saturday" :
case "Sunday" :
    $tax_rate += 7 ;
    $wages = $salary * 0.7 ;
    $msg_color = "purple" ;
    break;
}
    
```

18

这里不必加上 default 子句，因为我们要判断所有可能的值。但请注意，周六和周日之间没有 break 语句，也就是说周末两天的税率处理过程是一样的。这只是一个例子，说明在需要的时候你可以去除 break 语句。

正如你看到的，switch...case... 结构有它的优点，虽说同样是条件控制语句，但这种结构能使代码更容易阅读和修改。

## While...

现在让我们来看看 while 语句。这个语句能让一段代码重复运行，直到条件不为真时停止。语法上有两种基本形式。首先是直接以 while 开头的形式，像这样：

```

$repeat = 1;
while ($repeat <= 25) {
    echo "the counter is: " . $repeat . "<br />";
    $repeat ++;
}
    
```

另一种语法形式是 do...while，如下所示：

```

$repeat = 0;
    
```

```
do {
    $repeat ++;
    echo "the counter is: " . $repeat . "<br />";
} while ($repeat <= 25);
```

这两种形式的主要不同点在于 `do...while...` 结构会至少执行代码一次。通过对条件进行判断来决定是否循环。在第一个例子里，有可能永远也不会执行条件后的代码。比如，假设第一行代码中 `$repeat` 的值不是 1，而是 27，会使条件判断结果为假，那么 `while` 循环将永远不会被执行。



要小心使用第二种语法 (`do...while...`)，因为你写的代码会至少执行一次，这可能是也可能不是你要达到的目的。

## For

19

`for` 循环逻辑结构和 `while` 结构系列在逻辑实现上有一些不同，你可能已经注意到，在上一节的例子中，用于控制计数器的变量 `$repeat` 必须手动以 `$repeat ++` 命令实现递增。如果使用 `for` 循环结构，计数器会直接写在条件判断行里，这样就会使代码稍微简洁一些。前面的 `while` 例子，下面我们用 `for` 循环结构来实现，殊途同归，效果完全一样：

```
for ($i = 0; $i <= 25; $i++) {
    echo "the counter is: " . $i . "<br />" ;
}
```

这个语句的第一部分 (`$i = 0`) 是设置 `for` 循环的初始值。分号后面的部分是判断条件，在每次循环时都会由这个判断决定是否继续执行循环语句。最后部分是定义每次循环时如何对条件变量进行改动，当然必须是在逻辑上有效的值。例如，它可以每次递增 5 或递减 12，这取决于编写代码的需要。用这个结构，我们把四行代码变成了两行，缩减了 50% 呢！



为了指出这是一个迭代，我还把变量变从 `$repeat` 改成了 `$i` ——这是大家常用于循环操作的变量名。继续使用 `$repeat` 也行，这里只是为了更紧凑才使用简短一点儿变量名。

在本书第 3 章中我们会讨论 `include/require` 结构，在第 5 章中会接触到 `foreach` 语句。另外，一定要重点学习第 10 章中提到的新控制结构 `goto` 语句。

# Web页面交互

PHP 的一个最大特点是可以帮你在集成 Web 服务器的环境下生成 HTML，不管是在 Apache 还是 IIS 或者是其他知名的 Web 服务器软件里。本节我们将看看 PHP 在处理请求时如何与 Web 服务器发生交互以及读写数据。图 2-2 表示的是 Web 服务器和 PHP 配置的基础结构。

这是一张 Web 服务和 PHP 适当组合的简单视图。当今的网站通过应用 JavaScript 和 Ajax 以及在其他技术的帮助下，已经表现得极具互动性。但实际上，网站依然是无状态的，也就是说一个网页做了什么并不会和别的网页发生必然关系。它们是彼此完全独立、隔绝的和无法直接交互的。然而，你可以将部分信息存储到服务器端的相应位置，然后在稍后的处理中取出这段信息，只要这段数据还未过期。

20

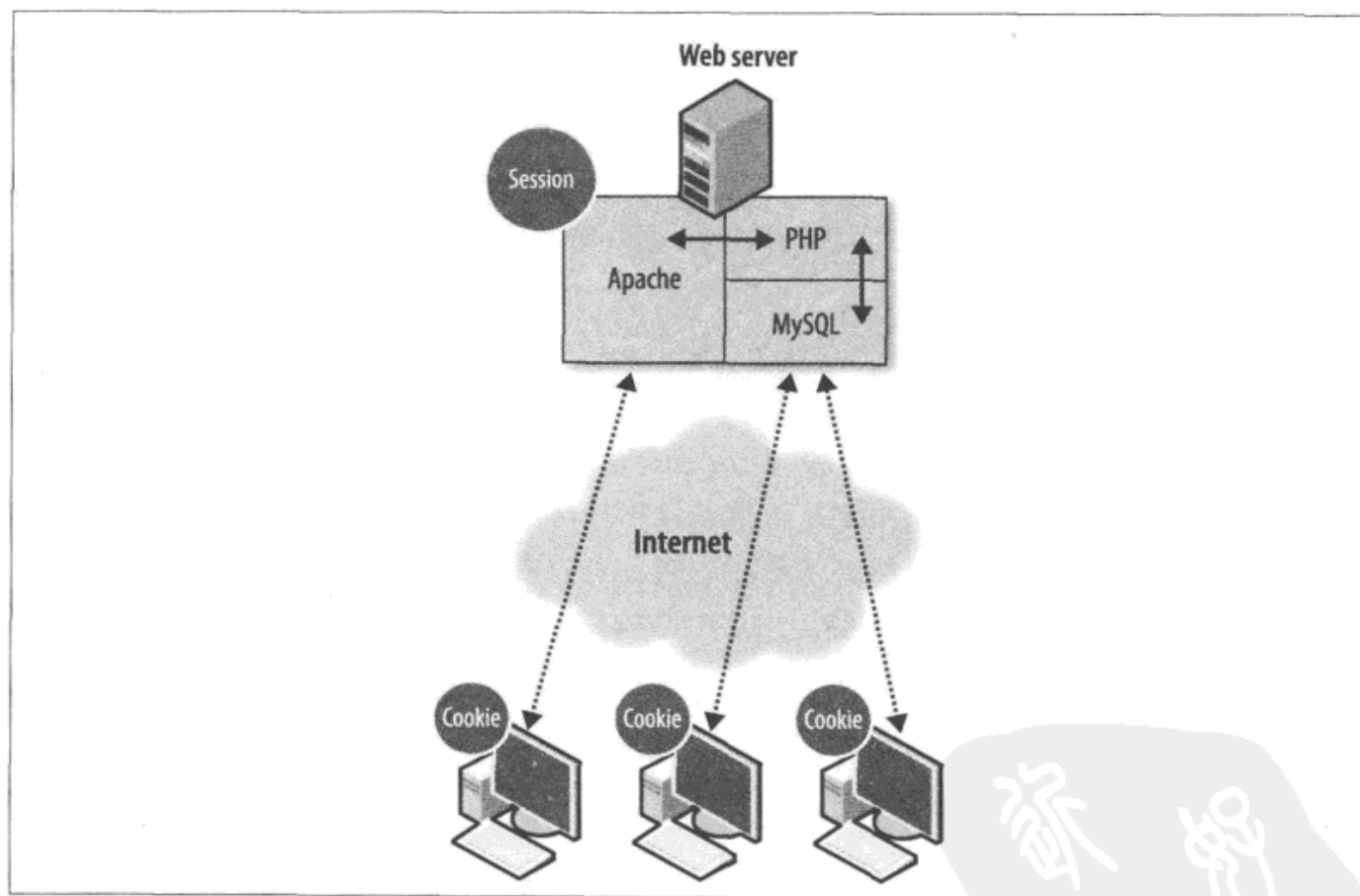


图2-2 Web服务器和PHP配置基础结构示意图

如图 2-2 所示，网页之间或 PHP 代码文件之间用来交互的变量数据可以存储在两个主要的地方：客户端的 Cookie 或服务端的 Session。PHP 尤其擅长处理这两个存储区域，

接下来的两节会详细解释它们的优势和不足。之后我们将在这些概念的基础上讨论如何利用好这些优势并发挥它们的作用。

## 客户端Cookie

我不是 Cookie 的超级粉丝。我很少使用它们，但它们也有自己的作用。每个网站的 Cookie 只是 Web 服务器存储在客户端电脑硬盘上的一个很小的文件。Cookie 有名字（用来标识）和表示的值，也有作为可选项的过期时间、路径和安全设置。下面的代码定义了两个 Cookie：

```
$data = "this will be placed in the cookie" ;  
setcookie("CookieName", $data) ;  
setcookie("AnotherCookieName", $data, time()+60*60*24*30) ;
```

第一个 setcookie 指令会在客户端电脑中建立一个名为“CookieName”的 Cookie，并将 \$data 变量的内容存储其中。由于未设置过期时间，这个 Cookie 会持续有效直到 Web 会话结束（一般为浏览器关闭）。第二个 Cookie 定义（名为 AnotherCookieName 那行）：和前一个例子一样，除了也有自己的名称和值，它还有个过期时间。这个过期时间的值基于 UNIX 的时间戳，所以这里你应该使用 time() 或 mktime() 函数来产生时间戳值。

21



由于 Cookie 是存储在客户端电脑上的，触发过期的条件是基于客户电脑的时间设置，而不是 Web 服务器上的时间。

上面的例子只是在客户电脑上设置 Cookie，另一个需要了解的是如何在你的代码或网站的其他页面中读取这些数据。本书中会出现 PHP 一系列的系统级变量，它们统称为超全局变量。这个顾名思义，就是在 PHP 的所有作用域中，不论是一般脚本、函数、类或者是其他外部包含的脚本中，它们通通都是可用的。第一个将要出场的就是马上要用来读取 Cookie 数据的超全局变量。这里我们用代码来实现读取前面定义的名为 AnotherCookieName 的 Cookie 值：

```
$newData = $_COOKIE["AnotherCookieName"] ;
```

超全局变量很好识别，它们总是由 \$\_ 开头，并且其余名称部分的字母全部大写。在这个例子里，\$\_COOKIE 提供了一个以名称 AnotherCookieName 来访问 Cookie 数组中的值的形式。只要这个 Cookie 存在（还未过期），就可以在任何需要用到的地方读取它。





当你建立 Cookie 时，最好象征性地设置一个过期时间，或者将其设置为 0，它会在浏览器关闭时过期。你也可以设置成一个过去的时间，这样它会被完全删除（而不是设置一个日期等它将来过期）。删除 Cookie 是在客户端电脑上进行的，从某种意义上说，作为一个不速之客，正确的事情是自己清理好自己留下的垃圾。

## Sessions

除了 Cookie 之外，另一个选择便是 Session。Session 本质上讲和 Cookie 类似，差别只在于它驻留在服务器端而不是客户端。通常来讲，其优势是 Session 不依赖于客户端电脑的资源<sup>注3</sup>，因此它在开发上往往会给你或其他程序员稍多一点控制。每个 Session 都是存储在服务器设定目录中的唯一文件，目录的位置由 `php.ini` 中的 `session.save_path` 控制。`php.ini` 中有相当多和 Session 有关的配置选项，要想尽可能高效地管理这部分的配置内容，你一定要先花一些时间了解它们。下面是一个 Session 文件名的例子：

```
sess_p6lhj0ih6hte5ar8kqmge629a6
```

文件名以 “`sess_`” 开头，然后是字母和数字的随机组合。这能让 PHP 在服务器上实现跟踪会话实例。Session 有效期内所有键 / 值对数据组合成一个关联数组，经过序列化后存储在该文件的内部结构中，一般来讲，这个有效期会持续到客户端浏览器关闭。要开始一个 Session，无论怎样，你都要使用 `session_start` 函数。如果是在一段时期内第一次调用这个函数，它会在服务器上存储一个空的关联数组；否则，会重新打开即有的 Session 文件并使数据可以访问。下面的示例是启动一个 Session 并保存一个值：

```
session_start( );
$today = date("Y-m-d") ; // 用 YYYY-MM-DD 的格式返回今天的日期
$_SESSION['today'] = $today ; // 添加这个日期到 Session
$_SESSION['login_name'] = "Peter" ; // 添加这个登录名到 Session
```

一旦 Session 建立完成，PHP 就可以通过它来处理浏览器生命期内文件运行环境的有效连接。



最新的浏览器都具备多标签浏览功能，允许在不同的标签内浏览不同的网址。如果在不同的标签内打开同一个网址，PHP 会认为它是同一个访问者，这样 Session 的内容就会在同一标签中共享，但这可能会导致一些意想不到的后果和行为。

如果在浏览器有效期内访问随后的其他页面，你可能想要读取即有 Session 的信息，你需要做的就是再次启动 Session，访问你感兴趣的数组键，像这样：

注 3：Session 多数情况下需要存储一个 Cookie 来支持。——译者注



```
session_start();
$loginName = $_SESSION['login_name'];
echo $loginName . " is now logged in";
```

这个名叫 `$_SESSION` 的超全局数组其实就是一个数据中转站，借助它数据将会在网站的不同页面之间穿插。作为一个程序员，你会发现 Session 技术会比 Cookie 技术提供给你更多的可控性。因为你可能恰好不擅长处理客户端浏览器编程环境。

## \$\_GET

下一个要讨论的超全局变量是 `$_GET`。这个值是由通过 URL 访问时的查询字符串而自动建立的，也可以通过 GET 方法的表单提交（仍然是使用 URL 作为媒介）来产生。被调用的页面运行时会将随查询字符串发送来的任何键 / 值对信息附加在 `$_GET` 数组变量里。

23

*http://www.mynewwebsite.com/access.php?login=1&logname= Fred*

这个网址的查询字符串中有两个键，一个叫 `login`，另一个叫 `logname`。当 `access.php` 这个文件被调用时，你可以根据需要操作这些值。通过键名来访问这个关联数组即可得到其值。考虑一下 `access.php` 文件中以下部分代码：

```
$login = $_GET['login'];
$logname = $_GET['logname'];
if ($login == 1) {
    echo "Welcome " . $logname;
} else {
    echo "login failed... please try again " . $logname;
}
```

用 `$_GET` 这个超全局变量的好处是，你可以在一个被调用的文件中使用指定页面的请求参数信息。



理解 `$_GET` 在每个页面调用时都会被刷新这一点至关重要，也正因为如此你必须把它传递到每一页进行向下调用的堆栈中。这一点和 Session 的概念有所不同。

## \$\_POST

`$_POST` 超全局数组在有效地从一个页面到另一个页面传递数值方面几乎和 `$_GET` 完全一样，所不同是传递信息的方法。`$_POST` 不使用被调用页面中 URL 查询字符串的办法，

而是对服务器使用超文本传输协议 (HTTP) 中的 POST 方法 (提交数据)。普遍来讲最常见的形式是在网页中用表单提交, 但也不是非要依赖于 HTML `<form>` 标签才能实现。另外, 由于它要向服务器 POST 数据, 提交的信息也不必作为 URL 的一部分, 对用户来说, 可见信息更少一些, 这或多或少会提升一些安全性。下面的代码演示如何在一个简单的网页中使用 `<form>` 标签以及如何在 PHP 文件中操作 `$_POST` 数组。

```
<html>
<head></head>
<body>
<form method='post' action='page2.php'>
please enter your name: <input type="text" size="15" name="fullname">
<input type=submit value="submit">
</form>
</body>
</html>
```

24 ➤ 当用户单击“提交”按钮时会调用 `page2.php` 文件, 其代码如下所示:

```
<?php
$name = $_POST['fullname'] ;
echo "the full name: " . $name ;
?>
<br/>
<a href="demo.php">back</a>
```

## \$\_REQUEST

本章中要讨论的最后一个超全局数组是 REQUEST 数组, 这是一个包含各类请求数据的数组, 即 `$_COOKIE`、`$_GET` 和 `$_POST`。美中不足的是, 数组中键名必须唯一, 否则对于 `$_REQUEST['lname']` 不能确定是来自前三者中的哪一个。看看下面的代码:

```
<html>
<head></head>
<body>
<?phpsetcookie('mname', 'Beck') ;?>
<form method='post' action='page2.php?fname=peter&lname=smith'>
<input type="hidden" value="Simon" name="fname">
please enter your last name: <input type="text" size="15" name="lname">
<input type=submit value="submit">
</form>
</body>
</html>
```

在上述代码中, 设置了一个 Cookie, 通过 POST 方法提交表单, 并且在表单提交时, 通

过 URL 的 GET 方法发送了一些数据。然而在实际的代码中出现这种情况的几率并不大，但它表明在使用 REQUEST 数据时有可能出现的意外情况。<form> 标签中的 action 属性指向 *page2.php*。下面是该文件的代码：

```
<?php
$fname = $_GET['fname'] ;
$lname = $_GET['lname'] ;
echo "the full name from GET: " . $fname . " " . $lname ;
$fname = $_POST['fname'] ;
$lname = $_POST['lname'] ;
echo "<br/>the full name from POST: " . $fname . " " . $lname ;
echo "<br/> the Request array -> " ;
var_dump($_REQUEST) ;
?>
<br/>
<a href="demo.php">back</a>
```

假设用户在前面表单中的 lname 输入框中写的是 “MacIntyre”，当这个页面显示时，能看到下面的文字：

```
the full name from GET: peter smith
the full name from POST: Simon MacIntyre
the Request array -> array(3) { ["fname"]=> string(5) "Simon" ["lname"]=>
string(9) "MacIntyre" ["mname"]=> string(4) "Beck" }
back
```

25

正如你看到的那样，尽管我们对 GET 和 POST 设置了不同的值，但它们有相同的键名，所以 REQUEST 数组默认按照顺序返回 POST 数据中的值。另外要注意的是，我们没有明确地引用前面代码中所设置的 Cookie 值——它在以后请求和调用 PHP 文件时会自动附加到 REQUEST 数组。



你可以通过 *php.ini* 中的配置指令 *variables\_order* 来控制整个环境中的超全局数组。我的服务器上配置的是 GPC，即 PHP 在加载后依次建立的超全局数组是 GET、POST 和 COOKIE。就如同它们开头字母一样，排在后面的要优先于前面的。“G”代表 GET，“P”代表 POST，“C”代表 COOKIE。如果你删除了其中的一个字母，保存后重启你的 Web 服务器，则那个字母所代表的超全局变量就不会自动建立。



## 函数（代码复用）

就像其他任何编程语言一样，PHP 也有函数。而且，知道如何编写大部分的函数将必然成为你的优势。

PHP 中定义的函数有两种形式：一种有返回值，一种没有。函数应由尽可能独一无二的代码段组成。函数定义的规则也很简单：使用保留字 `function` 跟一个唯一的单词作为函数名称，这个词可以由字母或下画线开头，后面可以跟任意个字母、数字或下画线。接着在函数名后面加一对括号 `()`——也就是用来传递数据的参数（稍后再细说）。最后，用一对大括号 `{}` 将函数要执行的代码括起来。

以下是一个简单的函数示例：

```
function MyFunction ( ) {  
    echo " 显示这行是由于名为 MyFunction 的函数被调用了 " ;  
}
```

定义一个函数和调用一个函数是有区别的。上面的代码仅仅是实现了函数的定义，而没有调用和激活的语句。下面看看定义并调用一个函数：

```
function MyFunction ( ) {  
    echo " 显示这行是由于名为 MyFunction 的函数被调用了 " ;  
}  
  
MyFunction ();
```

如果你不需要返回值，上面的代码也会正常运行。它会简单地打印一句话，“显示这行是由于名为 `MyFunction` 的函数被调用了”。



## 参数传递

让我们来看看函数其他几个方面的特点。刚刚提到过,函数可以接收传递给它们的值(参数),也可以返回值。

28



通常来讲,实践一个函数最好的做法是:由调用它开始(就像前面的例子)“进入”,在完成预定的工作后返回或不返回值来结束函数调用。相对不好的做法是:依据不同的条件返回不同的东西。之所以说这种做法不好,是因为它增加了不必要的复杂性,因此更难维护和调试。<sup>注1</sup>

向一个被调用的函数传递参数时,所传递的变量名不必和定义函数时使用的相应位置的名称一致,函数内部会将所传的值按照参数列表中相应位置进行分配。下面的示例代码中定义了两个不同的函数,以此来说明这一特点:

```
function MyList ($first, $second, $third) {
    echo "here is first: " . $first . "<br/> ";
    echo "here is second: " . $second . "<br/> ";
    echo "and here is third: " . $third . "<br/>";
}

function AddThese($first, $second, $third) {
    $answer = $first + $second + $third ;
    return $answer ;
}

MyList ("Peter", "Chris", "Dean") ;
echo "<br/><br/>";

$first = 5 ;
$second = 34 ;
$third = 237 ;
$math = AddThese($first, $second, $third) ;
echo "$first, $second, and $third add up to: " . $math ;
```

当你通过浏览器请求运行这段代码时,输出结果如图 3-1 所示。

第一个函数:MyList,允许通过 3 个分别叫 \$first、\$second 和 \$third 的变量来传递 3 个人名的文本值给它。只要运行到函数内部,这 3 个变量就会被打印出来进而显示在浏览器屏幕上。函数没有对这 3 个值做更多处理,调用时也没有返回值。

29

注 1:作者主要的意思是说,单纯的函数比复杂的函数要好。——译者注

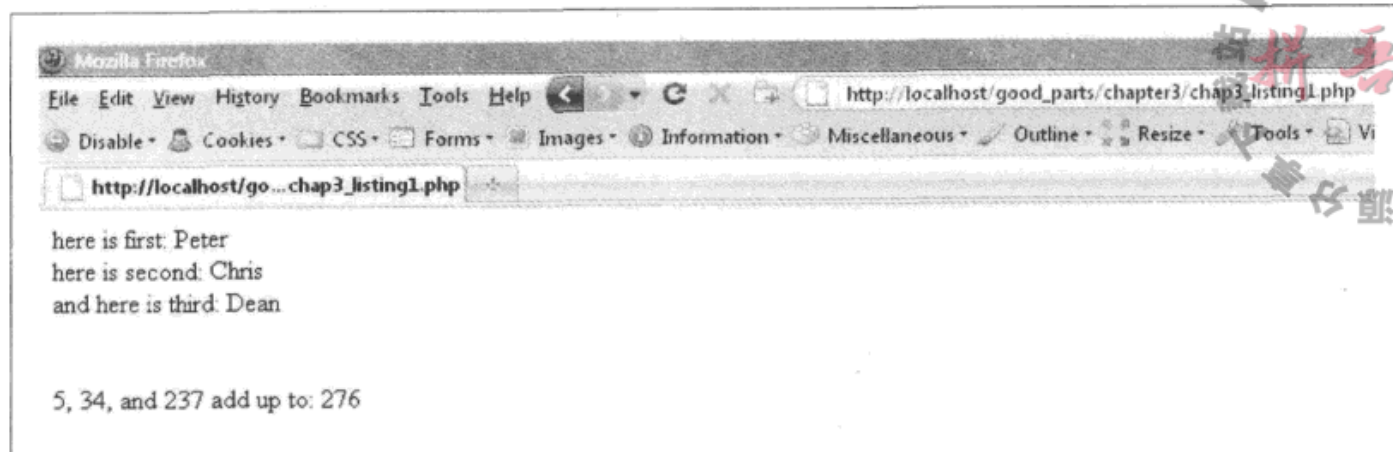


图3-1 代码运行的输出结果

第二个函数：`AddThese`，也是接收3个值。在当前例子中，它们都是数字，但函数在定义时并未对此做出区分（没有指定数据类型），所以假设这段代码中函数被调用时，3个变量被分配成数字值后传递给函数，该函数计算3个值的和，然后返回求和的结果，最后这个结果被存储在名叫`$math`的变量中，之后`AddThese`函数也未进一步处理这3个值。

请注意，这两个函数只是都把各自参数变量命名为`$first`、`$second`和`$third`，从某种意义上讲，这是两套独立的参数，虽然名字一样（`$first`、`$second`和`$third`），但它们互不干扰。正如你从输出结果中看到的，函数外部的`$first`、`$second`和`$third`这三者的值不会因为被传递到`AddThese`函数内部而改变或受到影响。

## 参数默认值

*Default Parameters*

再看另一种情况，你可以给函数中的某个参数定义一个默认值。借此之力，你可以让函数在不提供（全部或部分）参数时也可运行。考虑一下对`AddThese`函数做以下修改：

```
function AddThese($first = 5, $second = 10, $third = 15) {
    $answer = $first + $second + $third ;
    return $answer ;
}

$first = 5 ;
$second = 34 ;

$math = AddThese($first, $second) ;
echo "$first, $second, and $third add up to: " . $math ;
```

调用这个函数对 5 和 34 做相加运算，完全不需要第三个数字就可以得到结果 54。

实际情况是，当调用函数 `AddThese` 时，没有向它发送第三个参数。由于函数定义时指定了参数的默认值，当某个参数未传递时，会使用这个默认值。所以，可以借此规则来构建一些有相当弹性的函数。在本实例中，整数 15 会作为第三个值，使得运算结果是正确的，尽管显示出的文字容易使人产生误解。参数是按照相应位置进行接收的，因此，就算你在调用时把发送的参数变量名改为 `$forty` 或 `$fifty`，在函数看来，它们仍然被当做 `$first` 和 `$second` 接收。

## 传值和传引用

在默认情况下，所有的函数都仅仅是把参数变量的值传递给函数内部代码。这就意味着，你可以创建一个函数并接收一个叫 `$message` 的参数，但实际调用时传递的变量名叫 `$note`。只要在同样的参数顺序位置（不论是函数调用还是函数运行时），都可以照常工作。事实上，在函数内有自己的作用域，即使内外的变量名完全一样，也会被视为不同的变量。请看下面的简单示例。这里，名为 `$message` 的变量被传给一个叫 `displayit` 的函数，由参数接收到函数内部时变量命名为 `$text`，这样就把外部变量的值传进了函数。

```
function displayit ($text) {  
    echo $text ;  
}  
$message = "say hello to the web world";  
displayit($message) ;
```

在有些情况下，你会需要让变量及其值受到函数内部行为的影响。这时，你可以向函数传递变量的引用（第 2 章提到过）。就是告知 PHP 在将这个变量的值传递给函数的同时将此变量的作用域延伸到函数内部，这样在函数调用完成时，针对这个变量的任何变更将同步跟进。你所需要做的就是，在定义函数时将需要作为引用传递的变量前面加上一个 `&` 符号。当然，它仍然遵循前文中的顺序规则。下面是个实践的例子：

```
function displayit (&$text) {  
    $text .= ", you know you want to";  
}  
$message = "say hello to the web world";  
displayit($message) ;  
echo $message ;
```

以上代码的运行结果在浏览器中输出如图 3-2 所示。



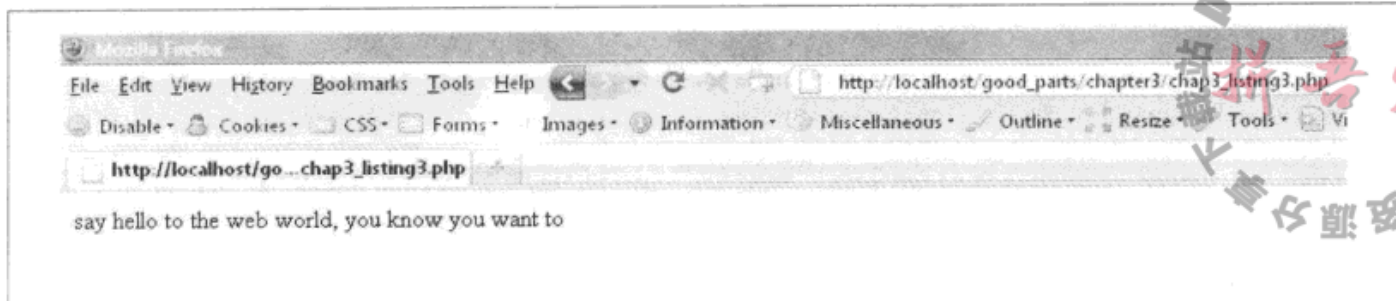


图3-2 函数传递引用示例运行结果

## include 和 require

31

函数定义好了，设计并撰写完成后需要多次使用。当你建立了一个函数并想在多个程序文件中用到，如果每次都把函数代码复制到各个想要用到它的文件中，那是件很痛苦的事情。幸好 PHP 允许在代码中插入别的代码文件，从而省去需要在多个地方复制替换代码的麻烦。这可以通过 `include`（意为包含）和 `require`（意为必需）语句实现（也是第 2 章未及讨论的一种流程控制结构）。当你在代码中定义了一个或多个函数，比如文件命名为 `my_functions.php`，你可以在其他需要使用这些函数的文件中指示 PHP 插入这个文件。

`include` 和 `require` 都可以实现包含指定文件名的内容。所不同的是，用 `include` 时如果指定的文件无法定位，代码会继续运行；但用 `require` 时如果未找到指定的文件，代码会停止运行并抛出一个致命错误。这两种情况下，文件未找到时都会报错，但只有 `require` 语句会完全终止代码的运行。

让我们用一个示例来说明如何让前一节中定义的 `displayit` 函数（保存并命名为 `my_functions.php`）在别的代码中使用（这个文件另保存为 `display_msg.php`）。

```
#####
# my_functions.php file
#####
function displayit (&$text) {
    $text .= ", you know you want to";
}

#####
# display_msg.php file
#####

include "my_functions.php";

$message = "say hello to the web world";
displayit($message) ;
echo $message ;
```



不管这个函数已经被多少文件引入 (include 或 require)，当你需要修改 displayit 函数时，只要打开 `my_functions.php` 这个文件修改即可。

32



PHP 在操作包含 (include) 和必需 (require) 指令时依据一定的顺序对指定文件进行查找。首先，逐个查找 `php.ini` 中 `include_path` 指定的路径位置，如果文件未找到，PHP 会在当前运行文件所在目录进行查找，当然了，如果你指定一个完整的文件路径，那样就不依赖于任何环境配置了。

PHP 中还存在两个非常类似的语句，分别叫 `include_once` 和 `require_once`。这两个带有 `_once` 后缀的语句其工作原理和“原身”完全一样，但它们可以保证对所包含的文件只引入一次，从而节省资源并避免可能出现重复定义和执行的问题。

希望你能从这些小小的示例代码中看到一旦开发出函数库对于 PHP 程序员来讲是相当有价值的。这种创建代码并多次重用的过程被广泛应用在面向对象的设计和开发中，具体请参见本书第 6 章。

## 内置函数和用户定义函数

至此，你已经对函数有所了解并可以根据特定需求来编写函数代码，这被称为用户定义函数 (UDF)。此外，PHP 还有大量的内置函数可供你在实际应用中使⽤，像字符串处理、数组管理、数据库连接、日期和时间信息操作等。所以一定要多看 PHP 手册和官方网站，说不定你马上要写的函数早已在 PHP 中存在了呢。另外要记住，原生的或者说内置的函数，总是比你用 PHP 代码写的函数快得多，因为它们针对 PHP 做过高度优化<sup>注2</sup>。但请注意，有些很特殊的函数依赖于 PHP 中添加的特定模块，比如 MySQL 数据库交互函数，你必须在 PHP 环境中添加 MySQL 库才可以正常工作。

注2：绝大多数内置函数都是本地编译的模块。——译者注

# 字符串

字符串被认为是网页输出中使用最广泛的形式之一。字符串由字母、数字、特殊字符组成，在 PHP 中，可以很轻松地分割、截取、拼接、组合字符串。我们已经在第 1 章和第 2 章中看到了一些关于字符串输出的例子。在本章中，我们将着重讲解字符串的处理。

字符串处理非常重要，回顾一下你这周访问过的网站，并试想一下有多少网站是基于文本的而不是基于图片或者视频的，即使像 YouTube 和 CNN 这样的网站也是严重依赖于文本内容，以便于与用户沟通，因此，让我们先来看看字符串是如何组成的，并考虑如何能够通过浏览器获取内容。

## 什么是字符串

如上所述，字符串是一种包含字符的简单集合。这些集合可以通过 `echo` 或者 `print` 语句输出到浏览器，把你想要显示的字符集合包含在特定的标记（通常是单引号或双引号）里，好让 PHP 明白哪部分是字符串。



尽管 `echo` 和 `print` 之间的差异很小（`print` 输出完成时返回 1，只支持一个参数，而 `echo` 支持多个参数），但笔者仍偏爱使用 `echo` 命令。`echo` 也能够用 PHP 短标记和等号（`=`）组合（只要 `php.ini` 配置文件中的 `short_open_tag` 设置为 `on`，但默认它是 `off`）实现，像这样：

```
<?="sending out some text"; ?>.
```

选择哪一种方式是个人喜好，但我推荐你使用上述这种方式，这么做能使你的代码保持一贯性。

# 你能引用我



字符串能被包含在单引号、双引号或者这两者组合标记中，还可以包含在 HEREDOC、NOWDOC（更多细节之后会讲解）中。如果你的字符串里含有变量，那么最好使用双引号将其括起来。

考虑下面这个简单示例：

```
$fname = "Peter" ;
$lname = "MacIntyre" ;
$string = "The first name of the author of this book
is $fname and his last name is $lname";
echo $string ;
```

在这里，定义的变量 `$string` 中提到了另外的两个变量名，PHP 插入它们的内容，然后通过 `$string` 变量输出它们，我们也能够使用单引号来实现，但是我们不得不用字符串连接操作符（即 `.`）来连接它们，因为单引号中的变量不会被解析（变量内容嵌入）。

接下来的这段代码即为单引号应用示例：

```
$fname = "Peter" ;
$lname = "MacIntyre" ;
$string = 'The first name of the author of this book is '
. $fname . ' and his last name is ' . $lname ;
echo $string ;
```

## 强大的转义符

如果你想要在字符串中插入双引号字符本身，则在它前面加一个反斜线即可，如下所示：

```
$string = "He said \"go away\" to me" ;
```

这就是所谓的字符的转义。这样 PHP 会在字符串中保留第二个双引号及其括起来的部分。或者，你可以使用单引号包含字符串，并实现完全相同的结果：

```
$string = 'He said "go away" to me' ;
```

你也能使用反斜线转义其他的字符，当你使用单引号时，用反斜线转义单引号和反斜线（虽然这样做使工作变得复杂了，例如，在 Windows 下创建一个文件夹路径的字符串）。在特殊情况下，双引号引用允许你转义一个完整的字符列表，例如，当你使用双引号引用字符串时，能够通过反斜线转义一个美元符号（`$`）并把它输

出到浏览器，以使 PHP 不会误认为你是在输出一个变量的内容。

除此之外，你还可以在其他地方使用反斜线，例如，当你使用双引号引用 `\n` 字符组合时，PHP 把它看做是一个换行符（同样使用单引号它就不起作用了），访问 <http://www.php.net/manual/en/language.types.string.php> 查看所有的转义序列。

此外，你也能使用双引号和单引号的组合来构建一个字符串，我们学到了如何在双引号中插入字符，也学到了字符串中的换行符是如何被双引号和单引号解释的，当换行符被单引号引用时，它是不起作用的，然而在双引号里它会起作用。

接下来的代码片段证明了这一点：

```
echo 'This sentence will not produce a new line \n';
echo "But this one will \n";
```

你可以很灵活地使用引用组合，大胆地测试你所完成的代码。

还有一种名叫 HEREDOC 的结构也能够创建字符串，就解析变量的意义而言，这种结构和使用双引号非常相似，在此之外它也适用于创建很长的字符串，因此，在程序中使用它会让程序更易于阅读，HEREDOC 使用三个小于号（<）和指定的名称开始，然后是字符串，另起一行使用指定的名称和分号结尾。

这儿有个例子：

```
$string = <<< RightHERE
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Fusce eget nisl a metus rhoncus placerat ac ac nisl.
Fusce consectetur tempus "tincidunt. Proin congue
dapibus neque", at congue lectus volutpat in.
Duis commodo, est tempor aliquam molestie, odio dolor fringilla arcu,
nec iaculis est libero vitae erat.
RightHERE;

echo $string ;
```

以上 HEREDOC 代码示例的输出结果如图 4-1 所示。

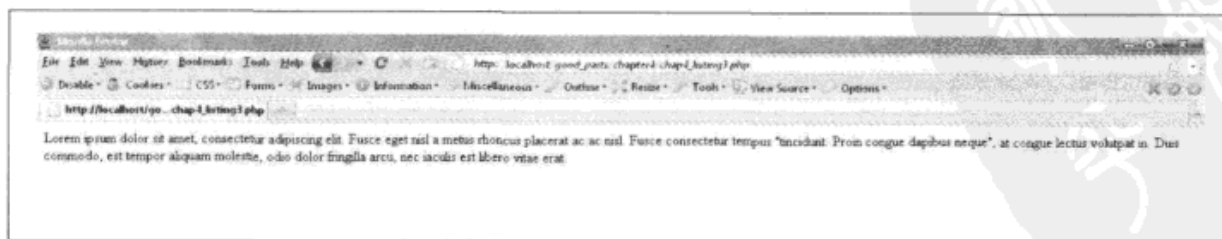


图4-1 HEREDOC示例的输出





你会发现 HEREDOC 结构提供了一种很好地创建结构化查询语言 (SQL) 语句的方法, 这种技术将会在本书第 7 章讨论数据库时广泛地使用。

正如你所看到的, 有很多种方法来定义字符串, 它们操作字符串是等价的, 注意与手动创建方式不同的是, 字符串也能从外部传递到你的代码中, 数组 (字符数字组合)、姓名、电话号码、邮编、E-mail 地址等信息都能通过 `$_POST` 或者 `$_GET` 方法, 将表单字段内容传递到代码里 (详见本书第 2 章)。

## 字符串函数 (精选)

—— 字符串函数 (精选)

下面我们来看看 PHP 中最好和最常用的字符串函数, 你需要这些工具以便于管理需要处理的问题, 我已经给这些函数按逻辑分了组, 大部分都给出了实例。在大多数实例中, 我们就像大海捞针, “针” 好比是我们要查找的字符串, “大海” 好比是我们要执行每次操作目标的全部内容。



在接下来的几章里你可能会注意到很多函数并不一定遵循普遍风格或者命名规则, 主要因为 PHP 是一个开源产品, 这么多年影响了很多, 但也要归因于一些函数是基于 C++ 相似风格命名的, 有时, 你会看到有些函数带下划线, 如 `strip_tags`, 同时又有很相似的函数没有带下划线, 如 `stripslashes`, 这种弹性既是 PHP 的优势, 也是它的劣势。

## 字符串截取

—— 字符串截取

字符串传递到代码里常常会在开头或者末尾包含空格, 为确保你的字符串不传输额外的内容, 可使用 `ltrim` 或者 `rtrim` 函数 (如果你知道字符串开头或结尾有空格), 如果你确信字符串首尾都有空格, 可以使用 `trim` 函数。

37

这儿有个例子:

```
$string = "    The quick brown fox jumps over the lazy dog ";
var_dump(ltrim($string));
echo "<br/>";
var_dump(rtrim($string));
echo "<br/>";
var_dump(trim($string));
The output of this code is:
string(48) "The quick brown fox jumps over the lazy dog "
```

```
string(48) " The quick brown fox jumps over the lazy dog"
string(43) "The quick brown fox jumps over the lazy dog"
```



在这个例子中我们使用了 `var_dump` 函数来打印输出，因为它比使用 `echo` 或者 `print` 能得到更多的输出信息。

在这个字符串的两边有 5 个空格，因此，你可以看到前两次截取返回了同样的长度，即 48 个字符，然而在第一次截取后字符串中还存在末尾空格，第二次截取后字符串中还存在开头空格，当我们使用 `trim` 函数时，首尾空格都被移除，这个字符串仅有 43 个字符。

当你真的想在“大海捞针”时，你也能使用 `trim` 函数从“海”（指字符串）里移除指定的“针”（即字符列表）。如同下面的例子所示：

```
$string = "The quick brown fox jumps over the lazy dog" ;
var_dump(trim($string, "The dog"));
```

输出结果如下所示：

```
string(37) " quick brown fox jumps over the lazy "
```

`trim` 函数能移除指定的首尾字符，注意首尾空格仍然保留，当给出第二个参数时，功能上会有所变化，即你可以在 `trim` 函数中指定要移除的字符列表，这也适用于 `ltrim` 和 `rtrim` 函数，如果你想将空格也一并移除的话，则需要在指定移除字符中包含空格。

## 管理字符的大小写

38

### Character Case Management

接下来的一组字符串函数是用来改变已知字符串中部分字符的大小写。依然以前面的文本为例，我们可以用 `ucwords` 函数来让字符中每个单词的首字母变大写，代码如下所示：

```
$string = "The quick brown fox jumps over the lazy dog" ;
var_dump ucwords($string) ;
```

预期输出如下：

```
string(43) "The Quick Brown Fox Jumps Over The Lazy Dog"
```

你可以用 `strtoupper` 或 `strtolower` 函数来改变整个字符串的大小写，这两个函数会将全部字符变成大写或小写。请看下面的代码和输出结果：

```
$string = "The quick brown fox jumps over the lazy dog" ;
var_dump( strtoupper($string));
echo "<br/>" ;
var_dump( strtolower($string));
```

string(43) "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG"  
string(43) "the quick brown fox jumps over the lazy dog"

和之前提到的函数相比，下面要介绍的两个函数可能不是很常用到，但仍有其特别作用，尤其是在保存 Web 表单数据的时候。ucfirst 和 lcfirst 两个函数可以只改变字符串的首字母为大写或小写。如果想确保一个英文名称的首字母是大写，可以使用 ucfirst 函数。下面有些示例：

```
$string = "smith" ;  
var_dump( ucfirst ($string)) ;  
echo "<br/>" ;  
$string = "SMITH" ;  
var_dump( lcfirst($string)) ;
```

上述代码的输出结果如下所示：

```
string(5) "Smith"  
string(5) "sMITH"
```



lcfirst 函数只在 PHP 5.3 及之后的版本可用。

## 字符串查找

除了大小写以外，你肯定需要在字符串方面做更多的操作。所以在这里，我们看看其他一些改变字符串内容的方法，包括在很多函数原型里可以看到的那个著名的“针”（needle）。<sup>注1</sup>

首先来让我们讨论一下如何统计字符串的大小，在向限定长度的数据库域输入数据时，计算字符串长度的函数显得很重要。这套函数有两个：一个是 str\_word\_count，它用来计算指定字符串的单词数量，第二个是 strlen，它可以返回指定字符串的长度。注意，strlen 会把字符串里的空格也算作长度的一部分，所以你可能要先去除空格，然后再计算长度。看看下面的示例：

```
$string = " The quick brown fox jumps over the lazy dog" ;  
echo "word count: " . str_word_count($string) ;  
echo "<br/>" ;  
echo "String length: " . strlen($string) ;  
echo "<br/>" ;  
echo "String length trimmed: " . strlen(trim($string)) ;
```

输出结果如下所示：

注1：在 PHP 里，常用 needle 代表待查找的内容，用 haystack 代表待查找的范围。——译者注



Word count: 9  
String length: 45  
String length trimmed: 43

我们还能让 PHP 查询给定字符串中是否包含某主题文本中的特定部分，有两个函数可以干这件事：一个是 `strstr`，此函数区分大小写；第二个是 `stristr`，它查询时不区分大小写。但它们都能让你完全实现“大海捞针”。并且在找到特定部分的时候，返回包含“针”在内的后半部分。如果没找到，则返回 `false`。下面是示例代码：

```
$string = "The quick brown fox jumps over the lazy dog" ;
$needle = "BROWN fox";
echo "strstr: " ;
var_dump( strstr($string, $needle) );
echo "<br/>" ;
echo "stristr: " ;
var_dump(stristr($string, $needle) );
echo "<br/>" ;
$needle = "the" ;
echo "strstr: " ;
var_dump( strstr($string, $needle) );
echo "<br/>" ;
echo "stristr: " ;
var_dump(stristr($string, $needle) );
strstr: bool(false)
stristr: string(33) "brown fox jumps over the lazy dog"
strstr: string(12) "the lazy dog"
stristr: string(43) "The quick brown fox jumps over the lazy dog"
```

40

第一次尝试在指定的字符串里查找大写的“BROWN”时，返回为假。但当我们使用不区分大小写的 `stristr` 函数时，就得到了预期的结果。第二轮查找中，我们更换了“针”的内容为“the”后，结果也尽如人意；当用区分大小写的函数时，输出的是第一个小写“the”及之后的内容；当用不区分大小写的函数时，输出的是第一个“The”及后面的内容，即整个字符串。

接下来的一组函数是关于查询位置、修改内容和把“针”从“海”里取出来。你可以用 `strpos` 函数非常精确地在“大海”（即一段字符串）中定位这个“针”（即要查询的内容），如果指定的未找到字符串，`strpos` 函数会返回 `false`，这个不等同于返回数字 0，所以为确保准确无误，在判断返回值时需要用 `===`（三个等于号）。你可以用 `str_replace` 函数来替换文本的部分内容。最后，你可以用 `substr` 函数将查找到的指定字符中的一部分内容保存在其他变量里。这些函数最好是在一起使用，例如，你可能想要先用 `strpos` 函数找到指定字符串的位置，然后在同一行代码中，再用 `substr` 函数提取出一部分长度的内容保存到另一个变量里。考虑下面的示例和其后的输出结果：

```
$string = "The quick brown fox jumps over the lazy dog" ;
```



```

$position = strpos($string, "fox");
echo "position of 'fox' $position <br/>" ;
$result = substr($string, strpos($string, "fox"), 8);
echo "8 characters after finding the position of 'fox': $result <br/>" ;
$new_string = str_replace("the", "black", $string);
echo $new_string;

position of 'fox' 16
8 characters after finding the position of 'fox' : fox jump
The quick brown fox jumps over black lazy dog

```

## 字符串编辑

另一组很有用的函数中包括了能够修改 HTML 字符串的方法。strip\_tags 函数可以清除字符串中的嵌入式 HTML 标签，它还允许我们有条件地保留一部分标签被允许使用。看下面的示例：

```

$string = "The <strong>quick</strong> brown fox <a href='jumping.php'>jumps</a>
over the lazy dog" ;
echo $string . "<br/>" ;
echo strip_tags($string) . "<br/>" ;
echo strip_tags($string, '<strong>') ;

```

41 浏览器中的输出结果如下所示：

```

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog

```

如果这时候在浏览器中查看源文件，会看到以下内容：

```

The <strong>quick</strong> brown fox <a href='jumping.php'>jumps</a>
over the lazy dog<br/>

```

```

The quick brown fox jumps over the lazy dog<br/>

```

```

The <strong>quick</strong> brown fox jumps over the lazy dog

```

显然在第二行输出中所有的标签都被移除了，第三行输出中则只保留了 <strong> 标签。

addslashes 和 stripslashes 函数实现的功能正好相反，至于具体使用哪个函数则依赖于所应用的环境。“强大的转义符”节曾提到有些特殊的字符，例如双引号 (") 或者反斜线 (\) 在字符中表示时需要在前面再加个反斜线。addslashes 函数所做的工作就是在指定字符串中查找这些特殊字符并转义它们（也即适当添加反斜线），stripslashes

函数完成的工作则相反。下面我们举个例子来说明：

```
$web_path = "I'm Irish and my name is O'Mally" ;
echo addslashes($web_path) . "<br/>" ;
echo stripslashes($web_path) ;

I\'m Irish and my name is O\'Mally
I'm Irish and my name is O'Mally
```



如果你对一段先前已经存在反斜线的字符串使用 `addslashes` 函数，然后再次使用 `stripslashes` 函数，则所有的反斜线都会被移除。但这或许并不是你想要的结果。

你或许已经了解了一些有关 HTML 的知识，它严重依赖标记标签来显示内容，然而往往又想较方便地提供这种标签本身，我喜欢叫它原生格式。例如，用 HTML 来呈现左尖括号 (<) 可以用 `&lt;` 表示，右尖括号 (>) 用 `&gt;` 表示，& (连接符号) 用 `&amp;` 表示，诸如此类。我们可以用 `htmlentities` 函数将包含这些字符的内容转换到原生格式。基于安全原因，这通常用于 Web 系统接收来自外部的数据输入。如有这样的需求，我们还可以用 `html_entity_decode` 函数来反转数据得到正确结果。来看下面的示例：

```
$string = "The <strong>quick</strong> brown fox <a href='jumping.php'>jumps</a> over the lazy dog" ;
echo htmlentities($string) . "<br/>" ;
echo html_entity_decode($string) ;

The &lt;strong&gt;quick&lt;/strong&gt; brown fox &lt;a
href= 'jumping.php' &gt;jumps&lt;/a&gt; over the lazy dog<br/>
The <strong>quick</strong> brown fox <a href= 'jumping.php' >jumps</a> over
the lazy dog
```

42

在博客评论或网站留言簿等方面的应用中，这个方法会非常有用。例如，为了防止所显示的文字中可能包含任何有争议的 HTML 标记，可以将其转成原生格式显示以避免生效。

还有两个字符串函数值得关注，它们主要应用在 Web 开发的密码交互等安全方面。第一个是 `str_shuffle` 函数，它可以产生给定字符串的随机重组。你可以用它来生成一段字符串的随机排序（譬如产生一个难以破解的密码）。再有，你可以用 MD5 函数来完全打乱一个字符串。MD5 函数返回给定字符串的一个 32 位长的十六进制字符串结果。



对于固定的字符串，MD5 函数总是返回相同的结果，而 `str_shuffle` 函数每次都会随机重组字符串内容，所以为了更安全，你可以先对字符串做随机处理然后再执行 MD5 操作。

下面是应用这些函数的实践代码：

```
$string = "The quick brown fox jumps over the lazy dog" ;
echo str_shuffle($string) . "<br/>" ;
echo md5($string) . "<br/>" ;
echo md5(str_shuffle($string)) ;
```

第一次浏览器可能有如下输出：

```
dhuo p qr xnus hzeyveloftaiewbTojrg mock
9e107d9d372bb6826bd81d3542a419d6
f71d7b9a5880c06163ed8adbdee5b55e
```

刷新浏览器后可能变成如下输出：

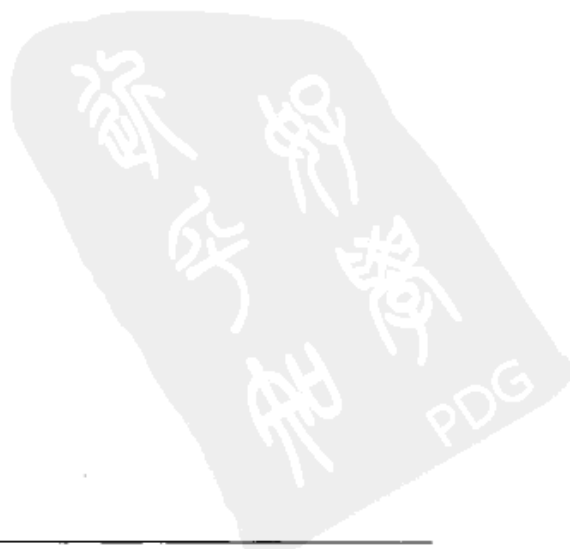
```
ugn uifertwckvxT thzrbh o mqo doeaoesjp y
9e107d9d372bb6826bd81d3542a419d6
1356809b12da9a25482891606ccfaa8f
```

43 第二行 MD5 的输出结果在每次刷新后都不会改变，其他输出内容在刷新后则会发生变化。



有关 MD5 函数的更多讨论（以及更安全的同类函数 sha1），请参见本书第 9 章。

PHP 提供了很多字符串类的函数，假以时日，你将会熟悉它们中的更多成员。我们这里提供的字符串函数或许能让你立刻获益。在下一章中，我们将延续这种风格讨论数组的问题。



说完字符串的概念，现在再让我们来看看强大而又极具灵活性的数组。数组被称为复合数据类型，意思是说，它比简单的字符串或整型以及其他已知的标量类型要复杂一些。你可以把数组想象成一个存放鸡蛋的纸盒（市场里能看到有很多槽的那种，我们一般叫蛋托）。但是呢，盒子可以分出很多更小的槽，而这些小小的槽可以被拆除并被合并成一个槽。此外，这些槽不仅能放鸡蛋，还能放面包、饼干、火腿肠什么的。当然，这个比喻其实并不十分恰当，因为真正的纸盒不能随意缩放，也不可能再在槽里装进去另一个纸盒。我之所以这么说，是想表达一个意思：数组真的很好很强大。

我们来更详细地谈谈数组。前面说了，它像蛋盒，有很多可以放数据的槽（元素）。这些元素和其中的数据总是一一对应（虽然你也可以让元素空着不放数据），这就是人们所说的键/值对。因此，如果我们有一个包含 5 个元素的数组，每个元素包含一个数字，具体范围是从 1 到 5，如表 5-1 所示（元素是从 0 开始计数的）。

表5-1 形象化表示数字（作索引）的数组

键	0	1	2	3	4
值	1	2	3	4	5

## 索引数组

用数字作为键名的数组一般叫索引数组<sup>注1</sup>。当然只要你愿意，键也可以用字符串表示，那样指的就是在本章稍后看到的关联数组。让我们考虑一个变量名为 `$myArray` 的数组。

注1：这时的键又叫下标。——译者注





数组变量的命名同样遵循 PHP 变量命名约定（参见“变量：数据类型、弱类型和作用域”部分）。

44

在 PHP 代码中，你可以使用方括号括起来的键名来访问数组中的元素。如果我们要取得数组中第三个元素的值——在这个例子里它是数字 3——并将它分配给一个要定义的变量，实现的代码如下所示：

```
// 切记，数组元素的键（即下标）从 0 开始算起
$singleValue = $myArray[2];
echo $singleValue ; // 会输出 3
```

现在，有两种方法可以实现在代码中定义一个数组。第一种方法就是使用方括号语法，具体代码如下所示：

```
$myArray[0] = 1;
$myArray[1] = 2;
$myArray[2] = 3;
$myArray[3] = 4;
$myArray[] = 5;
```

此方法假定我们已经知道数组的键和对应的值，但这是对键的硬编码。注意上面示例代码中最后一行并没有“强制”分配方括号内键的数值，这样做（就是不提供键的具体数值）的结果是将分配下一个可用的键索引数值给此元素。

用这种形式创建数组，或许并不符合你的初衷。另一种更常用的方法是将一组用逗号分隔的键 / 值对传递给 array 函数，如下所示：

```
$myArray = array(0 => 1, 1 => 2, 2 => 3, 3 => 4, 4 => 5);
```

这种形式更简单明了，但可能会稍稍不利于代码阅读。

如果你想创建一个空数组，只要写一个空参数的 array 函数即可：

```
$a = array();
```



任何数组的键必须唯一，否则会在由键取内容时造成混乱。PHP 不会禁止你为同一个键的内容反复赋值，但是每次赋值时都会覆盖同名键之前的内容。

## 关联数组

前面我们学习了索引数组，但就像我前面提到的，数组的键也可以用字符串来表示；同样的，每个键所对应的值不光可以是数字，也可以是字符串，如表 5-2 所示。

表5-2 形象化表示（以字符串为键名）关联数组

键	first	second	fname	initial	lname
值	1	2	Peter	B	MacIntyre

要创建表 5-2 中描述的数组，代码和上面的代码一样，只是把键的部分替换为字符串：

```
$myArray['first'] = 1 ;
$myArray['second'] = 2 ;
$myArray['fname'] = "Peter" ;
$myArray['initial'] = "B" ;
$myArray['lname'] = "MacIntyre" ;
```

同样的，你也可以用如下代码实现：

```
$myArray = array('first' => 1, 'second' => 2, 'fname' => "Peter", 'initial' =>
"B", 'lname' => "MacIntyre") ;
```

这里用到的单引号或双引号实际上是一样的，就像你看到的，它们可以互换。不过要注意的是，当你需要转义特定字符或者包含变量内容时，它们应当遵循字符串管理规则（参见“强大的转义符”）。

你可以用和索引数据一样的方式引用数组元素，同样是用字符串键名代替数字键名即可。因此，要显示数组中定义的全名内容，用下面的代码即可实现（为了更清楚，这里加了一些格式化操作）：

```
echo $myArray['fname'] . " " . $myArray['initial']
    . " " . $myArray['lname'] ;
```

## 多维数组

*1. 一维数组的进阶*

和普通变量一样，作为数组元素也可以保存多种数据类型：整数、字符串、日期、布尔值等。还有件了不起的事情是，它同样可以保存其他的数组，这样的数组就称为多维数组。数组的维数深度是没有限制的，但超过三级之后，要维护数组中键和值的对应关系会变得比较困难。下面用示例代码演示一个二维数组：

```
$myArray['first'] = 1;
$myArray['fruit'] = array("Apples", "Oranges", "Tomato");
$myArray['fname'] = "Peter";
$myArray['initial'] = "B";
$myArray['lname'] = "MacIntyre";

var_dump($myArray);
```



由于二维数组是数据库结果的主要结构化表示方法，所以你很有必要熟悉一下二维数组，更多信息请见本书第 7 章。

要引用数组中第二维（或更深层）的元素，只需要在后面添加方括号即可。例如，为了引用上面例子中包含字符串“Tomato”的元素，可以这样做：

```
echo $myArray['fruit'][2];
```

记住，数组的元素顺序计算是从 0 开始，所以，要获取第三个元素，只需要请求元素 2 即可。尽管我们将键和值都设置为字符串也是这样，西红柿也是一种水果（译注“Tomato”字符串是键名为“fruit”所指向数组的元素之一）。

## 数组可以动态构建

PHP 中的数组可以是动态的，这一点或许你已经实践过了。这意味着，使用恰当的指令或函数，不废吹灰之力就可以向一个已存在的数组添加元素。当然你也可以很容易地删除数组中的元素。实际上，你可以对数组做很多操作（本章稍后会一一验证这些内容），但现在，我们先来看看如何对数组做添加或移除元素的操作。

向一个已知数组的末尾添加元素，直接用一对空的方括号引用即可。



要是通过空方括号（也就是不提供键名）向一个关联数组的末尾添加元素，PHP 会将数组中下一个最大的数字索引作为新元素的键。这样就产生了混合数字和字符串键名的数组（这样也是允许的）。

我们来看下面的代码：

```
$myArray = array('first' => 1, 'second' => 2,
    'fname' => "Peter", 'initial' => "B", 'lname' => "MacIntyre");

echo $myArray['fname'] . " " . $myArray['initial']
    . " " . $myArray['lname'];
echo "<br/>";
$myArray[] = "555-5678";
var_dump($myArray);
```

这里，我们在数组的末尾添加了一个新值（一个电话号码），由于没有提供键名，当用 `var_dump` 函数打印这个数组时，显示最后添加的元素的键名为 0，因为在这个特别的关联数组里，它就是下一个可用的数字索引。如果你想避免这种行为的发生，就要给新元



素指定一个关联键名（例如：`$myArray["phone"] = "555-5678" ;`）：

```
Peter B MacIntyre
array(6){["first"]=>int(1)["second"]=>int(2)["fname"]=>string(5)"Peter"
["initial"]=>string(1)"B"["lname"]=>string(9)"MacIntyre"["0"]=>string(8)
"555-5678"
}
```

使用 `array_splice` 函数是删除数组中元素的方法之一，这是个支持多种参数选项组合的功能强大的函数，如果想更深入地了解如何使用这个函数，你可以在官网 [php.net](http://php.net) 上寻找帮助。现在，我们只是想要删除数组最后一个元素：即刚刚添加的那个电话号码。既然如此，我们可以用下面的代码来实现：

```
$myArray = array('first' => 1, 'second' => 2,
'fname' => "Peter", 'initial' => "B",
'lname' => "MacIntyre", 'phone' => "555-5678") ;

var_dump($myArray) ;
echo "<br/>" ;
array_splice($myArray, 4);
var_dump($myArray) ;
```

这时用 `var_dump` 函数显示数组的结果如下所示：

```
array(6) { ["first"]=> int(1) ["second"]=> int(2) ["fname"]=> string(5) "Peter"
["initial"]=> string(1) "B" ["lname"]=> string(9) "MacIntyre" ["phone"]=>
string(8)
"555-5678" }
array(5) { ["first"]=> int(1) ["second"]=> int(2) ["fname"]=> string(5) "Peter"
["initial"]=> string(1) "B" ["lname"]=> string(9) "MacIntyre" }
```

`array_splice` 函数的第一个参数指派了需要操作的数组，第二参数是指示从数组的第几位索引开始操作。在当前例子里，我们告诉 PHP 删除数组的第四个元素。注意这里我们用了索引位置 4，而不是键值 0<sup>注2</sup>。你可以根据条件添加第三个参数 `length`，即指示此次操作要删除多少个元素。由于我们没有提供这个参数，在默认情况下，删除操作会直至数组的末尾。如果你想维持原来的数组不变，并把 `array_splice` 函数的操作结果保存在新数组里，只需要直接将结果分配给一个变量即可，这样 `array_splice` 函数会建立一个全新的数组。我们还可以修改程序，将数组中涉及个人名字的内容全部移除并保存在一个新的数组里，代码如下所示：

```
$myArray = array('first' => 1, 'second' => 2,
'fname' => "Peter", 'initial' => "B",
'lname' => "MacIntyre", 'phone' => "555-5678") ;

$name_array = array_splice($myArray, 2, 3);
var_dump($myArray) ;
```

注2：数组的键的位置索引和键本身的值并不必然对应。——译者注



```
echo "<br/>" ;
var_dump($name_array) ;
```

输出结果如下所示：

50 `array(3) { ["first"]=> int(1) ["second"]=> int(2) ["phone"]=> string(8) "555-5678" }`  
`array(3) { ["fname"]=> string(5) "Peter" ["initial"]=> string(1) "B" ["lname"]=> string(9) "MacIntyre" }`

注意，这里 `array_splice` 函数保留了 `$myArray` 数组中最后一个元素：电话号码。而包含姓名的元素内容被有效提取出来，这是依赖于 `array_splice` 函数的第三个参数（`limit` 选项）实现的。另外一个和这个类似的用来维护数组的方法是使用 `unset` 函数，`unset` 实际上比 `array_splice` 更简单也更有效。如果我们想删除刚才那个数组中键名为 `initial` 的元素，可以用下面这行代码来实现：

```
unset($myArray['initial']);
```

## 遍历数组

在进入“数组函数精选”这部分内容之前，我们先来研究一下如何遍历一个数组。如果你看过前面第2章应该还有印象，我们跳过了一种流程控制结构并将其推迟到本章来讲述。这个流程控制结构指的就是对数组操作有很大价值的 `foreach` 结构，它允许任意代码单步访问所提供数组的每一个元素或每组键/值对。另外，这个结构在每次迭代时能将键和值分配给已定义的变量。以前面定义的数组为例，我们来实现一个循环并将每个键/值对显示出来，具体编码如下所示：

```
$myArray = array('first' => 1, 'second' => 2,
'fname' => "Peter", 'initial' => "B",
'lname' => "MacIntyre", 'phone' => "555-5678") ;

foreach ($myArray as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}
```

所产生的输出结果如下所示：

```
the Key is: first and its value is: 1
the Key is: second and its value is: 2
the Key is: fname and its value is: Peter
the Key is: initial and its value is: B
the Key is: lname and its value is: MacIntyre
the Key is: phone and its value is: 555-5678
```

如果你只对数组中的值感兴趣，仍然是用 `foreach` 结构，但放弃取键的部分就可以了：`foreach($myArray as $value)`，这样就会忽略掉元素中键的部分，只提供键所对应的元素的值。反过来说，如果你只需要数组的键文该怎么办呢，你可以写出以下 `foreach` 循环结构：

```
foreach(array_keys($a) as $key);
```

## 数组函数精选

数组有巨多的内置函数可供使用，这让它显得神通广大。这里，我会精选出最好的和最有效的函数并向你展示相关的示例代码。

### 数组排序

有许多方法可以组建或重组数组，通常使用的是排序函数。有些排序函数适合用在关联数组，有些适合用在索引数组，这一点我们会通过示例来演示其不同。`sort` 函数排序是基于值的比较，在排序完成时键的序列会重新整理；`rsort` 和 `sort` 所做的一样，只不过是完成倒序排列；`asort` 和 `arsort` 原理类似，也是比较值的大小，但会保留原始的键和值的对应关系；`ksort` 和 `krsort` 的排序过程是比较数组的键的大小（它们最适合用在关联数组的排序，因为索引数组的键通常是有序的）。下面的实例代码演示了上述这些排序函数。



我是摇滚乐队 Genesis 的超级粉丝，所以下面的示例代码是对于他们终于入选了摇滚名人堂表示致敬<sup>注3</sup>。

```
$ClassicGenesis = array("Tony Banks", "Phil Collins", "Mike Rutherford",  
"Steve Hackett", "Peter Gabriel" );  
  
sort ($ClassicGenesis) ;  
echo "<strong>Sorted on Values with re-generated Keys:</strong> <br/>";  
foreach ($ClassicGenesis as $key => $value) {  
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";  
}  
$ClassicGenesis = array("Tony Banks", "Phil Collins", "Mike Rutherford",  
"Steve Hackett", "Peter Gabriel" );  
rsort ($ClassicGenesis) ;  
echo "<strong>Now sorted in reverse order on Values with re-generated
```

注3：名人堂是指位于纽约市的一座国家纪念馆，其中陈列有许多美国名人的半身像。——译者注

52

```

Keys: </strong><br/>";
foreach ($ClassicGenesis as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}
$ClassicGenesis = array("Tony Banks", "Phil Collins", "Mike Rutherford",
    "Steve Hackett", "Peter Gabriel" );
asort ($ClassicGenesis);
echo "<strong>Now sorted in order of Values with Keys intact:</strong><br/>";
foreach ($ClassicGenesis as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}
$ClassicGenesis = array("Tony Banks", "Phil Collins", "Mike Rutherford",
    "Steve Hackett", "Peter Gabriel" );
arsort ($ClassicGenesis);
echo "<strong>Now sorted in reverse order of Values with Keys
    intact: </strong><br/>";
foreach ($ClassicGenesis as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}

$ClassicGenesis = array("Keyboards" => "Tony Banks", "Drums" => "Phil
Collins",
    "Bass Guitar" => "Mike Rutherford", "Lead Guitar" => "Steve Hackett",
    "Vocals" => "Peter Gabriel" );
ksort ($ClassicGenesis);
echo "<strong>Now sorted in order based on Keys: </strong><br/>";
foreach ($ClassicGenesis as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}

$ClassicGenesis = array("Keyboards" => "Tony Banks", "Drums" => "Phil
Collins",
    "Bass Guitar" => "Mike Rutherford", "Lead Guitar" => "Steve Hackett",
    "Vocals" => "Peter Gabriel" );
krsort ($ClassicGenesis);
echo "<strong>Now sorted in reverse order based on Keys: </strong><br/>";
foreach ($ClassicGenesis as $key => $value) {
    echo "the Key is: " . $key . " and its value is: " . $value . "<br/>";
}
    
```

之所以每次都重建数组，是由于排序函数会根据最新的键值对存在的数组进行重新排序，这也是为了不用重置数组和不使用已经被更改的数组。这些函数会在成功后返回真，失败时返回假。

上面的代码输出结果如下所示。

53

以值为基准正序排序并重新生成键值时的结果如下所示：

```

the Key is: 0 and its value is: Mike Rutherford
the Key is: 1 and its value is: Peter Gabriel
the Key is: 2 and its value is: Phil Collins
the Key is: 3 and its value is: Steve Hackett
    
```



the Key is: 4 and its value is: Tony Banks

以值为基准倒序排序并重新生成键值时的结果如下所示：

the Key is: 0 and its value is: Tony Banks  
 the Key is: 1 and its value is: Steve Hackett  
 the Key is: 2 and its value is: Phil Collins  
 the Key is: 3 and its value is: Peter Gabriel  
 the Key is: 4 and its value is: Mike Rutherford

以值为基准正序排序并保持原有键值时的结果如下所示：

the Key is: 2 and its value is: Mike Rutherford  
 the Key is: 4 and its value is: Peter Gabriel  
 the Key is: 1 and its value is: Phil Collins  
 the Key is: 3 and its value is: Steve Hackett  
 the Key is: 0 and its value is: Tony Banks

以值为基准倒序排序并保持原有键值时的结果如下所示：

the Key is: 0 and its value is: Tony Banks  
 the Key is: 3 and its value is: Steve Hackett  
 the Key is: 1 and its value is: Phil Collins  
 the Key is: 4 and its value is: Peter Gabriel  
 the Key is: 2 and its value is: Mike Rutherford

以键为基准进行排序的结果如下所示：

the Key is: Bass Guitar and its value is: Mike Rutherford  
 the Key is: Drums and its value is: Phil Collins  
 the Key is: Keyboards and its value is: Tony Banks  
 the Key is: Lead Guitar and its value is: Steve Hackett  
 the Key is: Vocals and its value is: Peter Gabriel

以键为基准进行倒序排序的结果如下所示：

the Key is: Vocals and its value is: Peter Gabriel  
 the Key is: Lead Guitar and its value is: Steve Hackett  
 the Key is: Keyboards and its value is: Tony Banks  
 the Key is: Drums and its value is: Phil Collins  
 the Key is: Bass Guitar and its value is: Mike Rutherford

PHP 还有一个功能强大的排序函数叫 `usort`。这个函数允许你用自己定义的特殊条件对数组进行排序。也就是定义一个比较函数用来重复对相邻两个元素进行比较。一定要看清楚，假如说你有一个数组需要排序，但前面提到的基础排序方法都不能满足要求，这才需要用到这个函数。

## 数学类函数

你可以对数组调用两个数学类函数，当然是针对那些索引数组及其中的数值。第一个函



数名叫 `array_sum`，它可以实现对数组中的元素数值进行相加然后返回总和。

另一个数学类函数叫做 `count`，顾名思义，它只是统计数组元素的个数并返回。来看看下面的示例代码，我们用这两个函数来计算一些考试成绩的平均值，并输出如下结果：

```
54 $grades = array(87,88,98,74,56,94,67) ;  
  
    $addedGrades = array_sum($grades);  
    echo "The sum of the provided grades is: $addedGrades <br/>";  
    $avgGrades = array_sum($grades) / count($grades) ;  
    echo "and the average of these grades is: " . round($avgGrades,2) ;  
  
    The sum of the provided grades is: 564  
    The average of these grades is: 80.57
```

代码中还用到了 `round` 函数，这是为了清理平均值和保留小数点后两位。

## 其他数组函数

现在让我们再看几个数组函数，这些是本质上都有独特功能的函数，所以很难把它们分到某一个类别里。我会分别描述每个函数及其工作原理，并给出相应的示例和输出结果。为了节约纸张（也节省木材），稍后我把这些示例代码汇总在一起集中展示。

如果你想确保一个数组中的所有元素都是唯一的，可以使用 `array_unique` 函数。这个函数会识别并删除数组中任何重复的元素，如果确实存在重复的值会对数组进行收缩。操作完成后元素的键不会被重新编号。

如果你想检测一个值是否存在于已知的数组中，可以用 `in_array` 函数。此函数会在数组中查找指定的数据段，并返回真或假。如果你想返回数组中的键或部分元素，而不仅仅是布尔类型的真或假，则可以使用 `array_search` 函数。

如果你想打乱一个数组中的所有值，就用 `shuffle` 函数。它会将指定数组的元素重新随机排序，并分配新的索引键值（即放弃或删除原始的键）。

如果你想返回指定数组中的全部或部分随机键，可以使用 `array_rand` 函数。如果只想返回一个随机键（默认情况），这个函数将会返回符合键的数据类型的一个变量（不论索引数组或关联数组）。如果你要请求更多的随机值（指定第二个参数即可），返回结果将会是个新数组。

下面是以上所谈到的有关函数的示例代码，以及相应的输出结果：

```
55 $grades = array(87,88,98,74,56,94,67,98,49) ;
```

```

var_dump($grades);
echo "<br/>" ;
$uniqueGrades = array_unique($grades);
var_dump($uniqueGrades);
echo "<br/>" ;

if (in_array(49, $grades) ) {
    echo "there is a 49 in here and it is at element: " . array_search(49,
$grades) ;
} else {
    echo "no 49s here" ;
}
echo "<br/>" ;
shuffle($grades) ;
var_dump($grades);
echo "<br/>" ;

$random = array_rand($grades);
echo "the random key from grades is: " . $random ;

array(9) { [0]=> int(87) [1]=> int(88) [2]=> int(98) [3]=> int(74) [4]=>
int(56) [5]=> int(94) [6]=> int(67) [7]=> int(98) [8]=> int(49) } array(8) {
[0]=> int(87) [1]=> int(88) [2]=> int(98) [3]=> int(74) [4]=> int(56) [5]=>
int(94) [6]=> int(67) [8]=> int(49) }
there is a 49 in here and it is at element: 8 array(9) { [0]=> int(87) [1]=>
int(56) [2]=> int(98) [3]=> int(67) [4]=> int(98) [5]=> int(74) [6]=> int(88)
[7]=> int(49) [8]=> int(94) }
the random key from grades is: 4
    
```

如果想要将一个数组中的元素按照各自的对应关系转换成变量，用键名作为变量名，用值作为变量的值，你可以使用 `extract` 函数。它最好用在关联数组上，主要原因是 `$0` 或 `$1` 不是有效的 PHP 变量名。



需要小心的是，当你提取数组元素作为变量时，键的名称有可能会因同名而覆盖代码中的其他变量——这也会在此函数应用于非信任输入时造成安全困扰，因为它有可能导致输入的数据覆盖已有变量。

如果你想实现相反的功能（即转换一系列变量到数组里），就用 `compact` 函数吧。调用 `compact` 函数时仅仅需要知道变量的名字即可，无须在前面加上 `$` 符号。下面是几行示例代码：

56

```

$Drums = "John Mayhew" ;
$LeadGuitar = "Anthony Phillips" ;

$Genesis = array("Keyboards" => "Tony Banks", "Drums" => "Phil Collins",
"BassGuitar" => "Mike Rutherford", "LeadGuitar" => "Steve Hackett",
"Vocals" => "Peter Gabriel" )

extract ($Genesis, EXTR_SKIP);
echo "Original Genesis Lineup: $Keyboards, $Drums, $BassGuitar, $LeadGuitar,
$Vocals" ;
    
```

```
extract ($Genesis);
echo "<br/>Classic Genesis Lineup: $Keyboards, $Drums, $BassGuitar,
$LeadGuitar, $Vocals" ;

$newGenesis = compact(Keyboards, Drums, BassGuitar, LeadGuitar, Vocals);
echo "<br/>";
var_dump($newGenesis);
```

注意例子中第一个 `extract` 设置了可选参数 `EXTR_SKIP`。这指示函数在执行过程中跳过同名的变量。输出结果如下所示，可以看到 `$Drums` 和 `$LeadGuitar` 保持不变：

```
Original Genesis Lineup: Tony Banks, John Mayhew, Mike Rutherford, Anthony
Phillips, Peter Gabriel
Classic Genesis Lineup: Tony Banks, Phil Collins, Mike Rutherford, Steve
Hackett, Peter Gabriel
array(5) { ["Keyboards"]=> string(10) "Tony Banks" ["Drums"]=> string(12)
"Phil Collins" ["BassGuitar"]=> string(15) "Mike Rutherford" ["LeadGuitar"]=>
string(13) "Steve Hackett" ["Vocals"]=> string(13) "Peter Gabriel" }
```



如有兴趣具体了解 `extract` 有哪几种不同的指定选项，可以到官网一探究竟，地址是 <http://www.php.net/extract>。

如果你想把两个或更多的数组放在一起组成一个更大的数组，可以使用 `array_merge` 函数。顾名思义，它会把两个或多个数组进行简单的连接，并且你不用担心会有重复数据的问题。



若是在合并关联数组时发现同名的键值，合并操作中最后的那个键会覆盖前面的键。如果合并以简单数字为下标的索引数组时发生键的冲突，则索引会重新计算。

下面是一些示例和输出结果：

```
$test1grades = array(1 => 87, 2 => 88,98,74,56,94,67,98,49) ;
$test2grades = array(1 => 67, 2 => 76,78,98,56,93,68,95,83) ;
```

```
$allgrades = array_merge($test1grades, $test2grades);
var_dump($allgrades);
echo "<br/><br/>";
```

```
$Genesis1 = array("Keyboards" => "Tony Banks", "Drums" => "Phil Collins",
"BassGuitar" =>"Mike Rutherford","LeadGuitar" => "Steve Hackett",
"Vocals" =>"Peter Gabriel" ) ;
```

```
$Genesis2 = array("Keyboards" => "Tony Banks",
"Concert Drums" => "Chester Thompson",
"BassGuitar" =>"Mike Rutherford","LeadGuitar" => "Mike Rutherford",
"ConcertLeadGuitar" => "Daryl Sturmer", "Vocals" =>"Phil Collins" ) ;
```

```
$allGenesis = array_merge($Genesis1, $Genesis2);
```



var\_dump(\$allGenesis);

```
array(18) { [0]=> int(87) [1]=> int(88) [2]=> int(98) [3]=> int(74) [4]=>
int(56) [5]=>
int(94) [6]=> int(67) [7]=> int(98) [8]=> int(49) [9]=> int(67) [10]=>
int(76) [11]=>
int(78) [12]=> int(98) [13]=> int(56) [14]=> int(93) [15]=> int(68) [16]=>
int(95) [17]=>
int(83) }
```

```
array(7) { ["Keyboards"]=> string(10) "Tony Banks" ["Drums"]=> string(12) "Phil
Collins" ["BassGuitar"]=> string(15) "Mike Rutherford" ["LeadGuitar"]=>
string(15)
"Mike Rutherford" ["Vocals"]=> string(12) "Phil Collins" ["Concert Drums"]=>
string(16) "Chester Thompson" ["ConcertLeadGuitar"]=> string(13) "Daryl
Sturmer" }
```

我们来看看本节最后一个数组相关函数 `array_walk`，此函数通过一个已定义函数来实现双重职责，即循环数组的每个元素并将元素逐个传递给已定义函数，你可以认为这就是个更简单的 `foreach` 数组循环。在下面的示例代码中，我们会调用一个函数，将数组中的每个成绩加上 10 分，然后在浏览器中显示出来。这个函数的第一个参数会接收一个传递进来的数组的值并作为可选项，第二个参数可传递数组的键。

```
function add10($value ) {
    $value += 10 ;
    echo $value . "&nbsp;" ;
}

$testgrades = array(87,88,98,74,56,94,67,98,49) ;
var_dump($testgrades);
echo "<br/><br/>";
array_walk($testgrades, 'add10');

array(9) { [0]=> int(87) [1]=> int(88) [2]=> int(98) [3]=> int(74) [4]=>
int(56) [5]=> int(94) [6]=> int(67) [7]=> int(98) [8]=> int(49) }
```

97 98 108 84 66 104 77 108 59

还有更多的数组函数无法在本章中一一呈现，所以，要了解它们，一定得去官网 [php.net](http://php.net) 看看。别担心它们对你是否有用，大胆去尝试吧！





在这一章里，我们来看看面向对象编程（OOP），这也是重点体现 PHP 精华的部分。面向对象编程已经存在了很多年，已然是业界非常成熟的程序设计方法。然而，它在 Web 编程领域只是在最近 5 ~ 7 年才得以发展。这是件好事，因为它让 Web 程序构建更加健壮。

类、对象、多态、封装、继承、方法还有属性——这些都是面向对象的时髦词儿，别被其吓倒，这些词通通都可以扔掉。我们先来看看这些术语的定义，然后通过实践一系列类的示例再看看它们如何协同工作。

## 类

类是用来激活一个对象的一段定义或代码模板。你能阅读和修改这段代码，但当它被执行的时候会作为一个对象存在。在实际代码中，类每次使用时都是产生一个实体的副本。

## 对象

一个对象就是类被复制并激活的形态。你或许听过“实例”这个术语，这个听起来很牛的词，其实指的就是一个对象在内存中具有唯一命名的副本。在 PHP 中，将类实例化成一个对象，需要用到 `new` 这个关键字。

## 方法

方法实际就是写在类定义段中的函数。在 PHP 中，方法指的就是用 `function` 来实现的一个写在类中的函数定义。

## 属性

属性就是指类定义段里的变量。

这又是一个牛气冲天的词，实际的意思是指在面向对象编程中允许两个独立的类有共同的接口名称，如方法和属性，而不用通过其他方式实现，所以，就算两个类中都有叫 `print` 的方法，PHP 也不会对此形成困扰，主要是因为对象名称是唯一的。

## 60 封装

封装是 PHP 经常会用到的另一个面向对象编程的特性。封装的唯一目的是为了管理类的内部属性而允许外界通过使用类的特定方法去影响一个类内部的受保护数据(即属性)。更详细的解释请参见“公开的、保护的和私有的”部分内容。

## 继承

面向对象编程允许类从已知的基类(也叫父类)复制或继承。如果你有一个类叫“`person`”，还定义了一些方法和属性，你想创建另外一个类，属于“`person`”，但是叫“`Lumberjack`”，你可以使用“`person`”类的所有属性，比如姓名、身高、眼睛的颜色等，这些属性又可以扩展到“`Lumberjack`”，从而节省了时间，提高了效率。

# 付诸实践

现在，为有助于理解前面说的大多数概念，我们来看一些代码。首先，让我来设定几个即将用到的基础类及其使用方法。我尽量让这部分内容不至于太过复杂，希望能帮你掌握面向对象编程的基础并认识用这种编程方法能完成多少工作。



如果你已经熟悉了其他语言的面向对象编程，那么你可以径自跳过这一节。在如何实现面向对象编程方法上，各种语言之间都有些细微差别，所以如果遇到稍有出入的情况，请一定仔细弄个明白。

首先，让我们看一个能够在 Web 页面渲染出数据录入表单的类。这个界面很简单，就是让用户提交姓名和评论(有点像留言簿的那种表单)。其中一个类用来定义基本的 HTML 标签生成，另一个类用来处理表单中的元素。另外还用到了一个能生成 HTML 表格的类，因为这是 HTML 中比较特别的一套标签。我们不管其他的 HTML 标签，只要在示例页面里够用就行了。当然了，你还可以通过这部分展现的代码来开发其他你想用到的任何 HTML 标签。

这次代码有点长，所以，请花点儿时间阅读它并深入理解本章前面罗列的知识点。

下面是构建 HTML 的类：

```

class html {

    // 首页定义类的属性（变量）
    private $tag ;

    // 接下来定义方法（类中的函数）

function __construct ($title="") {
    // 类的构造函数，每次实例化时都会被调用
    $this->tag = "<HTML> " ;
    $this->tag .= "<HEAD>" ;
    $this->tag .= "<title> $title </title>" ;
    $this->tag .= "</HEAD><BODY>" ;
    echo $this->tag ;
    return ;
}

function page_end () {
    // 结束 HTML 页面，关闭 body 和 html 标签
    $this->tag = "</BODY></HTML>" ;
    return $this->tag ;
}

function RawText($textString, $textColor="black", $bgcolor='', $fontSize="",
    $fontWeight="normal") {
    // 此方法用来将未格式化的文字内容发送到浏览器
    $this->tag = "<span style='color: $textColor ; background-color: $bgcolor ;
    font-size: $fontSize ; font-weight: $fontWeight'>" ;
    $this->tag .= "$textString";
    $this->tag .= "</span>" ;
    return $this->tag ;
}

function Image($source, $title="", $height="", $width="", $align="center",
    $border=0, $valign="middle",
    $class="", $id="", $name="", $onType1="", $onAction1="",
    $onType2="", $onAction2="", $onType3="", $onAction3="") {
    // 此方法用来在 HTML 页面中添加图片标签，它有三个
    // on 开头的事件 (onclick、onblur、onmouseup)
    $this->tag = 'tag .= 'name="' . $name . '" ' ;
    if ($height == "") $height = "16" ;
    if ($width == "") $width = "16" ;
    $this->tag .= 'height="' . $height . '" width="' . $width . '" ' ;
    $this->tag .= 'border="' . $border . '" ' ;
    if ($class) $this->tag .= 'class="' . $class . '" ' ;
    if ($id) $this->tag .= 'id="' . $id . '" ' ;
    if ($title) $this->tag .= 'title="' . $title . '" alt="' .
        $title . '" ' ;
    if ($align) $this->tag .= 'align="' . $align . '" ' ;
    if ($valign) $this->tag .= 'valign="' . $valign . '" ' ;
}

```



```

        if ($onType1)      $this->tag .= $onType1 . '=' . $onAction1 . ' ' ;
        if ($onType2)      $this->tag .= $onType2 . '=' . $onAction2 . ' ' ;
        if ($onType3)      $this->tag .= $onType3 . '=' . $onAction3 . ' ' ;
        $this->tag .= ">" ;
        return $this->tag ;
    }

    function Spacer($spaces = 1) {
        $this->tag = "";
        for ($i=1 ; $i <= $spaces ; $i++) {
            $this->tag .= "&nbsp;" ;
        }
        return $this->tag;
    }

    function NewLine($number = 1) {
        $this->tag = '';
        for ($i=1 ; $i <= $number ; $i++) {
            $this->tag .= "<br/>" ;
        }
        return $this->tag;
    }
} //html 类结束

```

接下来是表格生成类：

```

class table {

    private $tag ;

    function Begin($border=0, $align="center", $width='100%', $cellpadding=2,
        $cellspacing=2, $class='', $id='', $bgcolor='', $style='') {
        $this->tag = '<table ' ;
        if ($align)          $this->tag .= 'align="' . $align . ' ' ;
        if ($width)          $this->tag .= 'width="' . $width . ' ' ;
        if ($border > 0)     $this->tag .= 'border="' . $border . ' ' ;
        if ($cellpadding > 0) $this->tag .= 'cellpadding="' .
            $cellpadding . ' ' ;
        if ($cellspacing > 0) $this->tag .= 'cellspacing="' . $cellspacing . ' ' ;
        if ($class)          $this->tag .= 'class="' . $class . ' ' ;
        if ($id)             $this->tag .= 'id="' . $id . ' ' ;
        if ($bgcolor)        $this->tag .= 'bgcolor="' . $bgcolor . ' ' ;
        if ($style)          $this->tag .= 'style="' . $style . ' ' ;
        $this->tag .= ">" ;
        return $this->tag ;
    }

    function Header($text) {
        $this->tag = '<th>' . $text . '</th>' ;
        return $this->tag ;
    }
}

```

```

    }

function RowOn($align="", $bgcolor="", $class="", $height="") {
    $this->tag = '<tr ' ;
    if ($align)      $this->tag .= 'align="' . $align . '" ' ;
    if ($bgcolor)    $this->tag .= 'bgcolor="' . $bgcolor . '" ' ;
    if ($class)      $this->tag .= 'class="' . $class . '" ' ;
    if ($height)     $this->tag .= 'height="' . $height . '" ' ;
    $this->tag .= ">" ;
    return $this->tag ;
}

function ColumnOn($colspan=1, $align='left', $width="", $rowspan="",
    $bgcolor="", $class="", $valign="", $height="") {
    $this->tag = '<td ' ;
    if ($align)      $this->tag .= 'align="' . $align . '" ' ;
    if ($colspan)    $this->tag .= 'colspan="' . $colspan . '" ' ;
    if ($width)      $this->tag .= 'width="' . $width . '" ' ;
    if ($rowspan)    $this->tag .= 'rowspan="' . $rowspan . '" ' ;
    if ($bgcolor)    $this->tag .= 'bgcolor="' . $bgcolor . '" ' ;
    if ($class)      $this->tag .= 'class="' . $class . '" ' ;
    if ($height)     $this->tag .= 'height="' . $height . '" ' ;
    if ($valign)     $this->tag .= "valign='" . $valign . "'>" ;
    if (!$valign)    $this->tag .= "valign='middle'>" ;
    return $this->tag ;
}

function ColumnOff() {
    $this->tag = '</td>' ;
    return $this->tag ;
}

function RowOff() {
    $this->tag = '</tr>' ;
    return $this->tag ;
}

function End() {
    $this->tag = '</table>' ;
    return $this->tag ;
}
} // 表格类结束

```

最后是表单生成类：

```

class form {

    private $tag ;

    function Begin($action, $method='post', $name='', $id='', $style='',
    $class='') {
        $this->tag = '<form ' ;
    }
}

```

```

if ($method)      $this->tag .= 'method="' . $method . '" ' ;
if ($action)      $this->tag .= 'action="' . $action . '" ' ;
if ($name)        $this->tag .= 'name="' . $name . '" ' ;
if ($id)          $this->tag .= 'id="' . $id . '" ' ;
if ($style)       $this->tag .= 'style="' . $style . '" ' ;
if ($class)       $this->tag .= 'class="' . $class . '" ' ;
$this->tag .= "><input type='hidden' name='posted' value='1'>" ;
return $this->tag ;
}

function HiddenValue($name, $value="") {
    $this->tag = '<input type="' . 'hidden' . '" ' ;
    if ($name)      $this->tag .= 'name="' . $name . '" ' ;
    if ($value)     $this->tag .= 'value="' . $value . '" ' ;
    $this->tag .= ">" ;
    return $this->tag ;
}

function InputLabel($textLabel, $labelFor, $required=false, $class='') {
    if ($required == true) $required = "<font color='red'>*</font>";
    $this->tag = '<label for="' . $labelFor . '" class="' . $class . '">' ;
    $this->tag .= $textLabel . $required;
    $this->tag .= " :&nbsp;</label>" ;
    return $this->tag ;
}

function Input($InputType, $EntityName, $value="", $align="center", $size="",
    $id="", $align="center", $readonly="", $class="",
    $onType1="", $onAction1="", $onType2="", $onAction2="",
    $onType3="", $onAction3="") {

    $this->tag = '<input type="' . $InputType . '" name="' . $EntityName
        . '" size="' . $size . '" ' ;
    if ($align)      $this->tag .= 'align="' . $align . '" ' ;
    if ($id)         $this->tag .= 'id="' . $id . '" ' ;
    if ($value == "on"){
        $this->tag .= ' checked ' ;
    } elseif ($value){
        $this->tag .= 'value="' . $value . '" ' ;
    }
    if ($class)      $this->tag .= 'class="' . $class . '" ' ;
    if ($onType1)     $this->tag .= $onType1 . '=' . $onAction1 . '" ' ;
    if ($onType2)     $this->tag .= $onType2 . '=' . $onAction2 . '" ' ;
    if ($onType3)     $this->tag .= $onType3 . '=' . $onAction3 . '" ' ;
    if ($readonly)    $this->tag .= 'readonly ' ;
    $this->tag .= ">" ;
    return $this->tag ;
}

function Textarea($name, $cols, $rows, $value="", $align="left", $class="",
    $readonly="", $onType1="", $onAction1="", $onType2="", $onAction2="",

```

```

        $onType3="", $onAction3="") {
        $this->tag = '<textarea name="' . $name . '" cols="'
        . $cols . '" rows="' . $rows . '" ' ;
        if ($align)      $this->tag .= 'align="' . $align . '" ' ;
        if ($class)      $this->tag .= 'class="' . $class . '" ' ;
        if ($onType1)    $this->tag .= $onType1 . '=' . $onAction1 . '" ' ;
        if ($onType2)    $this->tag .= $onType2 . '=' . $onAction2 . '" ' ;
        if ($onType3)    $this->tag .= $onType3 . '=' . $onAction3 . '" ' ;
        if ($readonly)   $this->tag .= 'readonly ' ;
        $this->tag .= ">$value</textarea>" ;
        return $this->tag ;
    }

    function form_end(){
        return '</form>' ;
    }
} // 表单类结束

```

65

我把这三个文件分别命名为 *html\_calss.inc*、*table\_class.inc* 和 *form\_class.inc*，之所以要分开保存的一个重要原因是为了可以将其分别单独加载到别的程序中。这样你就可以在需要用到它们的时候在代码中使用 `include`（或者使用 `require` 更好。这三个文件仅仅是对象（类）的定义，对于这样一个很小的需求而言，这里的代码看起来有些多（俗话说：杀鸡用上了牛刀）。

在 PHP 中，要创建（实例化）一个激活的类（即对象），需要用到 `new` 关键字。下面是使用前面三个类文件并实例化对象的示例：

```

// 引用页面需要的类文件
require_once ('classes/html_class.inc');
require_once ('classes/table_class.inc');
require_once ('classes/form_class.inc');

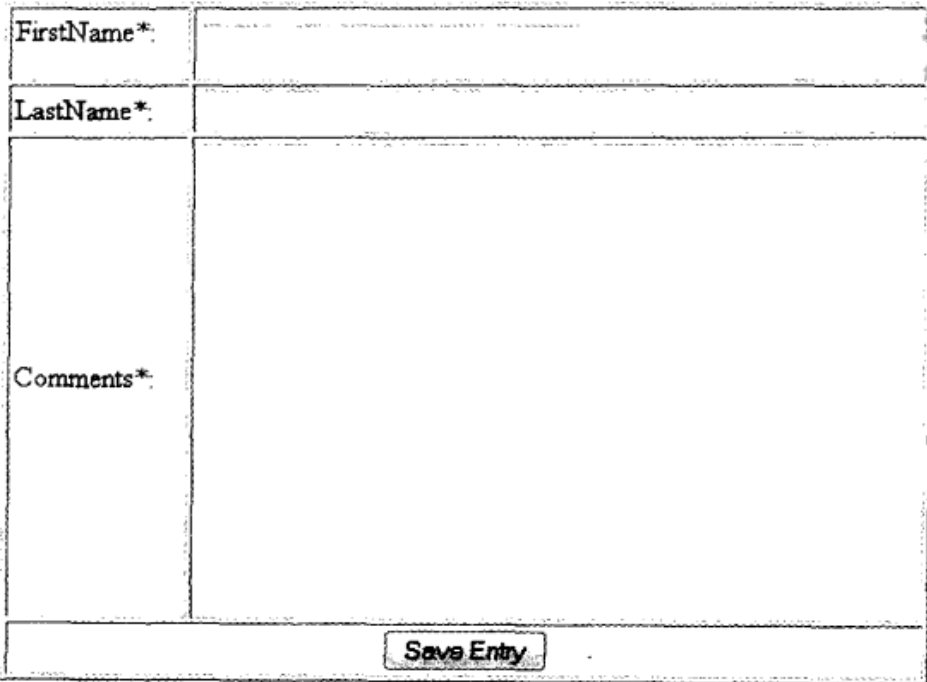
// 将准备用到的类实例化
$HTMLPage      = new html("GuestBook Page") ;
$MyTable       = new table() ;
$MyForm        = new form() ;

```

接下来，我们开始用类中定义的方法来构建 Web 输入页面，其中姓氏显示长度为 30 个字符，名字长度为 40 个字符，还有个 8 行高 40 列宽的评论。说明一下，仅仅是为了重点讲解面向对象编程，所以界面和结构就相对简陋一些。但我希望，这能让你管中窥豹并举一反三，逐步掌握编写面向对象程序库的方法。

那么，让我们来看看这个页面的大概样式。如图 6-1 所示的界面即是我们稍后要写的示例代码运行结果。





FirstName*:	<input type="text"/>
LastName*:	<input type="text"/>
Comments*:	<input type="text"/>
<input type="button" value="Save Entry"/>	

图6-1 用面向对象代码生成的简易表单

## 魔术方法

你或许注意到了，HTML 类定义了一个叫 `__construct` 的函数。这个方法特殊之处在于每次类被实例化时此方法都会被触发执行。我们可以据此添加一些在 HTML 类对象于内存中创建时都会执行的操作。另外，我们还把页面会用到的标题作为方法的参数传递进去。看起来 `__construct` 方法只要写上了，就会在每次实例化时触发，而没写就不触发。其实如果没写，也会调用，只不过没有任何效果罢了。

这种会自动执行的方法（像刚才那个 `__construct`）是一种预先定义的方法，PHP 中叫做魔术方法。所有的魔术方法都是随类的定义而与生俱来的，即使你可能没有实际写出这种方法的定义语句，但 PHP 把定义作为默认的行为，所以在解析的时候仍然会当做它可能存在而去试图在需要的时候调用它们。另一个魔术方法是常常作为清除对象使用的析构方法（`__destruct`），它会在对象被从内存中移除或脚本结束时调用。如果你有些特殊的操作需要在对象被消灭时执行，那就在代码中类定义区域里建立这个方法吧。



你或许还注意到在 HTML 类里那个 `__construct` 方法中有输出页面标签（像 `<html><head>` 之类）的语句。这里只是魔术方法的一个使用示例，并且这个方法也没有返回值，作为类的构造函数的魔术方法不允许有返回值。

想了解 PHP 中其他的魔术方法，请到 PHP 官网查阅，网址为 <http://www.php.net/manual/en/language.oop5.magic.php>。



## 变量 \$this

5.1.2.1

在前面类的代码中，你会看到一个叫 `$this` 的变量。`$this` 是定义对象中内部保留的变量。它不需要提前定义，当类被实例化为一个对象时，它便存在了。它通过“`$this->变量名`”的方式来引用内部可见的类的相关属性。事实上，你只能通过“`$this->变量名`”这一种形式来引用一个类的内部属性。

## 对象实战

67

5.1.2.2

关于生成余下页面的代码，简要地展开如下：

```
// 开始一个有边框且左对齐，宽度为 30%的表格
$webpage = $MyTable->Begin(1, "left", "500") ;
$webpage .= $MyTable->RowOn();
$webpage .= $MyTable->ColumnOn();
$webpage .= $MyForm->Begin() ; // "proof" of polymorphism
$webpage .= $MyForm->InputLabel("FirstName","fname", true);
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->ColumnOn(1,"left");
$webpage .= $MyForm->Input("text", "fname", "", "", 30);
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->RowOff();

$webpage .= $MyTable->RowOn();
$webpage .= $MyTable->ColumnOn();
$webpage .= $MyForm->InputLabel("LastName","lname", true);
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->ColumnOn();
$webpage .= $MyForm->Input("text", "lname", "", "", 40);
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->RowOff();

$webpage .= $MyTable->RowOn();
$webpage .= $MyTable->ColumnOn();
$webpage .= $MyForm->InputLabel("Comments","comments", true);
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->ColumnOn();
$webpage .= $MyForm->Textarea("comments", 40, 15);
$webpage .= $MyTable->ColumnOff();
```

```
$webpage .= $MyTable->RowOff();

$webpage .= $MyTable->RowOn();
$webpage .= $MyTable->ColumnOn(2, "center");
$webpage .= $MyForm->Input("submit", "submit", "Save Entry");
$webpage .= $MyTable->ColumnOff();
$webpage .= $MyTable->RowOff();

$webpage .= $MyForm->form_end();
$webpage .= $MyTable->End();

$webpage .= $HTMLPage->page_end() ;

echo $webpage ;
```

你能看到，代码中调用了生成页面相关类的那三个方法，并把所有的返回值存储在一个叫 \$webpage 的变量中。每个页面都有部分类的函数（方法）没用到，例如 html 那个类，只有构造函数和 page\_end 用到了，这很正常，你不可能每次做点儿零活就把工具箱里的所有型号的螺丝刀都用一遍。

注意许多方法在调用的时候都没有提供参数，这也是 PHP 中用户定义函数方面挺叫人受用的一种特性。你可以在方法定义的时候设置默认值，如果没有提供参数，则使用那个特定的默认参数。请看下面这行代码：

```
$webpage .= $MyForm->Begin("save_entry.php") ;
```

这儿的调用只提供了一个参数，但类的原型如下：

```
function Begin($action, $method='post', $name='', $id='', $style='',
    $class='')
```

除了第一个参数 (\$action)，其他的全都设有默认值。有些参数的默认值是空，因此通常不用提供。实际上，只要你愿意，甚至在调用的时候也不用写上可选参数。在这里，你必须提供 \$action 参数，因为它没有默认值。把必需的参数放在参数列表的第一个位置始终是个好习惯。事实上，如果必需的参数不是放在参数列表靠前的位置（如果有的话），PHP 不会工作得太顺畅。如果你觉得参数列表太长了，那考虑用一个或两个数组代为传递吧。当查阅 HTML 页面中生成的 <form> 标签时，它看起来像这样：<form method="post" action="save\_entry.php" >，所以无论如何，我们总是需要在 <form> 标签中添加 action 属性和 POST 提交方法。

## 公开的、保护的和私有的

正像作用域在面向过程中存在一样，在 PHP 面向对象编程方面也有作用域的概念。你可



以标识类的属性和方法是公开的、保护的或私有的三者之一。类定义的公开的实体可以在所有作用域中使用和访问；保护的实体仅允许被类自己和父类或继承它的类访问，而私有的实体只能被类自己访问。

封装，是一个保护类内部实体免受外部影响的概念，当然私有作用域（`private`）属性最能达成这个目的（本章稍后我们会看到一个 `person` 类的实践）。基本上，设置限制访问作用域是免受外部改变最好的做法，尤其是你在编写将被很多不同项目所使用的类库，或者是制作商业可用版本类库的时候。要保护你的代码免遭任何方式的破坏，不管是不正确的用法还是直接从外部访问类内部的属性。保护的（`protected`）作用域较少用到，除非你想从一系列层级的对象树（即有继承关系的类）中获益。如果没有定义作用域标识，则默认是公开的（`public`），需要外部访问的接口方法，最好都设定为公开的。一个类最好的向外部共享信息的办法是将属性设置为私有，并设置公开 `get` 和 `set` 方法（`get` 和 `set` 又叫访问器）。我们将在下一小节中看到具体示例。

69

## get和set访问器

在本章中面向对象编程模式的最后一部分内容里，我们来看看针对类属性的 `get` 和 `set` 方法。这个概念允许在类本身范围内实现更多被保护的接口。每个类的属性都有各自对应的 `get` 和 `set` 方法，并且只能通过这些访问器方法来影响每个属性。下面要演示的 `person` 类有这些属性：`firstname`、`lastname` 和 `gender`，这三个属性都有各自的 `get` 和 `set` 方法。

```
class person {  
    private $firstname ;  
    private $lastname ;  
    private $gender ;  
  
    public function getFirstname() {  
        return $this->firstname;  
    }  
  
    public function getLastname() {  
        return $this->lastname;  
    }  
  
    public function getGender() {  
        return $this->gender;  
    }  
  
    public function setFirstname($firstname) {
```



```

        $this->firstname = $firstname;
    }

    public function setLastname($lastname) {
        $this->lastname = $lastname;
    }

    public function setGender($gender) {
        $this->gender = $gender;
    }
} // person 类 结束

```

70



你可能想在 set 方法里添加验证代码，这会允许你在数据通过类实际保存之前先做些更精确的处理。例如，在前面的 setGender 方法里，你可以在认可输入值前先验证输入的数据是否是期望的男性或女性（分别用 M/F 表示），如果不是就拒绝。

注意这里没有实现构造函数，这完全没有任何问题，PHP 会去找 \_\_construct 方法，如果找到就运行它，如果没有找到，则什么事儿也不会发生，除非是要在内存中创建类的实例。调用已经存在的类并且使用访问器，可以这样做：

```

$newPerson = new person();
$newPerson->setFirstname("Peter");
$newPerson->setLastname("MacIntyre");
$newPerson->setGender("male");

echo "the Person class currently has these values: " ;
echo "<br/> First Name:" . $newPerson->getFirstname();
echo "<br/> Last Name: " . $newPerson->getLastname();
echo "<br/> Gender: " . $newPerson->getGender();

```

上面的代码会产生如下输出：

```

the Person class currently has these values:
First Name:Peter
Last Name: MacIntyre
Gender: male

```

你可以看到这里没有直接访问类中那三个属性的代码。比如，我们不能这样写：

```
echo $newPerson->lastname ;
```

还有很多 PHP 面向对象方面的内容，这里都没有涉及，像继承、接口、对象克隆、延迟静态绑定等。面向对象编程是强大而朴素的。本章中，我的目的只是为了 Web 编程入门而做些简单的示范，并给出一些实际应用中的简短示例。要想完全和通盘掌握 PHP 的面向对象开发，我推荐一本书叫《Object Oriented PHP 面向对象（No Starch Press）》，作者是 Peter Lavin。

# 数据库交互

Database Interaction

你有没有觉得前面所接触到的示例都是静态页面，如果不手动修改每个 HTML 或 PHP 文件，页面内容不会发生变化。通常的网站或 Web 应用都需要存取动态信息等相关内容。在本章中，我们来看看如何让页面中显示一些从数据库中读取的内容。

## MySQLi 对象接口

MySQLi Object Oriented Interface

MySQL 是和 PHP 搭配使用的最流行的数据库平台。如果你看过 MySQL 网站，能找到几个不同版本的 MySQL 产品。我们着重来看在开源社区自由发布的版本。PHP 还有几个不同接口的数据库操作工具，但我们主要使用能体现面向对象方式的 MySQL 增强扩展（MySQLi）。如果你阅读过前一章的 PHP 面向对象部分，应该对这个接口的实现方式不会感到太生疏。

首先我们来建一个非常基础的数据结构，我提示一下，上一章中有个简易留言本示例，就从扩充它开始吧。为了能实际保存相关内容，我们向数据库中添加一张表，将这张表命名为 *guests*，结构如下：

```
table: guests
guestid int(11)
fname   varchar(30)
lname   varchar(40)
comments text
```

建立表的 SQL 语句如下所示：

```
CREATE DATABASE 'website' ;
USE 'website' ;

CREATE TABLE 'guests' (
```



```
'guestid' INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
'fname' VARCHAR( 30 ) NOT NULL ,
'lname' VARCHAR( 40 ) NOT NULL ,
'comments' TEXT NOT NULL
)
```

PHP 标准配置中即内建了支持面向对象接口，只要在 PHP 配置环境中激活 MySQL 扩展即可。要启用它的类实例，你所必须做的就是写出下面这样的代码：

```
$mydb = new mysqli('localhost', 'dbuser', 'dbpassword', 'dbname');
```

在本例中，有个名叫 `website` 的数据库，另外我们假设数据库的用户名是 `petermac`，密码是 `1q2w3e4r`。实际代码就会是这样：

```
$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');
```

这就给了我们一个用 PHP 来访问数据库引擎的入口，稍后会访问特定表及其他数据。一旦这个类实例化并将对象赋给变量 `$mydb`，就可以通过它的相关方法开始我们的数据库工程。



本章假定你对 SQL 查询语言有一定了解，这里我们不再花时间去复习它。有很多线上资源和书籍能帮助你精通 SQL 规则。

我们来写例子中其他需要用到的代码，以便将来宾信息保存到表中。我们必须修改这行代码：`$webpage .= $MyForm->Begin()`；将参数改成在表单提交时可以保存信息的目标地址。这个目标我们命名为 `save_data.php`，所以现在这行代码是这个样子：

```
$webpage .= $MyForm->Begin('save_data.php') ;
```

此文件会从 `$_POST` 数组中取数据并保存到数据库中，完整的代码如下所示：

```
$mydb->close();

$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');

$sql = "INSERT INTO guests(fname, lname, comments)
VALUES ('$_POST[fname]', '$_POST[lname]', '$_POST[comments]')";

if ($mydb->query($sql) == TRUE) {
    echo " 来宾数据保存成功 .";
} else {
    echo " 插入数据失败，请稍后重试或寻求技术支持 " ;
}

$mydb->close();
```





基于简单和清晰的演示目的，我们先不关心来自用户输入的内容（\$\_POST 数组）是否安全。在第 9 章中，专门讲了跨站脚本和 SQL 注入的问题，在任何公共站点上用到 SQL 时，请一定注意这部分内容。

73

首先，我们将 MySQLi 类实例化成一个对象，并把对象变量命名为 \$mydb。接下来，我们构建一条 SQL 查询字符串并保存到变量 \$sql 中。然后调用类的 query 方法，并且同时测试返回值，如果确定成功（即结果为真）就把相应的结果显示到屏幕上。最后，我们调用类的 close 方法来清理和消除内存中的对象资源。

## 取得数据并显示

在网站的其他地方，你可能想提取并在页面上列出访客信息以及他们评论内容的片断。要达到这个目的，我们同样可以用 MySQLi 类的相关成员，并编写一句 SELECT SQL 语句查询出结果。这样的信息要在浏览器中显示出来有多种方案，我们看一个示例就可以了。注意，和前面的 \$mydb 实例相比，返回数据所用的对象是另一个不同的实例。PHP 负责将返回的数据填充到结果对象中。来看代码：

```
$mydb = new mysqli('localhost', 'petermac', '1q2w3e4r', 'website');

$sql = "SELECT * FROM Guests ORDER BY lname, fname";

$result = $mydb->query($sql);

while( $row = $result->fetch_assoc() ){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论说： " . substr($row['comments'],0,150) ;
    echo "<br/>";
}

$result->close();

$mydb->close ();

$result->close();

$mydb->close ();
```

这里我们还是调用 query 方法并把返回的信息保存到变量 \$result 中。然后，我们循环调用这个结果对象的 fetch\_assoc 方法来一次提取一行数据，并且每次都把单行数据保存到变量 \$row 中。只要取出的一行数据里有内容，则 while 循环就会继续运行下去。在循环期间把 \$row 里相关内容直接显示到浏览器窗口中。最后，要关闭结果和数据库对象。





我发现 MySQLi 最有用的方法之一是 `multi_query`，此方法允许你一次传递并运行多行 SQL 语句。如果你想要在执行 INSERT 之后接着执行 UPDATE 之类的语句，可以用它来一次完成。

说到这儿，当然，这些仅仅是 MySQLi 类所能提供的一些皮毛而已。在 <http://www.php.net/mysqli> 中能找到关于这个类的说明文档，你会发现 MySQLi 有丰富的方法可供使用，每种结果类都有相对应的主题文档。

## PHP数据对象（PDO）

接下来我们看看 PHP 数据对象（PDO），这是 PHP 里提供的另一个数据库接口。和 MySQLi 的主要区别是 MySQLi 类只能限制在 MySQL 数据库引擎上使用，但 PDO 拥有许多数据库平台的额外驱动，你可以利用 PDO 的一致性方法来访问它支持的所有数据库引擎，只要确定你所写的 SQL 语法能被那个数据库引擎认可就行。这里列出了一些可用的数据库平台：

- 微软 SQL Server 和 Sybase
- Firebird/Interbase
- IBM
- Informix
- MySQL
- Oracle
- ODBC and DB2
- PostgreSQL
- SQLite
- Driver 4D for PDO

为了使用 PDO，要修改 `php.ini` 文件来使相应驱动生效，无论如何你都要自己动手来达到这个目的。以 Windows 平台为例，开启 MySQL 的 PDO 驱动要这样修改：

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

如果你想切换到另一个数据库平台，比如 Informix，只要禁用 MySQL 那行扩展并插入一行 Informix 的扩展即可<sup>注1</sup>：

```
extension=php_pdo.dll
```

注1：禁用那行不是必需的。——译者注

//extension=php\_pdo\_mysql.dll  
extension=php\_pdo\_informix.dll

你应该已经知道了所有要使用到的新数据库引擎的标识名称。当然了，只要你喜欢，也可以同时使用多行 PDO 引擎。那样的话，原来那行指令保持不变，再添加一行就可以了。

如果要用 PDO 接口重复实现一次本章前面的两个示例（分别是存和取），开头那段代码大致如下所示：

```
$dsn = 'mysql:dbname=website;host=localhost';  
$myPDO = new PDO($dsn, 'petermac', '1q2w3e4r');  
  
$sql = "INSERT INTO guests (fname, lname, comments)  
VALUES ('$_POST[fname]', '$_POST[lname]', '$_POST[comments]')";  
  
$result = $myPDO->query($sql);  
  
if ( $result !== False ) {  
    echo " 来宾数据保存成功 .";  
} else {  
    echo " 插入数据失败，请稍后重试或寻求技术支持 " ;  
}
```

很明显，这段代码和我们的第一个示例极其相似，只不过多了一行 `$dsn` 的变量定义，它清楚地表示出 PDO 对象要怎么连到数据库引擎。实际上，要是打算更换数据库引擎的话，你必须修改这一行代码（要是把前面的两行保存在一个 `include` 文件里，那样在你的整个 Web 项目中只要修改一行代码就可以了）。

这里同样的，我们也把查询结果放在变量里，尽管这个 `insert` 语句没有返回数据集，但我们仍然可以校验这个返回的结果，只要不是为假（`false`），就证明代码运行正确。

第二段使用 PDO 格式的代码大致如下：

```
$dsn = 'mysql:dbname=website;host=localhost';  
$myPDO = new PDO($dsn, 'petermac', '1q2w3e4r');  
  
$sql = "SELECT * FROM Guests ORDER BY lname, fname";  
  
$result = $myPDO->query($sql);  
  
while ($row = $result->fetch(PDO::FETCH_ASSOC)){  
    echo $row['fname'] . " " . $row['lname'] ;  
    echo " 评论说： " . substr($row['comments'],0,150) ;  
    echo "<br/>";  
}
```

## PDO 预处理对象

76

PDO 可以构造一个预处理语句 (MySQLi 勉强也可以做到)。你可以通过此方式建立需要重复执行且只有细微差别的 SQL 语句。这能让你从大量整段 SQL 语句的海洋中脱身, 并且还能带来切实可靠的更多安全性。构造预处理语句的时候, 你可以用名称或问号来标记占位符, 我们把这两种情况都做一下演示。使用名称作为占位符的可读性更好一些。来看一下对前面 SELECT 示例中 ORDER BY 子句所做的修改:

```
$dsn = 'mysql:dbname=website;host=localhost';
$myPDO = new PDO($dsn, 'petermac', '1q2w3e4r');

$stmt = $myPDO->prepare('SELECT * FROM Guests ORDER BY ? ');
$stmt->execute(array('lname'));

echo "以名字排序评论列表 <br/>";
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论道: " . substr($row['comments'],0,150) ;
    echo "<br/>";
}

$stmt->execute(array('fname'));

echo "以姓氏排序评论列表 <br/>";
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论道: " . substr($row['comments'],0,150) ;
    echo "<br/>";
}
```

这里我们重复利用了 SQL 语句, 只是修改了如何排序。而这一切所必须要做的就是不断调用 execute 方法并传递一个不同的参数, 就能让差不多一样仅有微小区别的 SQL 运行。这个例子使用问号作为占位符, 参数值通过解析位置替换。如果有多个问号占位符, 你要确保数组的传递数量和顺序都完全一致, 这样 execute 方法才能正确执行。下面是使用名称作为占位符的示例, 注意在实际应用中可能比这个要复杂:

```
$stmt = $myPDO->prepare('SELECT * FROM Guests ORDER BY :ordervalue ');
$stmt->execute(array('ordervalue' => 'lname'));

echo "以姓氏排序评论列表 <br/>";
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论道: " . substr($row['comments'],0,150) ;
    echo "<br/>";
}
```



```
$statement->execute(array('ordervalue' => 'fname'));

echo " 以姓氏排序评论列表 <br/>";
while ($row = $statement->fetch(PDO::FETCH_ASSOC)){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论道： " . substr($row['comments'],0,150) ;
    echo "<br/>";
}
}
```

77

上述 SQL 语句里，用名称参数代替了问题标记，并且名称以冒号 (:) 开头作为参数标识，用来实现预处理语句中的变量替换。另外，需要明确的是，提供的参数数组必须是包含相同名称键值的关联数组。预处理语句能够让你免于堆砌重复 SQL 语句从而有效节省时间，而且你会发现它在 INSERT 和 UPDATE 语句中比在 SELECT 查询中要更加有用。



MySQLi 和 PDO 都能支持基于数据库事务的行为，这在你需要保证数条语句在任何情况下都能最终成功执行上有非常大的价值（例如银行交易事务）。只要确认你的 Web 应用程序中是否要用到这样的功能和特性即可。

要是你担心外部来源的数据输入，比如从表单输入的数据，一定要考虑用一下 PDO 的 quote 方法，此方法会把输入数据中所有引号和特殊字符都进行转义或转换（请结合基础安全最佳实践，详见第 9 章）。

PDO，类似于 MySQLi，但所提供的远不止上面这些，当然实际使用时你也会深入研究它，尤其是它不会限制你开发的应用程序固定在某个数据库平台上。限于篇幅，这里就不展开讨论了，否则类似的内容又要占去一个章节。但是你一定要知道它的可用性。

## 低成本数据管理方案

本章到目前为止，我们已经学习了用数据库引擎实现数据管理。你还可以考虑另外两种入门级方案：使用轻量级的数据库 SQLite 和通过文件来管理数据。这两者用得较少，一般用在一些特别场景像移动技术方面（例如安卓系统）。坦率地讲，人们使用它们往往是由于一些应急方案和需求。但我们还是来看一下吧。

### SQLite

SQLite 数据库扩展在 PHP 默认安装即可用，它和其他大多数数据库扩展功能类似。值得注意的是，它的所有数据都基于单个文件存储，因此它不需要独立的数据库引擎即可

78



工作。你要是想创建一个只依赖 PHP，但不依赖于其他产品的软件，这一点会非常有用。因为 SQLite 是 PHP 内建支持的，你所要做的就是直接引用它。



如果你在用 PHP 5.3，可能需要修改 php.ini 文件来包含指令：`extension=php_sqlite.dll`，从这个版本开始，默认指令是 `extension=php_sqlite3.dll`，和前面的不是同一个扩展。

SQLite 也有面向对象式的接口，所以你可以用下面的代码来实例化对象：

```
$database = new SQLiteDatabase('c:/copy/website.sqlite');
```

这个语句的巧妙之处是如果未找到指定的文件，SQLite 会帮你建立。下面继续我们的来宾留言数据库示例，用 SQLite 建表的语句如下所示：

```
$sql = 'CREATE TABLE guests (  
    guestid INTEGER PRIMARY KEY ,  
    fname TEXT ,  
    lname TEXT ,  
    comments TEXT )';  
  
$database->queryExec($sql);
```



SQLite 不像 MySQL 那样有自增选项，它是把任何整数字段类型并且同时还是主键的作为自增列。你可以在执行 `INSERT` 语句的时候用指定的值覆盖这样的列。

注意它的数据类型和我们知道的 MySQL 还不完全一样。记住，SQLite 是很小巧的数据库工具，所以，它的数据类型也相对简单。表 7-1 列出了 SQLite 用到的数据类型。

表7-1 SQLite支持的数据类型

数据类型	解释
文本	可以存储 NULL、TEXT 或 BLOB 类型的内容，如果传入的是数字，在存储前会被转换成文本
数字	可以存储整数或实数，文本在存储前也会被转换
整数	和数字类型相似，但如果是实数，会先转换成整数，这会影响数据精度
实数	也和数字类似，但会强制将整数作为浮点数存储
空	是个比较杂的类型，并不限制是哪种方式存储，提供什么就存什么，很精确

79 运行下面的代码将数据存入数据库文件：

```
$sql =  
    'INSERT INTO guests (fname, lname, comments) ' .  
    'VALUES ("Peter", "MacIntyre", "Nice work pilgrim!"); ' .
```

```
'INSERT INTO guests (fname, lname, comments) ' .
'VALUES ("Dawn", "Riley", "Great site, love what you have
done with the place!"); ' .

'INSERT INTO guests (fname, lname, comments) ' .
'VALUES ("Peter", "MacIntyre", "Me again... just saying hello."); ' ;

$database->queryExec($sql) ;
```

请注意，这里我们同时执行了多条 SQL 语句。其实用 MySQLi 也能完成，你只要记得是用 multi\_query 方法就行了。SQLite 所支持的方法叫 queryExec。然后，同样是加载数据，下面的代码也会有同样的输出：

```
$sql = "SELECT * FROM guests ORDER BY lname, fname";

$result = $database->query($sql);

while ($row = $result->fetch()){
    echo $row['fname'] . " " . $row['lname'] ;
    echo " 评论道： " . substr($row['comments'],0,150) ;
    echo "<br/>";
}
```

前面的代码输出结果如下所示：

```
Peter MacIntyre 评论道： Me again... just saying hello.
Peter MacIntyre 评论道： Nice work pilgrim!
Dawn Riley 评论道： Great site, love what you have
done with the place!
```

SQLite 能力不算太差，几乎可以算是“较强”的数据库引擎，虽然名字中有个“lite”（意为轻的），但不表示它真就功能薄弱，相反的，它只是在系统资源占用方面比较轻量罢了。长期来讲，它非常适合于便携设备或对资源要求较敏感的环境。



如果你正开始准备动态网站开发，访问 SQLite 时，建议直接用 PDO 接口。这样的话，能够轻装上阵，等将来准备好的时候，再逐步迁移到更强健的数据库，比如 MySQL，也会比较容易。

## 用文件替代数据库

PHP 有非常多的工具套件，其中有许多较为隐蔽的特性。举其中一个例子（通常不怎么被人注意到的），它有操作各种文件的神奇能力——当然，大家都知道 PHP 能打开文件，但究竟怎么去实现呢？比如开发一个网上调查，从那些缺少资金的潜在客户的需求里，

80

了解接近真实范围的可能性。当然了，我原本想让客户通过 PHP 和 MySQLi 数据库交互来实现这个任务，但紧接着，那个当地 ISP 的月租费吓了我一跳，尽管如此，客户询问是否还有其他方式可以实现这个需求。最终结果是，如果不想用 SQLite，那就只能换另一个方案，只用文件来管理和操作少量文本文件，这样也可供日后检索。在即将讨论的函数里，要单独拿出来说的话，其实没有什么与众不同的，实际上，它们都是些基础工具，或许大多数人都很熟悉了，见表 7-2。

表7-2 通用PHP文件管理函数

函数名	功能描述
<code>mkdir()</code>	用来在服务器上建立目录
<code>file_exists()</code>	检测指定的文件或目录是否存在
<code>fopen()</code>	打开一个文件以实现读或写（准确用法请参见参数选项）
<code>fread()</code>	读取文件内容并保存到 PHP 变量中
<code>flock()</code>	为所写的文件添加一个排他锁
<code>fwrite()</code>	将变量的内容写入文件
<code>filesize()</code>	当读取文件时，用此方法检测总共读了多少字节
<code>fclose()</code>	用来在文件使用完成后关闭文件

有趣的事情是，要完成上述目标，需要将所有这些函数捆绑在一起配合实现。例如，让我们建立一个网上调查，这其中涉及了两个问题页面。用户能输入回答一些选项，并在稍后返回日期信息，这样就完成了调查，而我们则保留了他的选择信息。通过我们这个小应用程序的逻辑范围，希望你能看到，它有扩展成完全生产环境可用的基本前提。

首先，我们要允许用户在任何时候能够返回调查页面，并提供额外的回复。为此，必须得有一个唯一的标识，以区别用户的身份来源。通常个人邮箱地址是唯一的（虽说有时候你也知道别人的邮箱，但不见得你会知道他的个人安全信息或能够控制它）。为了简便起见，我们假定每个邮箱是真实无虚的，并且不关心密码之类的东西。所以，一旦得到用户的邮箱地址，我们需要在不同于其他来宾的位置保存这些信息。为实现这个效果，我们将在服务器上为每个访问者建立一个目录（当然，假定你有服务器上相应位置的适当访问权限，且允许读写文件）。因为我们要以访问者的邮箱地址作为相对的唯一标识，简单地用这个标识建立相应的目录。一旦目录建立（同时检测用户是否来自上一个会话），就会读取已经存在的任何文件，并将内容显示在表单控件 `<textarea>` 里，这样访问者上一次提交的信息就一目了然了。然后，我们再把把他接下来用表单提交的调查问卷保存起来。下面是第一页的代码（`<?PHP` 标签也包含在里面，这样可以很好地区分 PHP 代码和 HTML 内容）：

```
<?php
session_start();
```



```

if ($_POST['posted'] && $_POST['email'] != "") {
    $folder = "surveys/" . strtolower($_POST['email']);

    // 路径信息保存到 session 里
    $_SESSION['folder'] = $folder;

    if (!file_exists($folder)) {
        // 建立目录, 添加空文件
        mkdir($folder);
    }

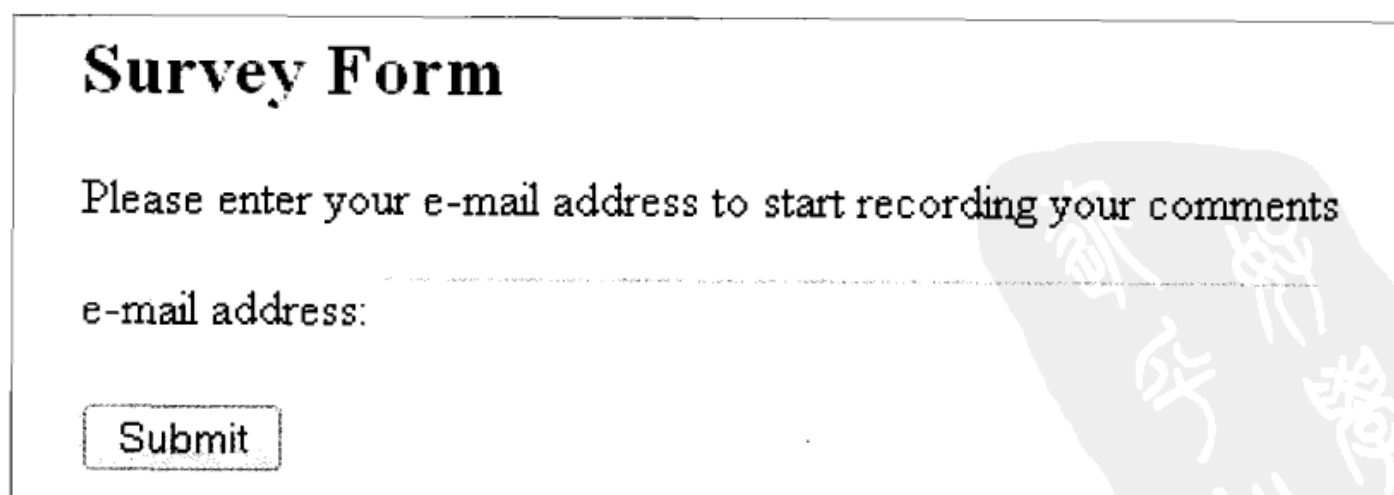
    header( "Location: page1.php" );
} else { ?>
    <html>
    <head>
    <title>Files & folders - On-line Survey</title>
    </head>
    <body bgcolor="#FFFFFF" text="#000000">
    <h2>Survey Form </h2>
    <p>Please enter your e-mail address to start recording your comments</p>
    <form action="<?=$PHP_SELF ?>" method=POST>
    <input type="hidden" name="posted" value=1>
    <p>e-mail address: <input type="text" name="email" size="45" >
    <br/><br/>
    <input type="submit" name="submit" value="Submit">
    </form>

    <?php } ?>

    </body>
    </html>

```

访客提交邮箱地址的页面如图 7-1 所示。



**Survey Form**

Please enter your e-mail address to start recording your comments

e-mail address:

图7-1 调查入口页



你一定看明白了，首先，启动一个新的 session，保存访问者信息，这样就可以将其传递给下一页，然后，我们检测在后面代码中的表单是否真的被提交了，也就是说检查邮箱地址字段中是否被输入了相应的内容。如果检测失败，表单会再次显示。当然了，在真正产品化的版本中，类似这样的功能肯定会输出一些错误信息，并告诉用户需要输入有效的内容。

一旦检测到符合要求的情况（假定表单正确提交了），我们就把包含邮箱地址的内容保存在变量 `$folder` 中，用它作为目录名，来保存调查信息。同样的，我们会通过 session 在之后的页面中保存这个新生成的内容。这里只是简单地取得邮箱并准备使用它（再次强调，在一个相对安全的站点，为保护数据，需要再配以适当的安保措施）。

接下来，我们要了解，目录已经存在的情况。如果不存在，我们就用 `mkdir()` 函数建立目录，传给此函数一个路径作为目录名，系统就会尝试建立它。



在 Linux 环境下，`mkdir()` 函数有额外的参数可以控制新建目录的访问级别和权限，所以，要根据你的环境来确定相应的参数。

确认目录存在之后，我们只需要重定向浏览器到调查首页面即可。

打开调查首页（见图 7-2），即可看到供用户填写的表单。

Please enter your response to the following survey question:

What is your opinion on the state of the world economy?  
Can you help us fix it ?

Submit

图7-2 调查首页

动态生成表单所需的代码如下所示：

```
<?php
session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question1.txt" ;

$file_handle = fopen($filename, "a+");
// 打开需要读取的文件
// 读取文件中存在的任何内容
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // close this handle

if ($_POST['posted']) {
    // 第一次需要建立文件，并将 $_POST['question1'] 中的文字内容保存起来
    $question1 = $_POST['question1'];
    $file_handle = fopen($filename, "w+");
    // 完全改写打开的文件

    if (flock($file_handle, LOCK_EX)) {
        // 设置一个排他锁
        if (fwrite($file_handle, $question1) == FALSE) {
            echo "不能写入文件 ($filename)";
        }
        flock($file_handle, LOCK_UN);
        // 释放锁
    }

    // 关闭文件句柄并重定向到下一页？
    fclose($file_handle);
    header( "Location: page2.php" );
} else {
?>
<html>
<head>
<title>Files & folders - On-line Survey</title>
</head>
<body>

<table border=0><tr><td>
Please enter your response to the following survey question:
</td></tr>
<tr bgcolor=lightblue><td>
What is your opinion on the state of the world economy?<br/>
Can you help us fix it ?
</td></tr>
<tr><td>
<form action="<?=$PHP_SELF ?>" method=POST>
<input type="hidden" name="posted" value=1>
```

```
<br/>
<textarea name="question1" rows=12 cols=35><?= $comments ?></textarea>
</td></tr>
<tr><td>
<input type="submit" name="submit" value="Submit">
</form></td></tr>
</table>
<?php } ?>
```

有几行代码，我想在这儿强调一下，因为这是在管理和操作真实的文件空间。先取出 session 中的目录名信息，和文件名一起合并为一个 `$filename` 字符串变量，这算做好了操作文件的准备。请记住，这个过程的重点是，任何已经存在的文件内容都会显示出来，也允许用户输入或修改之前保存的信息，所以，代码一开始有这样的指令：

```
$file_handle = fopen($filename, "a+");
```

然后使用 `fopen()` 函数打开文件，PHP 会返回一个指向文件的句柄，将其保存为变量 `$file_handle`。注意这个函数还有第二个参数：`a+` 选项，在 PHP 的官方网站上，你能找到这个参数的完整选项及相关解释，这里用此选项是为了用读和写的方式打开，并将指针指向文件末尾，如果文件不存在，PHP 会尝试建立。接下来有两行代码，读取整个文件（使用 `fread()` 函数检测文件大小）并保存到变量 `$comments` 中，然后，将文件关闭。

```
$comments = fread($file_handle, filesize($filename));
fclose($file_handle);
```

接下来，我们看看表单那部分程序的执行结果，无论如何，文本框中的输入内容都应保存起来。这回，我们还是打开同一份文件，但第二个参数换成 `w+` 选项，这是为了用只写的方法打开文件——如果文件不存在就新建，若存在就清空，操作指针位于文件开头。实质上就是，用新输入的文本替换原先文件中的内容。为此，我们需要借用 `fwrite()` 函数：

```
// 设置一个排他锁
if (flock($file_handle, LOCK_EX)) {
    if (fwrite($file_handle, $question1) == FALSE) {
        echo "不能写入文件 ($filename)";
    }
    // 释放锁
    flock($file_handle, LOCK_UN);
}
```

必须保证信息真正存进了指定的文件，所以，我们要将文件写操作限制在一个条件语句中，以确保一切顺利进行。首先，尝试获得一个针对文件的排他锁（用 `flock()` 函数）——这是为了确定在操作过程中没有其他的进程会访问到这个文件，在写操作完成后，我们释放这个锁。

如你所见，写文件函数使用变量 `$file_handle` 来将 变量 `$question1` 的内容写入文件。完成之后，直接关闭文件，并跳转到调查的下一页，效果如图 7-3 所示。

图7-3 调查第二页

请看下面的调查第二页代码，用来操作另一个文件（名为 `question2.txt`），除了本身的名  
称不同，代码完全一样。

```
<?php
session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question2.txt" ;

$file_handle = fopen($filename, "a+");
// 打开需要读取的文件
// 读取文件中存在的任何内容
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // close this handle

if ($_POST['posted']) {
    // 第一次时建立文件，并将_POST['question1'] 保存到文件里
    $question2 = $_POST['question2'];
    $file_handle = fopen($filename, "w+");
    // 完全改写打开的文件

    if (flock($file_handle, LOCK_EX)) { // 设置排他锁
        if (fwrite($file_handle, $question1) == FALSE) {
            echo "Cannot write to file ($filename)";
        }
    }
}
```



```

    }
    flock($file_handle, LOCK_UN); // 释放锁
}

// 关闭文件并重定向到下一页
fclose($file_handle);
header( "Location: last_page.php" );
} else {

?>
    <html>
    <head>
    <title>Files & folders - On-line Survey</title>
    </head>
    <body>

    <table border=0><tr><td>
    Please enter your comments to the following survey statement:
    </td></tr>
    <tr bgcolor=lightblue><td>
    It's a funny thing freedom. I mean how can any of us <br/>
    be really free when we still have personal possessions.
    How do you respond to the previous statement?
    </td></tr>
    <tr><td>
    <form action="<?= $PHP_SELF ?>" method=POST>
    <input type="hidden" name="posted" value=1>
    <br/>
    <textarea name="question2" rows=12 cols=35><?= $comments ?></textarea>
    </td></tr>
    <tr><td>
    <input type="submit" name="submit" value="Submit">
    </form></td></tr>
    </table>

<?php } ?>

```

只要你愿意，可以继续添加类似的文档处理，因此，你的调查也可以根据你的个人喜好进行处理。你可以在一个页面中显示多个问题，并直接将每个反馈保存到各自的文件中。

当然，若之后的每页有多达五个问题，你会发现有大量的单个文件需要管理。幸运的是，PHP 有其他的文件处理函数。例如，file() 函数就是 fread() 函数的备选方案，它能将整个文件读入到一个数组里，一行算一个元素。如果你的资料内容格式恰当——每行都限制在一行的结束符 \n 里——你就能很容易地将多个部分的资料存储在一个文件里。自然而然的，这个规则既可以用 HTML 表单来循环生成控件，也可以用于表单数据的录入。

当需要处理文件时，在 PHP 网站上，仍然可以找到很多选择。如果你打开手册中的“文件系统”那一节，能发现不少于 70 种函数——当然也包括这里所讨论的部分。例如，你

会分别找到能检测文件是否可读和可写的函数：`is_readable()` 和 `is_writable()`，也能找到文件权限相关、释放磁盘空间或统计磁盘空间和删除文件、复制文件等方面的函数。当你能完全掌握它们时，只要有足够的时间和意愿，你甚至能实现整个网站应用，而  
87

当有一天（而且极有可能）你遇到一个客户，而他不想在数据库上花费一分钱，你将会为他们准备一个尽可能相似的替代方案。



# PHP周边

*PHP and Friends*

PHP 是一门神奇的语言——既健壮又灵活且友好。说到友好，我指的是它能集成大量免费的基于外部源文件编译的扩展库。这也与在开源发展领域中一项重要且始终存在的告诫相一致：即“别重复造轮子”。PHP 能够和许多不同的 Web 之外的软件良好结合，而且实际上它们也是基于 PHP 开发出来的。在本章里，我们就将看到 3 个不同的扩展库，并讨论如何使用这些工具来增强 PHP 的 Web 开发。

将要涉及的这 3 个库全部基于 PHP 面向对象方式，或者您已经熟悉了面向对象编程但在进入实例之前，请务必先读一下本书第 6 章。这里选择的 3 个扩展库有助于实现最新 Web 应用中的一些尖端需求：邮件发送或文字短信服务（SMS）、生成 PDF 文件、生成图形数据报表（如饼图、柱状图等）。

## 电子邮件/短信生成

*Email/SMS Generation*

PHP 已经内建了一个邮件函数叫 `mail()`，它能使用简单邮件传输协议（SMTP）向外发邮件。此函数相当简陋和基础，所以，通常它不是最佳选择，就其本身而言，邮件的任务更重些。例如，要发送带附件的内容，光用 `mail()` 函数就显得捉襟见肘。

有个 PHP 库叫 `PHPMailer`，就是填补这一空白的有效处方。它是基于面向对象的接口，你可以通过 `include` 指令容易地将其集成到脚本里，用 `require` 指令更合适。可以在这个地址找到 `PHPMailer` 库：<http://phpmailer.worxware.com/index.php?pg=phpmailer>。





如果你有控制服务器的权限，并可以解决文件定位问题，你应该考虑将这个库放在普通用户易于访问的目录里，以便共享给所有的 PHP 应用程序，从而防止安装多个重复的库。这也能帮助你实现保持所有网站的库都是最新的。要想让库能用于所有的 PHP 文件，你可以将 class.phpmailer.php 文件移到 *php.ini* 中设定的包含目录里。此项建议绝对适用于本章所有的库。

在你使用 `require` 指令引用了 `PHPMailer` 类之后，就可以简单地初始化此类了。考虑下面这个简单示例，代码取自于 `PHPMailer` 安装手册：

```
require("class.phpmailer.php");

$mail = new PHPMailer();

// 设置使用 SMTP
$mail->IsSMTP();
// 指定主服务器和辅服务器
$mail->Host = "smtp1.example.com;smtp2.example.com";
// 打开 SMTP 认证选项
$mail->SMTPAuth = true;
// SMTP 用户名
$mail->Username = "petermac";
// SMTP 密码
$mail->Password = "secret";

$mail->From = "from@example.com";
$mail->FromName = "Mailer";

// 收件人及回件地址，名字是可选的
$mail->AddAddress("josh@example.net", "Josh Adams");
$mail->AddAddress("ellen@example.com");
$mail->AddReplyTo("info@example.com", "Information");

// 设置换行字符为 50 个
$mail->WordWrap = 50;
// 添加一个附件
$mail->AddAttachment("/var/tmp/file.tar.gz");
// 再添加一个附件
$mail->AddAttachment("/tmp/image.jpg", "new.jpg");
// 设置邮件格式为 HTML
$mail->IsHTML(true);

$mail->Subject = "这是邮件主题";

$mail->Body = "这是 HTML 信息  
正文 <b>加粗显示 !</b>";

$mail->AltBody = "非 HTML 格式纯文本邮件内容";

if(!$mail->Send())
{
```

```

        echo " 消息未发送 . <p>";
        echo " 错误: " . $mail->ErrorInfo;
    } else {
        echo " 消息已发送 ";
    }
}

```

如你所见，包括操作附件在内，用它来构建发送电子邮件的代码可以很明晰、很漂亮。有一件事儿你也可以用它来做，就是循环调用 `$mail->AddAddress` 方法，向从数据库中读取的收件人列表批量发送邮件。如果打算生成大量不同方面的邮件消息，一定要非常熟悉这个类中的方法，这样才能提高你的编码效率。但是请记住，没有人会喜欢收到垃圾邮件！

向手机发送文本信息和发送普通邮件一样简单。同样是用 `PHPMailer` 库，你只需要做少量调整，就可以将短消息发到一部手机上，而不是电子邮件账户。当然了，你必须知道对方的手机号码，并且恰好对方愿意接收你发的短消息。再次建议，为了方便维护这些信息，将其保存到数据库是非常合适的，只要有相应的授权即可。

接下来，你需要收信人的短信服务提供商地址，即SMS域名，例如，在加拿大的服务商 `Bell Aliant`、`Rogers` 和 `Telus`，他们的短信服务地址分别是：`@txt.bell.ca`、`@pcs.rogers.com` 和 `@msg.telus.com`。每个服务商都应该乐意在他们各自的网站上公开这样的域名地址<sup>注1</sup>。

来看一些生成和发送短消息的示例代码：

```

if($OK_to_SMS) {
    $SMSPhoneNum = str_replace('-', '', $CellNumber);

    $sql = <<<SQL
SELECT SMSDomain
FROM SMSProvider
WHERE SMSProviderID = '$SMSProviderID'
SQL;

    $db = new mysqli(
        "localhost",           // 数据库服务器
        $db_username,          // 用户名
        $db_password,          // 密码
        $db_name )             // 数据库名称
    or die(" 连接数据库失败 . 错误代码: %s\n" .
        mysqli_connect_errno());

    $result = $db->query($sql) or die(" 不能执行 SQL: $sql");
    $row = $result->fetch_assoc();
}

```

注 1：某些地区除外。——译者注

```

$body = "尊敬的 $Fname $Lname: \r\n \r\n"
. "我们谨通知你已被选入陪审团, 请履行你的陪审职责。"
. "具体信息请检查您的邮箱 $ContactEmail。";

$mail = new PHPMailer();
$mail->IsSMTP();
$mail->SMTPAuth = true;

$mail->Host = "localhost";
$mail->From = "notification@juryduty.com";
$mail->FromName = "某某镇法庭";
$mail->AddAddress($SMSPhoneNum, $row['SMSDomain']);
$mail->Body = $body;
$mail->WordWrap = 50;

if(!$mail->Send()) {
    echo "陪审职务通知未发送: <br/>";
    echo "短信系统错误: " . $mail->ErrorInfo . "<br/>";
} else {
    echo "陪审职务通知已发送";
}
}

```

注意这里, 我们首先验证是否需要向接收人发送消息。然后, 替换手机号码中可能存在的横线分隔符, 如果电话区号两边存在括号, 你也应该将清理工作考虑在内。之所以如此, 是因为正确的电话号码格式中只有数字, 没有其他标点什么的。接下来的过程和发送电子邮件非常相似, 准备好要发送的文本信息, 再把手机号码和短信服务商合并在一块儿当成电子邮件地址发送出去即可。



记住, 短信一般不会太长, 所以, 你要控制好信息的长度在有限的字数以内。

## PDF生成

Adobe 的便携式文档格式 (PDF) 文件几乎已成为编写良好格式文档的标准。大多数浏览器都可以读取和显示 PDF 文件, 所以, 如果你的 Web 应用程序需要的话, 没有任何理由不为你的用户提供这类格式的文档。标准化的表单和统计报表全部都由 Web 系统的数据生成, 这也是对数据进行通用布局格式化的原因。

PHP 有专门的扩展库允许你生成动态的 PDF 格式的内容输出, 实际上这样的库还不止一个, 但这里我们只看其中使用最广泛的一个, 名叫 FPDF。此库也是面向对象的, 并且

可以像 PHPMailer 那样包含在你的脚本里。

好，开始实践，下面的示例代码将完成三个简单的任务：创建一个空的 PDF 文档、添加一些文字并显示出来。

```
require("../../fpdf/fpdf.php");

$pdf = new FPDF( );
$pdf->AddPage();
$pdf->SetFont('Arial','B',16);
$pdf->Cell(0,10,'PHP - The Good Parts!');
$pdf->Output();
```

从代码中你能看到，在引入库文件之后，我们实例化了一个 FPDF 类的对象，命名为 \$pdf。然后，我们用 AddPage 方法添加一个页面，接着设置输出字体，为字符串输出定义一个位置单元，用 Output 方法在浏览器中显示 PDF 文件。浏览器实际显示的内容如图 8-1 所示。



有时在创建 PDF 页面时，你可能会遇到这样的错误：“FPDF error: Some data has already been output, can't send PDF file.” 这只是表明已经有一些内容输出到浏览器，你要么需要查找并清除无关的输出语句，要么在代码开始和结束时使用 ob\_start 和 ob\_end\_flush 函数，以合并所有的输出。



图8-1 PHP 生成简单PDF文件的输出效果

FPDF 中的文档单元，指的是能够在页面中建立和控制的矩形区域。它可以有高度、宽度、边框，还有文本属性，单元方法的基本语法如下：

```
Cell(float w [, float h [, string txt [, mixed border  
[, int ln [, string align [, int fill [, mixed link]]]]]])
```

Cell 方法的参数有这些：单元宽度 (w)、高度 (h)、所包含的文本 (txt)、边框 (border 可选)、



换行控制 (ln)、文本对齐方式 (align)、单元填充色 (fill 可选)、最后是文本的 HTML 资源链接 (如果需要的话)。



如果你设置单元宽度 (第一个属性) 为 0, 则单元将取整个页面宽度。这仅用于你要在实际页面中显示边框 (参见下面的例子), 或者是在同一水平位置显示不同大小的单元, 如果是后一种, 你可能会遇到文字重叠的情况。

举例来说, 如果我们想改变前一个例子为有边框且文件右对齐, 需要修改方法中的参数如下:

```
$pdf->Cell(0,10,'PHP - The Good Parts! ',1,0,'R');
```

这会在浏览器中产生如图 8-2 所示的输出结果。

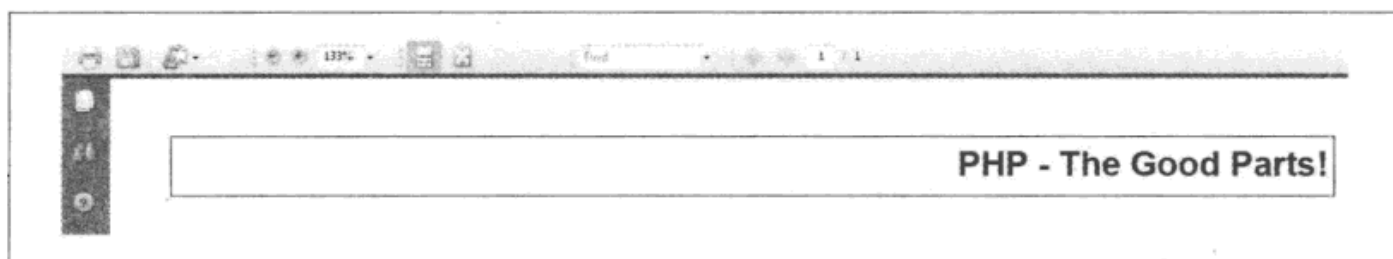


图8-2 PHP 生成有边框且右对齐的PDF页面

Cell 方法是操作 PDF 文档输出的一员主力, 也是以 FPDF 来生成 PDF 文档的常用方法, 所以, 好好花时间学习这个方法的细节和来龙去脉, 定能让你受益匪浅。

理解此方法的换行控制选项 (也包括 FPDF 其他方法) 很重要, 它用来控制方法结束时光标的位置, 在上面的例子里, 它设为 0, 就是说在行操作完成时光标还是停留在同一行, 这样接下来的输出仍然从左边界开始, 由此会造成输出重叠。但如果设为 1, 光标将会移动到下一行并保留前一个调用所定义的高度。

FPDF 还有另外的方法可以让你很方便地在同一行中放置单独的数据, 这个方法名叫 SetX。它可以按页面左边界指定的距离移动光标 (在下一节里我们会谈到有关度量的属性)。听起来或许有些令人困惑, 那么我们来看两个调用 Cell 方法的简单示例。第一个例子会保持光标在同一行, 第二个会将其移动到下一行。

先来看第一个例子的代码和生成效果 (图 8-3):

```
require("../../fpdf/fpdf.php");

$pdf = new FPDF( );
$pdf->AddPage();
```

```
$pdf->SetFont('Arial','B',16);
$pdf->Cell(10,10,'PHP - The Good Parts!', 0,0,'L');
$pdf->SetX(90);
$pdf->Cell(90,10,'Beware the Ides of March!', 1,0,'C');
$pdf->Output();
```

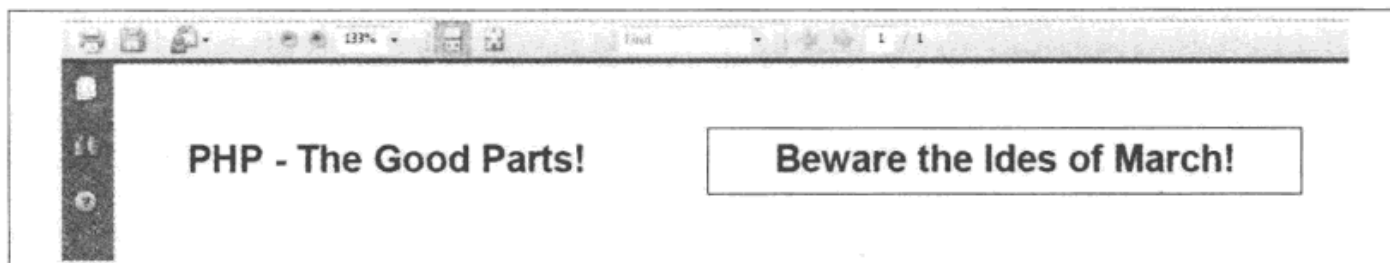


图8-3 输出两个区块在同一行的PDF页面



在使用 FPDF 开发和测试 PDF 页面生成的过程中,你可能想禁止浏览器的缓存功能,因为有些浏览器在改动很小时并不会在缓存控制中作登记,从而不会重新加载页面。

下面是第二个例子的代码和浏览器输出结果 (图 8-4) (我们这里不需要使用 SetX 方法,还是以调用 Cell 方法设置参数的办法来移动光标):

```
require("../..../fpdf/fpdf.php");

$pdf = new FPDF( );
$pdf->AddPage();
$pdf->SetFont('Arial','B',16);
$pdf->Cell(10,10,'PHP - The Good Parts!', 0,1,'L');
$pdf->Cell(90,10,'Beware the Ides of March!', 1,0,'C');
$pdf->Output();
```

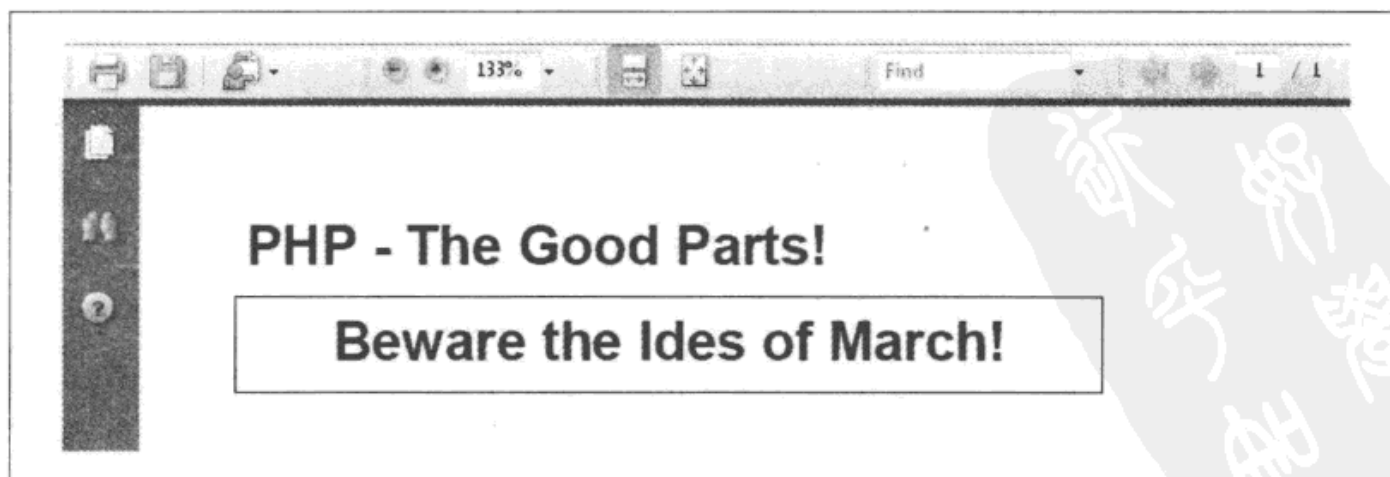


图8-4 生成两个区块在不同行里的PDF页面

早先提到过，FPDF 生成 PDF 时，可以设置不同的页面度量单位。你可以通过在实例化一个新的类副本时，向构造方法传递参数实现。前面你已经看到了 `$pdf = new FPDF()`；这样的实例化语句，这是使用默认参数建立对象。你可以传递以下三类参数给构造方法：

- 页面方向：纵向 (P) 或横向 (L)，默认是纵向。
- 度量单位：点 (pt)、毫米 (mm)、厘米 (cm) 和英寸 (in)，默认为毫米。
- 页面大小：可用选项有 A3、A4、A5、信封 (Letter) 和公文 (Legal) 等，默认是 A4。

以下是定义纵向、度量单位为英寸且尺寸为信封的构造方法调用：

```
$pdf = new FPDF('P', 'in', 'Letter');
```



只要你愿意，甚至可以自定义一个特殊的页面格局，只需要给构造方法的第三个参数传递一个数组结构的面积定义即可。商业卡片或特殊的传单有各自的页面尺寸需求。假如要定义一个 4×5 英寸且为横向结构的页面，构造方法应该这样写：

```
$pdf = new FPDF('L', 'in', array(4,5));
```

## 添加页眉和页脚

*Let's Document Header and Footer*

让我们来瞧瞧如何实现对象的继承和扩展。在多页的 PDF 文档里常常需要页眉和页脚。FPDF 本身有定义一对空的页眉和页脚方法，在每次调用 `AddPage` 方法时其内部都会调用 `Header` 和 `Footer` 这两个方法。要想用我们自己定义的同名方法来达到显示自定义的页眉和页脚而不继承这个类，显然是不行的。所以，让我们来继承此类，并在子类中添加定制页面的 `Header` 和 `Footer` 方法。代码如下：

```
require("../fpdf/fpdf.php");

class myPDF extends FPDF {

    public $title = "FPDF Sample Page Header";

    // 页眉方法
    function Header() {
```

```

        $this->SetFont('Times','',12);
        $w = $this->GetStringWidth($this->title)+150;
        $this->SetDrawColor(0,0,180);
        $this->SetFillColor(170,169,148);
        $this->SetTextColor(0,0,255);
        $this->SetLineWidth(1);
        $this->Cell($w,9,$this->title,1,1,'C',1);
        $this->Ln(10);
    }

    // 页脚方法
    function Footer()    {
        // 距离底边 1.5 厘米
        $this->SetY(-15);
        $this->SetFont('Arial','I',8);
        $this->Cell(0,10,'page footer -> Page '
            . $this->PageNo().'/{nb}','0,0','C');
    }
}

$pdf = new myPDF('P', 'mm', 'Letter');
$pdf->AliasNbPages();
$pdf->AddPage();
$pdf->SetFont('Times','',24);
$pdf->Cell(0,0,'text at the top of the page',0,0,'L');
$pdf->Ln(225);
$pdf->Cell(0,0,'text near page bottom',0,0,'C');
$pdf->AddPage();
$pdf->SetFont('Arial','B',15);
$pdf->Cell(0,0,'Top of page 2 after page header',0,1,'C');
$pdf->Output();

```

在扩展页眉和页脚方法的代码里，还调用了另外几个方法。你在此处看到的整个继承类里并不仅仅只有页眉（Header）和页脚（Footer）两个方法。同样，除了页眉和页脚区域外，显然还调用了其他一些方法。要了解完整的方法列表，请查看 FPDF 产品网站的手册（Manual）链接里的内容。前面代码运行后在浏览器中显示的结果如图 8-5 所示。图中前半部分是一页的页脚，后半部分是下一页的页眉。



你可以在某一页中消除页眉或页脚，还可使用 `PageNo()` 方法来返回当前页面的页码。请确认在对文档添加第一页之前先调用 `AliasNbPages` 方法，这样 FPDF 会在页面创建时自动统计页数。



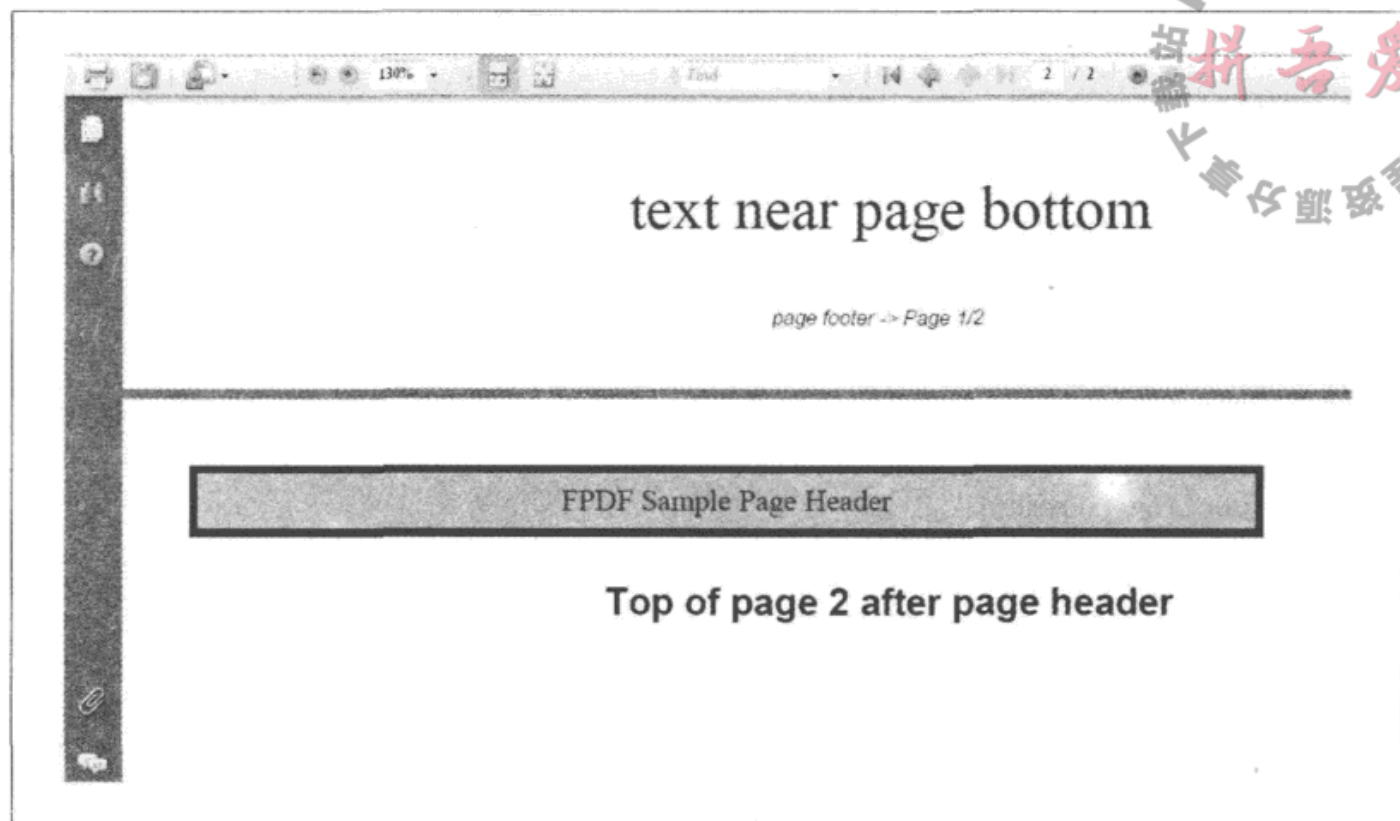


图8-5 生成带页眉和页脚的PDF页面

## 添加图片和链接

使用 FPDF 库，你还能用 `Image()` 方法在 PDF 文件中添加图片和链接内容。链接可以是锚点或者完整的网站 URL 地址。首先，我们来看看在 PDF 文件中插入图片，然后，我们将从一张图片或文字上创建一个链接。

要在文档中添加一张图片，直接使用 `Image` 方法即可。在下面的代码中，我们将 `Image` 方法添加到 `Header` 函数里，实现在页眉中添加 PHP 图标，并且移除之前 `Cell` 方法中的背景填充选项，这样就能让图片显示出来。`Image` 方法的参数包括：图片文件名、图片的  $X$  和  $Y$  坐标、图片的宽度和高度等。

```
function Header() {
    $this->SetFont('Times','',12);
    $w = $this->GetStringWidth($this->title)+150;
    $this->SetTextColor(0,0,255);
    $this->SetLineWidth(1);
    $this->Image('phplogo.jpg',10,10.5,15,8.5);
    $this->Cell($w,9,$this->title,1,1,'C');
    $this->Ln(10);
}
```

此 PDF 文档现在看起来如图 8-6 所示。

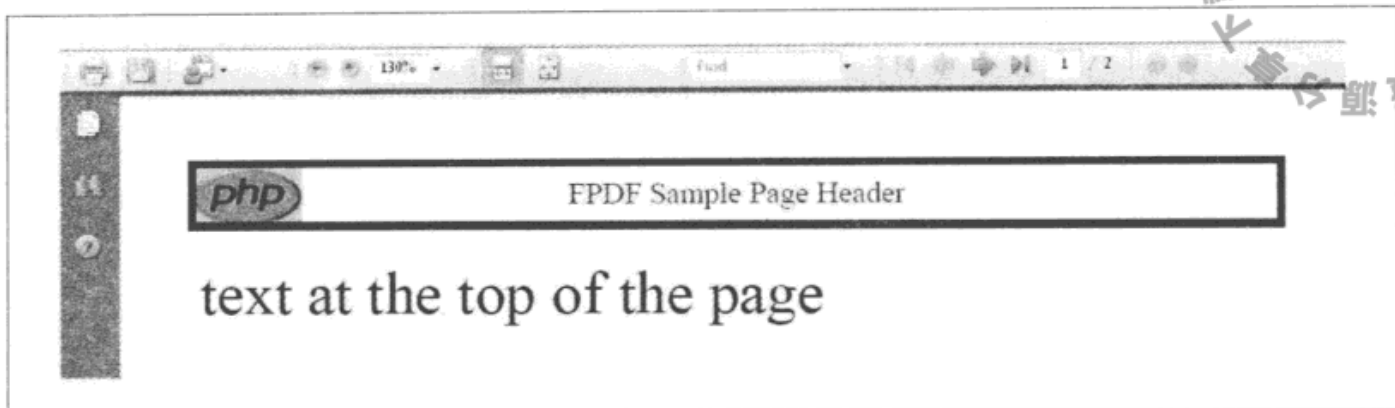


图8-6 生成页眉包含图片的 PDF 页面

现在，给图片添加一个指向 PHP 主页的链接，直接在 **Image** 方法的最后一个参数后添加一个 URL 即可。跳过图片类型参数可以用逗号 (,)。



先将 URL 赋给某个字符串变量，再将其添加到方法的参数中，这是个不错的主意。这会让代码更易于阅读，如果文档中有其他的链接也用到这个 URL，还可以重复利用。

添加新链接的方法代码如下所示：

```
$php_url = "http://www.php.net" ;  
$this->Image('phplogo.jpg',10,10.5,15,8.5,"",$php_url);
```

这样图片就成了一个可以单击的有链接的图片，当鼠标覆盖上去的时候会有链接提示，如图 8-7 所示。

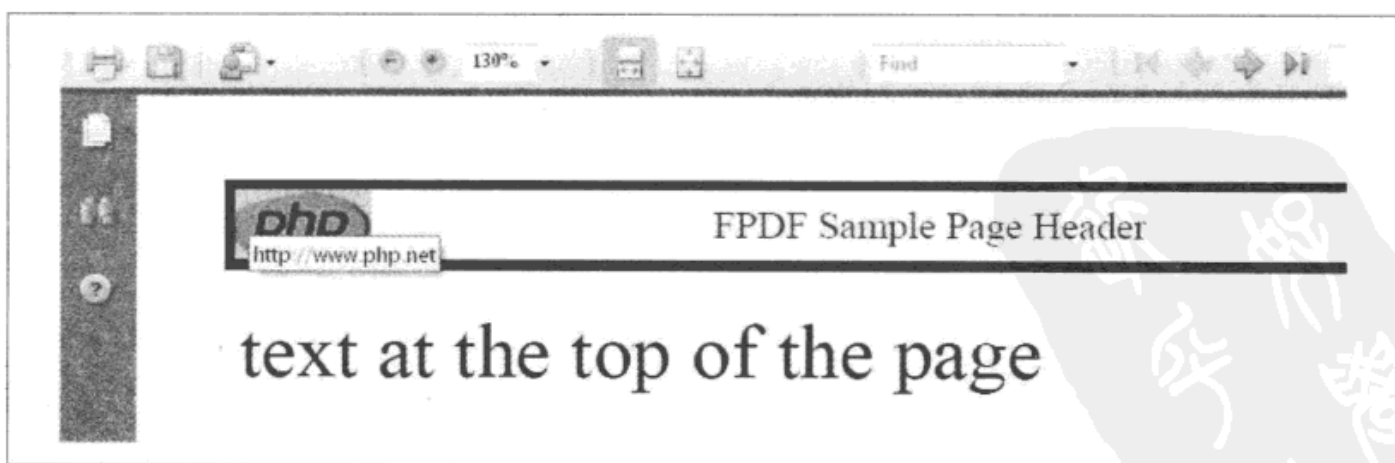


图8-7 插入带有活动链接的图片

还有另外一种链接形式，我们可以在 PDF 文档中添加一个指向此文档中锚点位置的链接。此概念包含表格内容或索引。创建一个内部链接有两部分工作。首先，你要定义一个起点链接（即链接本身），然后，要设置一个锚点（就是当点击链接时要定位的目标）。

设置一个起点链接，要用 `AddLink` 方法。此方法会返回一个可用的句柄，它指向调用 `SetLink` 方法时产生的目标位置，当把句柄作为链接参数时，它会在起点链接和目标这两者之间建立连接。下面的一些示例代码，实现了创建起点链接和目标两部分内容。注意，这里用了 FPDF 的 `Write` 方法，它是另一个在文档中写入文字的办法（和 `Cell` 方法截然不同）。

```
require("../../fpdf/fpdf.php");

$pdf = new FPDF();

// 第一页
$pdf->AddPage();
$pdf->SetFont('Times','',14);

$pdf->write(5,'For a link to page 2 - Click ');
$pdf->SetFont('','U');
$pdf->SetTextColor(0,0,255);
$link_to_pg2 = $pdf->AddLink();
$pdf->write(5,'here',$link_to_pg2);
$pdf->SetFont('');

// 第二页
$pdf->AddPage();
$pdf->SetLink($link_to_pg2);
$pdf->Image('phplogo.jpg',10,10,30,0,'','http://www.php.net');
$pdf->ln(20);
$pdf->SetTextColor(1);
$pdf->Cell(0,5,'This is a link and a clickable image', 0, 1, 'L');
$pdf->SetFont('','U');
$pdf->SetTextColor(0,0,255);
$pdf->Write(5,'www.oreilly.com','http://www.oreilly.com');
$pdf->Output();
```

图 8-8 和图 8-9 分别显示了代码中的链接和目标页面的浏览器运行结果。

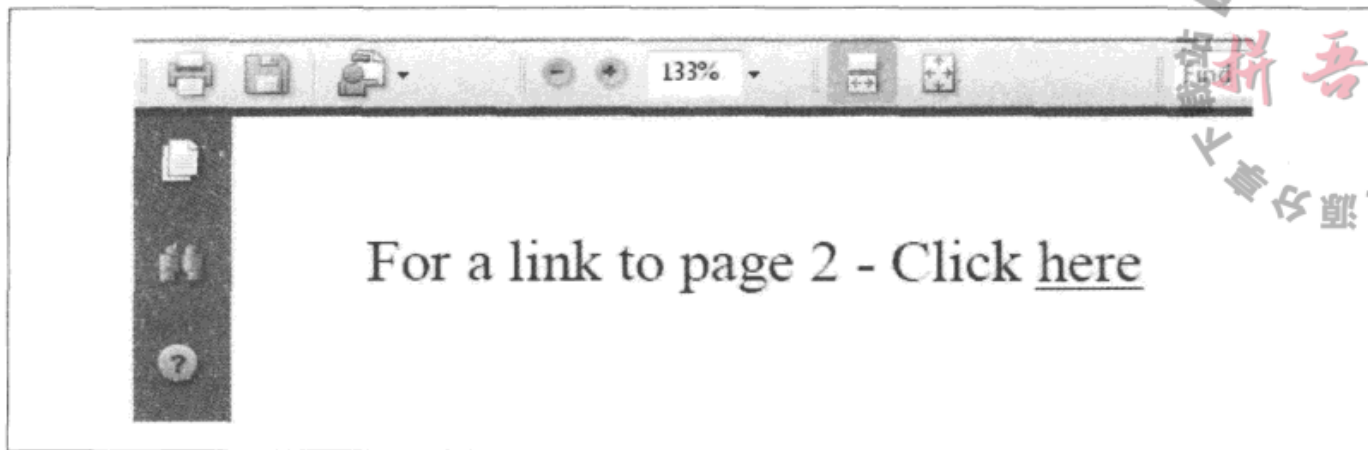


图8-8 生成有内部链接的 PDF 文档



101

图8-9 示例代码中有锚点和外部链接的 PDF 文档

## 添加水印

下一个特征，我们来看看如何让 PDF 文档中显现水印。在你想用 PHP 程序生成报表或销售手册时，这会是个蛮不错的技巧。来看看添加水印的代码：

```
require("../..../fpdf/fpdf.php");

$pdf = new FPDF( );
$pdf->AddPage();
```



```
$pdf->SetFont('Arial','B',16);
$pdf->SetXY(26,100);
$pdf->image('php_watermark.jpg');
$pdf->SetY(35);
$text = "This is sample text to show the watermark underneath it.";
for($i = 0; $i < 35; $i++) { $pdf->Cell(0,5,$text,0,1); }
$pdf->Output();
```

所有这一切，真正发生的就是通过 SetXY 和 SetY 方法让光标在页面上来回移动，而且我还有一张设置成半透明有阴影效果的图片。这其实和在页面中添加图片没什么区别，除了我们用额外的文字覆盖在图片上。如果没有用半透明的图，文字和图片将会混在一起，那样会看起来乱糟糟的。

要确保你在文档一开始就添加了可用作水印的图片，如此方能让后来添加到文档的任何东西覆盖在上面。在图 8-10 里，如果文字先发送出来，图片跟在文字后面，那么图片就会覆盖在文字上。

102

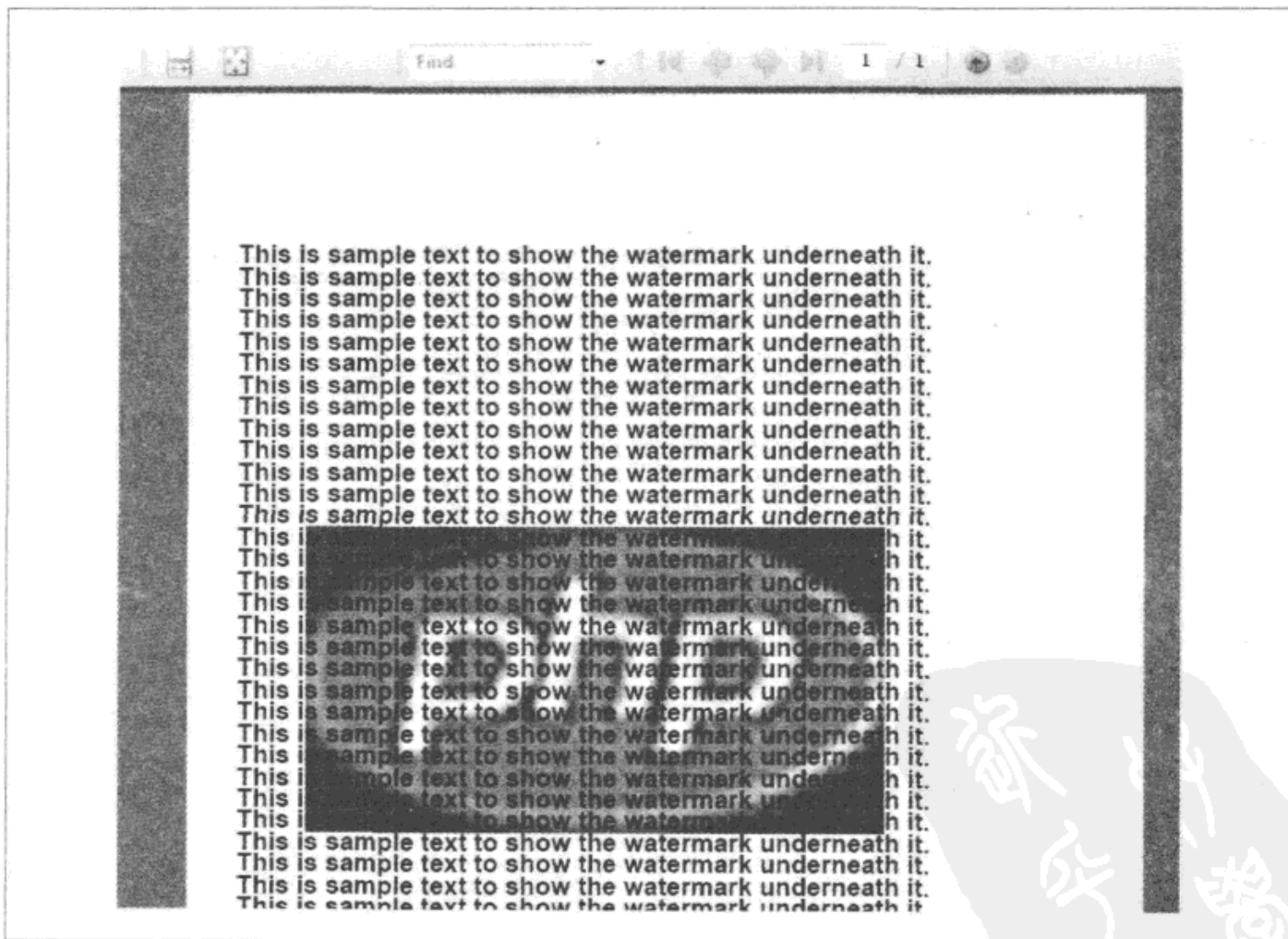


图8-10 生成含水印的 PDF 文档

## 显示动态 PDF 文件和表格

现在，我们要让 FPDF 库真正值得保留。到目前为止，我们在创建 PDF 时都只是添加了静态内容。让我们看看如何生成一个包含数据库查询结果的完整 PDF 文档。我们要在 PDF 里显示一个经过美化的数据表格，以此在 PDF 文件中加入动态内容。下面列出的代码有些长，但它有良好的注释，后文中也有详细论述。

```
require("../../fpdf/fpdf.php");

class PDF extends FPDF {

    function BuildTable($header,$data) {
        // 设置颜色、行高及加粗字体
        $this->SetFillColor(255,0,0);
        $this->SetTextColor(255);
        $this->SetDrawColor(128,0,0);
        $this->SetLineWidth(.3);
        $this->SetFont('', 'B');
        // 页眉
        // 通过数组设定列宽
        $w=array(85,40,15);
        // 向 PDF 发送页眉
        for($i=0;$i<count($header);$i++)
            $this->Cell($w[$i],7,$header[$i],1,0,'C',1);
        $this->Ln();
        // 颜色和字体复位
        $this->SetFillColor(175);
        $this->SetTextColor(0);
        $this->SetFont('');

        // 脱机输出 $data 数组

        $fill=true; // 用于交替显示行背景色
        foreach($data as $row)
        {

            $this->Cell($w[0],6,$row[0], 'LR',0,'L',$fill);
            // 设置显示 URL 风格的链接颜色
            $this->SetTextColor(0,0,255);
            $this->SetFont('', 'U');
            $this->Cell($w[1],6,$row[1], 'LR',0,'L',$fill, 'http://www.oreilly.com');
            // 恢复正常颜色设置
            $this->SetTextColor(0);
            $this->SetFont('');
            $this->Cell($w[2],6,$row[2], 'LR',0,'C',$fill);

            $this->Ln();
            // 反转真假值
        }
    }
}
```

```

        $fill =! $fill;
    }
    $this->Cell(array_sum($w),0,'','T');
}

}

// 连接数据库

$conection = mysql_connect("localhost","user", "password");
$db = "library";
mysql_select_db($db, $conection)
    or die( "不能连接数据库: $db");

$sql = 'SELECT * FROM books ORDER BY pub_year';
$result = mysql_query($sql, $conection)
    or die( "不能执行 sql: $sql");

// 从数据库记录结果中建数组
while($row = mysql_fetch_array($result)) {
    $data[] = array($row['title'], $row['ISBN'], $row['pub_year'] );
}

// 开始创建 PDF
$pdf = new PDF();

// 列的标题
$header=array('Book Title','ISBN','Year');

$pdf->SetFont('Arial','',14);
$pdf->AddPage();
// 调用生成表格的方法
$pdf->BuildTable($header,$data);
$pdf->Output();

```

104

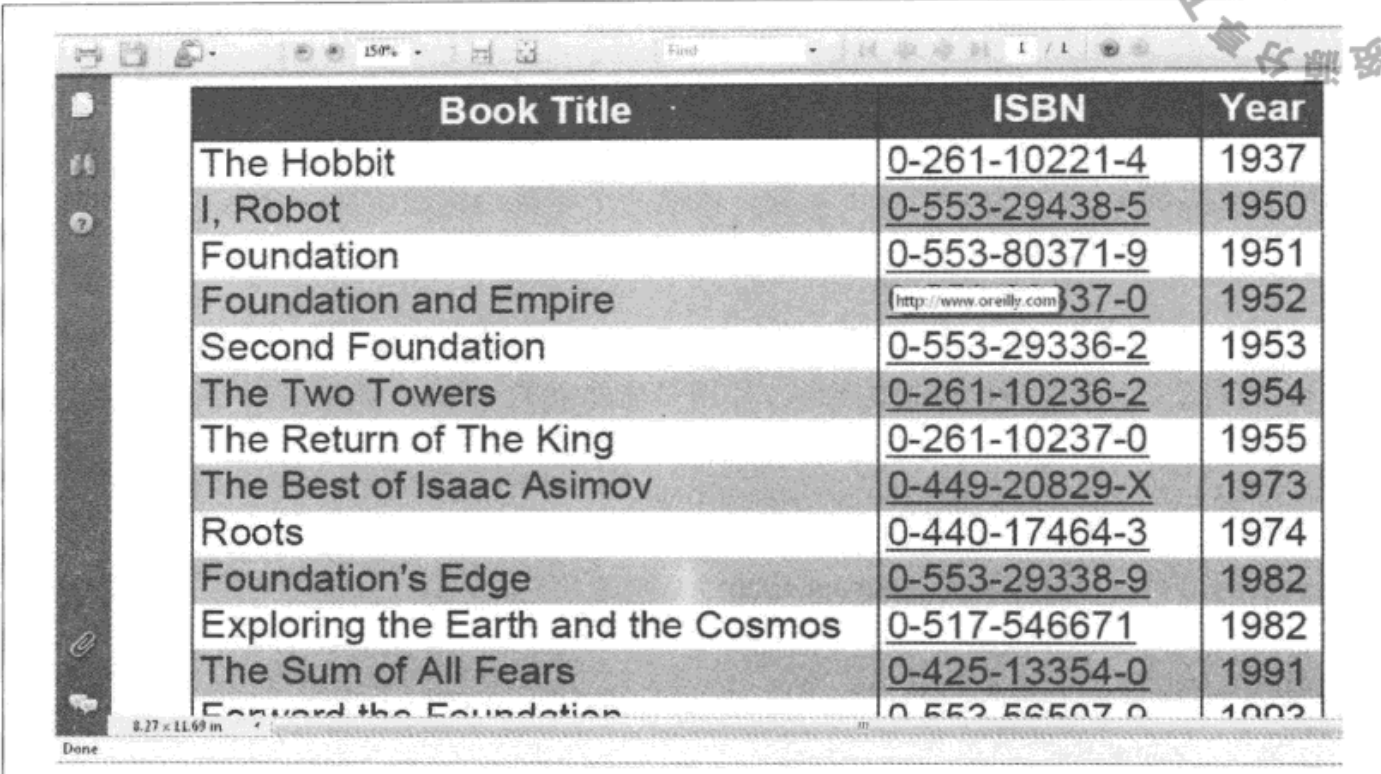
在这段代码中，我们先连接数据库，再将两个数组传给继承类中定制的 BuildTable() 方法。在 BuildTable() 方法里面，我们设置了表头的颜色和字体属性，然后将通过参数传入的第一个数组作为表头的内容发出。变量 \$w（宽度）数组是列宽的集合，此数组在调用 Cell() 方法的时候用得到。

在送出表头之后，我们通过 foreach 循环来遍历包含数据库信息的 \$data 数组。注意，这里的 Cell() 方法用 LR 作为边框参数，意为在单元中显示左边框和右边框，这样就像是添加了单元格的那种效果。我们还在第二列里添加了一个 URL 链接，这只是为了表明表格里也能建立连接。最后，我们用 \$fill 变量来回反转以实现背景色在表格的行与行之间交替变化。

BuildTable() 最后调用 Cell() 方法来显示表格底边和结束列操作。



代码的运行结果在浏览器中显示如图 8-11 所示。



Book Title	ISBN	Year
The Hobbit	0-261-10221-4	1937
I, Robot	0-553-29438-5	1950
Foundation	0-553-80371-9	1951
Foundation and Empire	<a href="http://www.oreilly.com">http://www.oreilly.com</a> 37-0	1952
Second Foundation	0-553-29336-2	1953
The Two Towers	0-261-10236-2	1954
The Return of The King	0-261-10237-0	1955
The Best of Isaac Asimov	0-449-20829-X	1973
Roots	0-440-17464-3	1974
Foundation's Edge	0-553-29338-9	1982
Exploring the Earth and the Cosmos	0-517-54667-1	1982
The Sum of All Fears	0-425-13354-0	1991
Forward the Foundation	0-553-56507-0	1992

图8-11 从 MySQL 读取数据来生成动态 PDF文件

## 图形报表生成

### Graphical Reports Generation

JPGraph 库通常用来建立图形化的统计报表,像柱状图和饼图。它也是面向对象的代码库,所以现在你应该可以很轻松地使用它。和前面差不多,使用 `require` 语句来开始对这个库的使用。

一个典型的图形化报告会依照使用者输入的要求来建立,这些要求包括报告所涵盖的日期范围、数据排序方法等项。然而在我们的例子里,为查阅方便,只简单地使用事先准备好的数值来建立这些图形。

### 饼图

#### Example 8-12

首先我们来看看饼图。下面所列出的代码中,你能看到有个叫 `$data` 的数组变量,它提供用于绘制图形的数据,并被分配给制图对象。此变量的数据一般是由数据录入页面提



供，也可由一个 SELECT 语句在数据库查询出的结果提供，或者是二者的结合。在引入即将建立的图形的相关库之后，就可以做到这一点。

JPGraph 库和其他库感觉上有一些不同，就是它有一个所有图形生成都必需的基本库，还有个别特殊的库或子库，每个都适合一种图形。在当前情况下，因为我们要建立一个饼图，所以要用 jpgraph\_pie.php 文件。在引入正确的库并提供了原始数据之后，我们从饼图类实例化一个 \$piechart 对象，传递两个参数给构造函数，分别代表图的宽度和高度。然后，我们就能使用此对象里的方法来建立图形了。

我们能用设定的文字、字体和颜色来控制所生成图形的标题。然后，我们实例化一个叫 \$pPlot 的对象，用来渲染出圆饼的形状，并用早先建立的 \$data 数组进行切分。接下来，我们要描绘每一个切片的标签。最后，我们用 Add 方法将描绘完成的圆饼形状添加到饼图中，再用 Stroke 方法将图形完整地绘制出来：

```
include ("../jpgraph/jpgraph.php");
include ("../jpgraph/jpgraph_pie.php");

$data = array(12, 15, 23, 18, 5);

// 建立一个饼图容器
$piechart = new PieGraph(300,350);

// 设置标题
$piechart->title->Set("Sample Pie Chart");
$piechart->title->SetFont(FF_VERDANA,FS_BOLD,12);
$piechart->title->SetColor("darkblue");
$piechart->legend->Pos(0.1,0.2);

// 建立一个饼形状
$pPlot = new PiePlot($data);
$pPlot->SetCenter(0.5,0.55);
$pPlot->SetSize(0.3);

// 设置标签
$pPlot->SetLabelType(PIE_VALUE_PER);
$pPlot->value->Show();
$pPlot->value->SetFont(FF_ARIAL,FS_NORMAL,9);
$pPlot->value->SetFormat('%2.1f%%');

// 添加和绘制
$piechart->Add($pPlot);
$piechart->Stroke();
```



这里的 time 函数可以让浏览器缓存触发差异请求登记，以便于同样的文件被多个页面中同样的用户使用。

代码生成的图形在浏览器中显示如图 8-12 所示。记住，如果需要的话，你可以用其他 HTML 标记来增大显示效果。

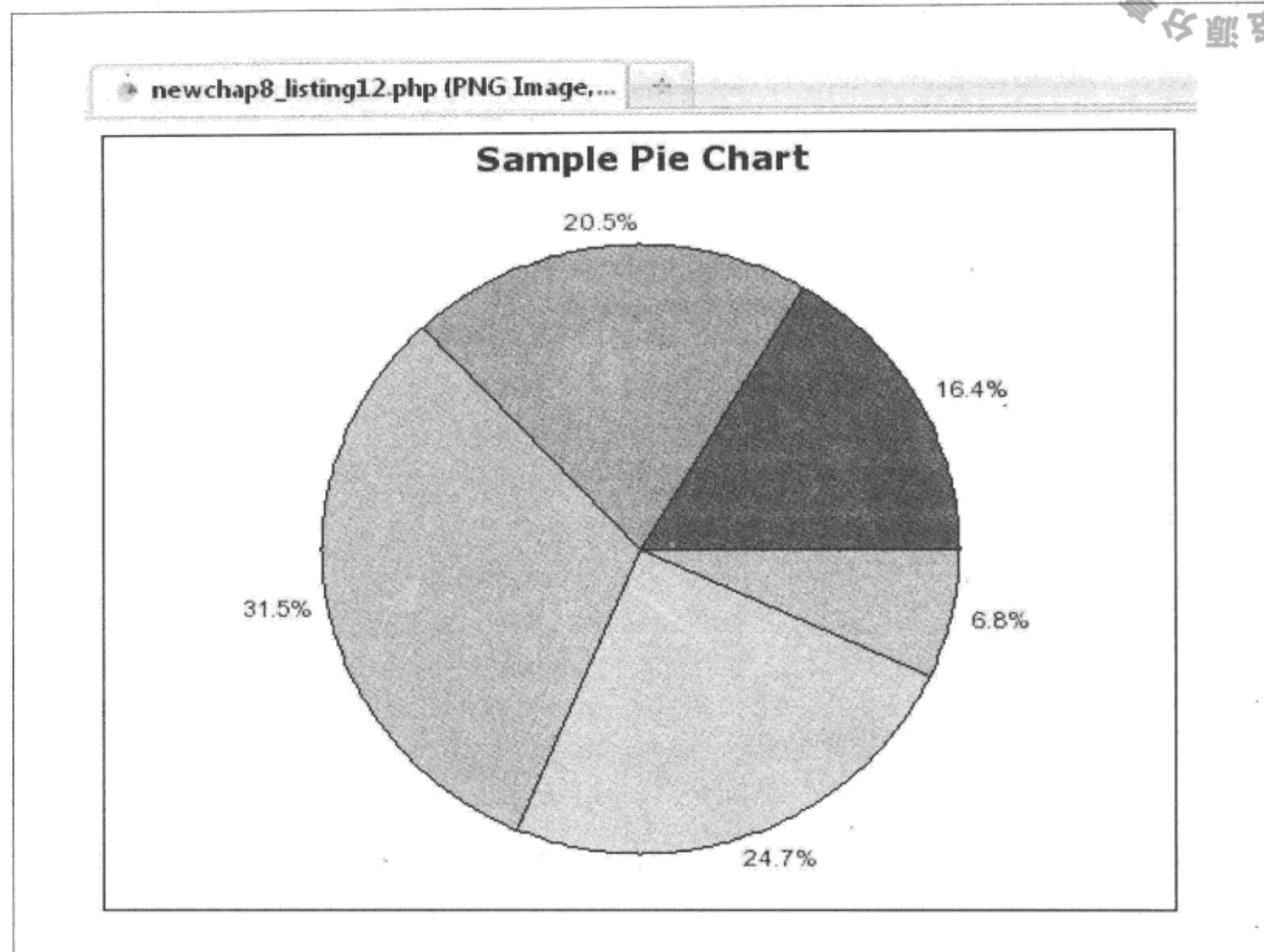


图8-12 用 JGraph 生成的饼图



若是决定添加其他的输出格式，你会将生成的图形保存为文件并放到服务器的合适位置，供日后随 HTML 显示。Stroke 方法生成图形时常常会跟一个可选的文件名参数，这可以为你保存图片文件。保存图形的代码如下：

```
$graph->Stroke("graph.jpg");
```

将图形带到 HTML 标签中的代码是：

```
echo ('');
```

## 柱状图

我们能创建的另外一种图形是柱状图。同样的，为了易于实现，我们将提供生成图形的基准数据。在下面的示例代码中，你将能看到，它用了适合此次示例的柱形图子库。除了选择正确的子库，和前面的并无太多差别，而且很接近——虽有各自特殊的方法，但概念上是一样的。来看代码：

```
include ("../../jpgraph/jpgraph.php");
include ("../../jpgraph/jpgraph_bar.php");
include ("../../jpgraph/jpgraph_line.php");

// 我们需要一些数据
$datay=array(31,44,49,40,24,47,12);

// 设置图形容器
$graph = new Graph(600,300,"auto");
$graph->img->SetMargin(60,30,30,40);
$graph->SetScale("textlin");
$graph->SetMarginColor("teal");
$graph->SetShadow();

// 建立一个柱形
$bplot = new BarPlot($datay);
$bplot->SetWidth(0.6);

// 设置渐变填充的颜色
$tccl=array(100,100,255);
$fccl=array(255,100,100);
$bplot->SetFillGradient($fccl,$tccl,GRAD_VERT);
$bplot->SetFillColor("orange");
$graph->Add($bplot);

// 设置图形标题
$graph->title->Set("Sample Bargraph");
$graph->title->SetColor("yellow");
$graph->title->SetFont(FF_VERDANA,FS_BOLD,12);

// 设置坐标和标签
$graph->xaxis->SetColor("black","white");
$graph->yaxis->SetColor("black","white");

// 设置坐标字体
$graph->xaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
$graph->yaxis->SetFont(FF_VERDANA,FS_NORMAL,10);
$graph->yaxis->title->Set("Value Range");
$graph->yaxis->title->SetColor("white");
$graph->yaxis->title->SetFont(FF_VERDANA,FS_NORMAL,10);
```

```
// 设置 X 坐标的标题 (颜色和字体)
$graph->xaxis->title->Set("item Count");
$graph->xaxis->title->SetColor("white");
$graph->xaxis->title->SetFont(FF_VERDANA,FS_NORMAL,10);

// 最后, 将图形发送到浏览器
$graph->Stroke();
```

代码产生的效果如图 8-13 所示。

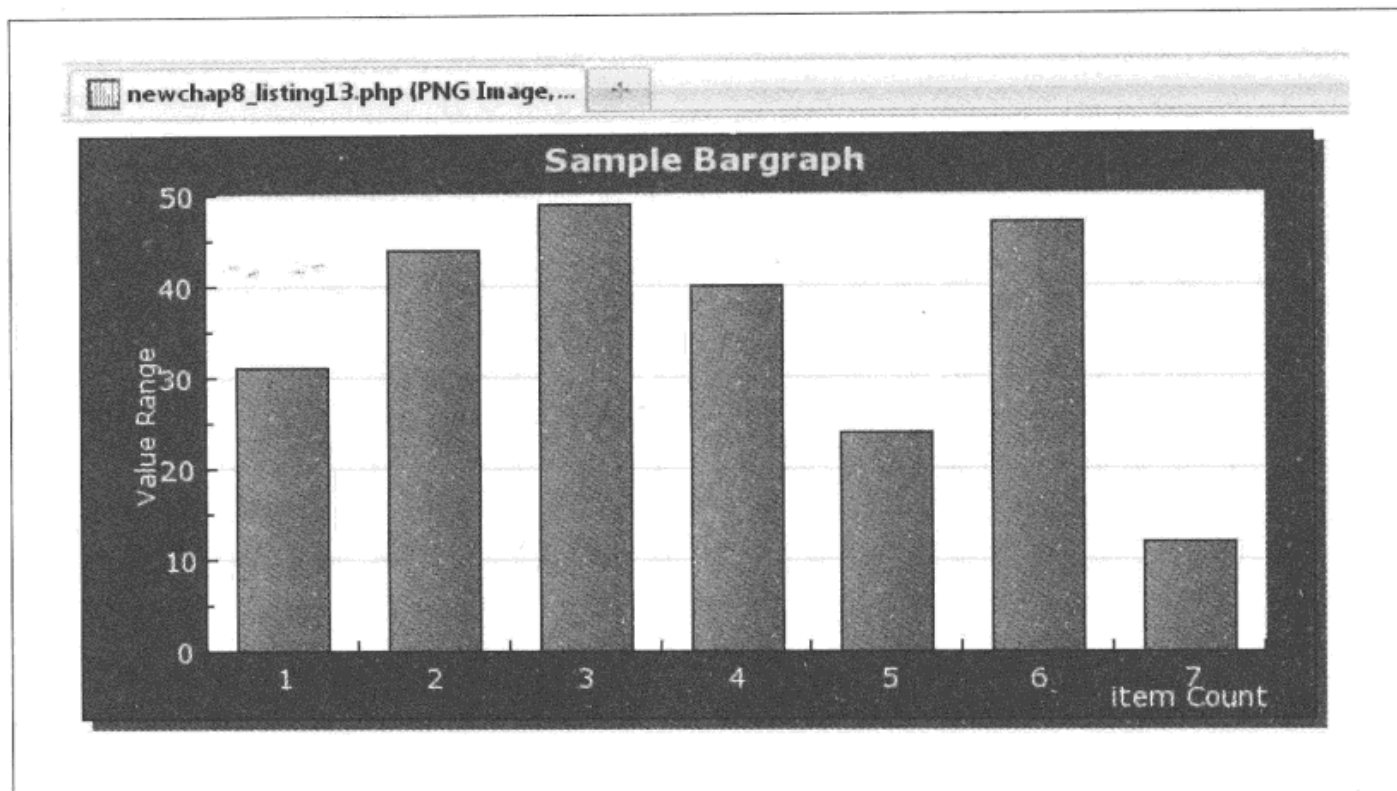


图8-13 用 JPGraph 生成的柱状图

## 图形验证码

Captchas

下面是最后一个快速示例了。如果你曾经在网上订购过演唱会门票, 应该对如图 8-14 所示的验证码很熟悉。

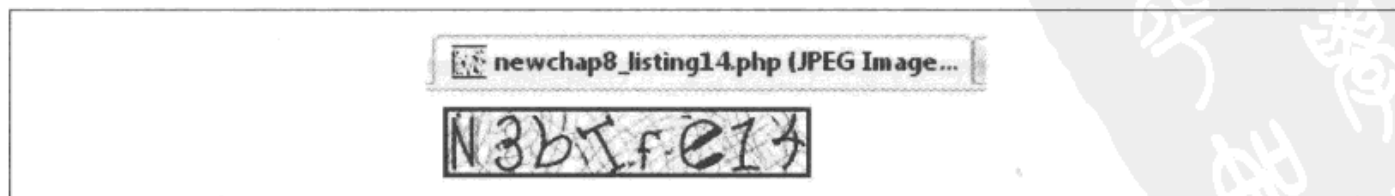


图8-14 用 JPGraph 生成的图形验证码



JPGraph 只需要很少几行代码就能生成防垃圾请求的图形验证码，而且支持手动指定字符范围和随机生成两种方式。

```
require_once "../jpgraph/jpgraph_antispam.php";

$spam = new AntiSpam();
// 保存 $chars 变量里的内容，以便稍后进行验证
$chars = $spam->Rand(8);
$spam->Stroke();
```

要查阅所有可能提供的图形选项，请一定要访问此库的官方网站。例如，你能够给一张图添加背景或调整柱状图里面的风格线等更多情况。还有很多可用的图形种类，像折线图、雷达图、散点图、极坐标图，以及甘特图。

# PHP的安全性

在当今这个充斥着身份盗用和信用卡欺诈的世界里，对一个网站而言，安全理所当然地成为重中之重。现实已经一次又一次地证明了没有百分之百的万无一失，因此我们有必要在保证信息和网站安全方面多投入些精力。在使用 PHP 时，有不少最佳实践可以让我们降低风险。

任何操作数据的地方都有可能成为网站安全最薄弱的环节。这听起来似乎显得有点模糊并且空洞，我的本意是让你留心网站上那些潜在攻击者可能感兴趣的地方。接下来让我们通过几个例子来看看如何降低风险。

## 数据验证

网站上任何允许输入数据的地方都是潜在的风险区域：表单、搜索框、查询字符串等。这里的经验法则是怀疑一切外来数据，并且尽可能过滤。过滤是什么意思？即检查数据，如果需要还可以修改数据，如果不满足条件可以丢弃数据。这是最基本的数据验证。在“跨站脚本（XSS）和 SQL 注入”一节中我们会深入探讨这个问题。

数据可以通过 `$_GET` 和 `$_POST` 这类超级全局数组（以及它们的父实体 `$_REQUEST`）传递给表单。数据也可以通过 `$_COOKIE` 和 `$_SESSION` 数组传递给一个网站。让我们来看看一般情况下怎么处理这类信息。你必须知道数据是从那里来的（表单由你所建立），否则的话此信息将会是无害的。假使说在 `$_COOKIE` 中的数据是有害的，但你从来不使用它的话，你也总是安全的。

对于捕捉（或拦截）提交数据而言，数组非常合适，在代码开头初始化一个空数组，这样你就可以确保它的内容始终是干净的，并且在你的控制之下。事实上，初始化变量是

一个好习惯，它让你一开始就能控制变量的内容，为此我使用一个名为 \$trusted 的数组，举一个很基本的表单提交的例子，包括名字、姓氏、电话号码，其中名字最多包含 35 个字符，姓氏最多包含 45 个字符，电话号码只能是 10 个数字（区号不用加括号，分机也不用加破折号），这个表单使用 POST 方法提交，代码如下所示（纯 HTML）：

```
<html>
<body>
<table>
<form method=' post' action=' chap8_listing2.php' >
<tr>
    <td> First Name:</td>
    <td> <input type=' text' name=' firstname' size=35> </td>
</tr>
<tr>
    <td> Last Name:</td>
    <td> <input type=' text' name=' lastname' size=45> </td>
</tr>
<tr>
    <td> Phone:</td>
    <td> <input type=' text' name=' phone' size=10> </td>
</tr>
<tr>
    <td colspan=2><input type=' submit' value=' Submit' ></td>
</tr>
</table>
</body>
</html>
```

下面是接收提交并过滤输入的代码：

```
$trusted = array() ;

if (strlen($_POST[ 'firstname' ]) <= 35) $trusted[ 'firstname' ] = $_
POST[ 'firstname' ] ;

if (strlen($_POST[ 'lastname' ]) <= 45) $trusted[ 'lastname' ] = $_
POST[ 'lastname' ] ;

if (strlen($_POST[ 'phone' ]) <= 10 && is_numeric($_POST[ 'phone' ]))
    { $trusted[ 'phone' ] = $_POST[ 'phone' ] ;
    }

var_dump($trusted) ;
```

如果每个字段都输入有效信息，那么 var\_dump 的输出效果如下所示：

```
array(3) { [ "firstname" ] => string(5) "Peter" [ "lastname" ] => string(9)
"MacIntyre" [ "phone" ] => string(10) "9025551234" }
```



这段代码里使用 `is_numeric` 方法验证，而不是 `is_int` 方法，这是因为 HTML 表单里的 `text` 类型字段内容是作为字符串返回的，可以在输出中确认这一点，数组里的电话号码列出的数据类型是 `string(10)`。如果你确实想测试数值类型的数据，可以先把输入数据转换成整型。

有许多其他的数据验证和维护方法可供使用，例如强制首字母大写 (`ucfirst` 函数)、强制所有单词首字母大写 (`ucwords` 函数)、确保一个字段非空 (`if ($fname == "") { //raise an error }`) 等，这里不可能列举出所有的验证方法，因为它们依赖于表单数据和你想用这些数据做什么。这里的关键是从外部（不可信）数据源获取的数据必须通过拦截和处理后展现在网页中。尽管这并不完全是安全编程技术，但有助于实现数据的一致性，同时也有助于数据在保存到数据库之前处于原始状态。

## 转义输出

### *Example Output*

网络数据安全的另一面是信息通常发送到两个目的地之一：浏览器作为显示输出或者数据库作为存储输出。一个常见的错误是想当然地以为在接收数据的时候已经过滤过了，所以它们是没有问题的，因此可以直接输出。牢记安全告诫：永远不要完全信任你的数据，换句话说就是“防患于未然”。

先看一下将数据输出到浏览器的情况。记住，PHP 本质上是一个 HTML 生成器，当你显示的是原始 HTML 时，实际的代码会显示在用户浏览器上，为了帮助 PHP 得到正确的 HTML，我们可以使用 `htmlspecialchars` 函数，这个函数会把相关的数据转换成 HTML 实体，这样就只会显示它们而不是执行它们，下面是一个例子：

```
$string = "' Fred' & 'Barney' <a href=' http://www.abadsite.com' >
click here, you know you want to</a>" ;
// 请注意这里的单引号

echo htmlspecialchars($string);
echo "<br/>" ;
echo htmlspecialchars($string, ENT_QUOTES);
```

注意第二次 `htmlspecialchars` 的函数调用中，使用了一个常量作为第二个参数，它告诉 PHP 如何处理字符串中的引号。表 9-1 列出了这个参数中的多个选项。

114



表9-1 htmlspecialchars函数中的常量选项

常量	用法
ENT_COMPAT (默认)	转换双引号并保留单引号
ENT_QUOTES	双引号和单引号都会转换
ENT_NOQUOTES	双引号和单引号都不会转换

浏览器输出字符串：

```
'Fred' & 'Barney' <a href=' http://www.abadsite.com' >click here,
you know you want to</a>
'Fred' & 'Barney' <a href=' http://www.abadsite.com' >click here,
you know you want to</a>
```

相关源代码显示如下：

```
'Fred' &amp; 'Barney' &lt;a href=' http://www.abadsite.com' &gt;
click here, you know you want to&lt;/a&gt;
<br/>
&#039;Fred&#039; &amp; &#039;Barney&#039; &lt;a href=&#039;
http://www.abadsite.com&#039;&gt;click here, you know you want to&lt;/a&gt;
```

正如你所看到的，如果有人想在你的网址中插入一条链接，那么仅仅会输出文本，链接是未被激活的（可单击）。



htmlspecialchars 函数只能处理 HTML 元素最常见的形式。比如说 &、双引号、单引号、大于号 (>)、小于号 (<)。如果你想处理其他的符号，可以使用 htmlentities 函数，它能搞定所有的 HTML 元素。

PHP 应用中下一个常见的输出目的地是数据库，绝大多数 PHP 网站都使用 MySQL 作为数据存储，PHP 提供了一个函数可以转义准备发送到数据库的数据：mysql\_real\_escape\_string。当 SQL 命令会作为字符串发送到数据库时，其中某些潜在的字符可能会意外地终止你的内容。看一个例子：当你在姓氏中插入“O’ Mally”时：

```
$sql = "UPDATE team SET name = 'O' Mally' WHERE teamid = 15";
```

如果没有转义，当这个字符串发送到数据库时，实际上会报错，因为中间的单引号会终止先前的字符串，当姓氏数据来自一个变量时，这很难预知，下面是报错信息：

115

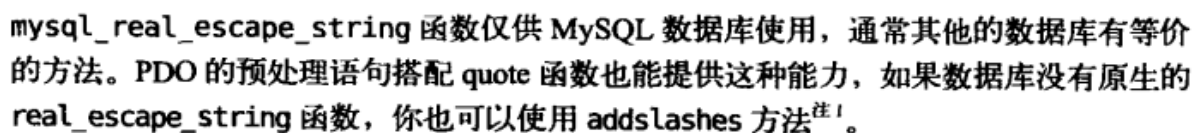
```
ERROR: Unclosed quote @ 31
```

```
STR: '
```

```
SQL: UPDATE team SET name = 'O' Mally' WHERE teamid = 15
```

解决方法是把数据通过 mysql\_real\_escape\_string 函数转义，然后再发送到数据库。具体代码如下所示：

pin51.com  
最新编程资源分享  
站载拼吾爱

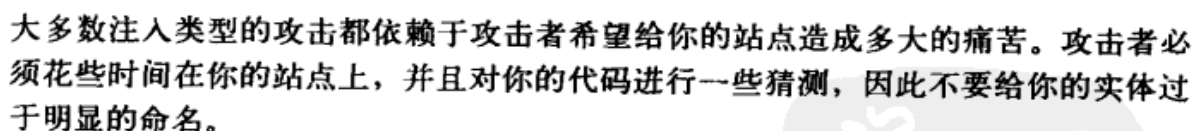


```
UPDATE team SET name = 'o\' mally' WHERE teamid = 15
```

## 跨站脚本 (XSS) 和SQL注入

当攻击者想往你的网站中注入一些代码（JavaScript 或者 SQL）时，通常会使用跨站脚本和 SQL 注入。表单中任何数据入口都可能成为这种类型工具的目标。下面并不是教你如何黑别人，只是做一个简单测试：在网站搜索框中输入下面脚本并提交：

如果弹出一个警告框，那么这个网站就是有问题的。图 9-1 显示了存在这个问题的网站。



```
flintstone' ; drop table customers;
```

注1：详见 <http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string>。——译者注

同时使用 MySQLi 的多重查询，但这样奇怪的事情确实发生了。在上述情形下有许多方法来保护你的站点。方法之一是对输入进行过滤并且不允许任何分号。另一个方法是保证网页用户（你用来访问数据库的标识）没有删除表格的权限。

116

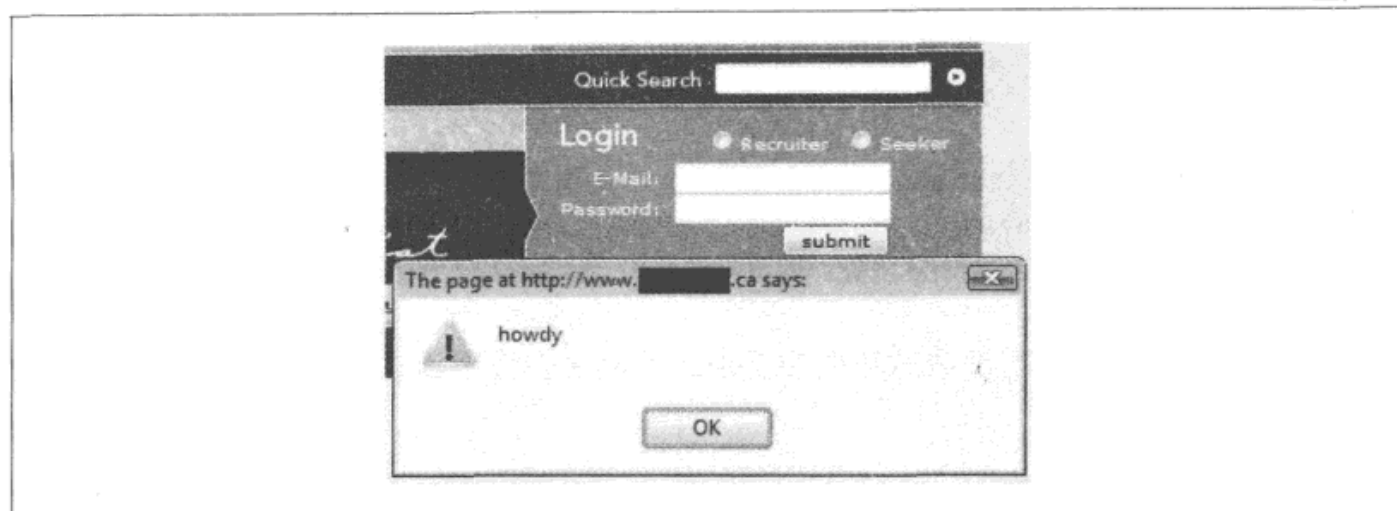


图9-1 浏览器显示XXS注入漏洞

使用前面章节描述过的两种方法（过滤输入和转义输出），能有效避免这些类型的攻击。

## 密码加密安全

最后一个重要的安全问题是确认你想加密的原始数据是不可见的，我是说大多数网站登录所需的用户密码。还有其他的例子：例如社会安全（在加拿大的话是社会保险）号码、信用卡号码等。sha1 加密算法（很快会被升级到 sha2）是行之有效的方式。当使用此加密函数的时候，PHP 会把字符串转换成 40 个字符的十六进制数字，这是单向加密，因为没有方法可以还原这个加密（当然这就是关键点）。下面是一个例子：

117

```
$string = 'myPassword' ;
echo sha1($string) ;
```

下面是浏览器中显示的十六进制数。

5413ee24723bba2c5a6ba2d0196c78b3ee4628d1



有人曾经报道过 sha0 和 sha1 的弱点，另一个广泛使用的 MD5 算法已经不再安全，所以最好现在就使用 sha1，等到未来新算法有效时再转换成 sha2，这或许要到 2012 年。



如果你想更安全，可以使用“salt and pepper”方法，也就是说，你可以向待加密数据的前后加入硬编码的值，而后对整个字符串进行加密。以此方法硬编码过的值，攻击者将很难猜测出数据的加密算法，下面是一个例子，查看浏览器输出：

```
// 正常情形下，这个值将由一个登录表单提供
$string = 'myPassword';

$salt = "peter" ;
$pepper = "MacIntyre" ;

echo "Here is the original sha1 encryption: " . sha1($string) ;
echo "<br/>" ;
$salt = sha1 ($salt) ;
$pepper = sha1 ($pepper) ;

$string = $salt . $string . $pepper ;
echo "Here is the prepared string about to be further encrypted: " . $string ;
echo "<br/>" ;
echo "Here is the well seasoned encryption: " . sha1($string) ;
```

输出结果如下所示：

```
Here is the original sha1 encryption: 5413ee24723bba2c5a6ba2d0196c78b3ee4628d1
Here is the prepared string about to be further encrypted:
4b8373d016f277527198385ba72fda0feb5da015myPassword0f1d5a3227a9ce2b3bf67178b6ba
6e5264149a26
Here is the well seasoned encryption: 6dc853b97b4998340b28a84ba714299af1bddadf
```

## 安全技巧

### Security Tips

最后，这里列出一些 PHP 环境需要设置的地方，可以降低风险，避免提供给攻击者不必要的敏感信息。这里假设你能完全控制你的环境，如果你没有这个能力，请告诉你的主机提供商改变 *php.ini* 里的某些设置。

- 在 *php.ini* 文件里关闭 `error_display`（使用 `error_log`）。
- 确信关闭了 *php.ini* 里的 `register_globals`（详细信息参见附录）。
- 在网站需要的地方使用 SSL 认证。
- 把包含的库、SQLite 文件（如果用到的话）和配置文件，都放到非文件根目录下（这样就不能通过 Web 服务器访问它们了）。

118

如果你想更深入地了解这个主题，我强烈推荐 Chris Shiflett 的《*Essential PHP Security* (O'Reilly)》和 Ilia Alshanetsky 的《*php|architect's Guide to PHP Security*》(Marco Tabini & Associates, Inc.)，这两本书解释得更全面。





# PHP 5.3 精粹

PHP 5.3 是在 2009 年 6 月发布的。这个版本花了很长的时间，也有着非常多的改善和增强。部分增强本来是属于版本 6 的，但版本 6 看起来需要更长的时间，因此最终决定把已经完成的、本应属于版本 6 的增强先发布出来。这样也可以为版本 6 赢得更多的时间。在这一章里，我们将简单地探讨一下版本 5.3 中一些非常好的增强。

PHP 5.3 的增强非常像一个聚宝盆（我一直很想把这个词用在书里）。由于这些增强涉及了整个 PHP 产品，我们接下来会在不同的概念间做些跳跃。比如说命名空间主要影响 PHP 的面向对象特性，而 NOWDOC 则主要影响字符串的管理和显示。如果任何章节让你感到困惑，那么请参照此前的相关章节。

## 命名空间

命名空间用于在一个开发项目中避免不同对象库或者函数库间的命名冲突。你可能会有一些外来的库，也会有些你自己开发的库，而这两个或多个库之中可能会有一些命名相似的类或者函数。比如说你可能有一个生成 PDF 的库和一个生成邮件的库，但这两者都有一个名为 `Generate()` 的函数。没有一套完整的定制化编程手段（怎么才能决定变更哪个类呢？），你将会陷于混乱之中。而对这些类使用命名空间有助于避免这种混乱。命名空间这个概念和硬盘驱动器有些相似：你不能让两个名字相同的文件在同一个目录下，但你可以让同名文件存在于不同的目录下——这就是命名空间对 PHP 做的事情。实际上，处理引用命名空间的过程也和处理反斜杠（\）的方式类似（后续有更多说明）。

创建命名空间是有一些规则的，这些规则相当严格：

1. 在文件中，第一个命名空间声明之前，不能有任何代码（不可以有 HTML，不可以有空格，

什么也不能有)。不过事实上还是有两个例外。在命名空间前面可以有注释,如果你喜欢的话,在声明语句前还可以有常量声明。

2. 必须使用 namespace 关键字。

3. 命名空间必须拥有唯一的名字,并且不可以使用 PHP 关键字。

4. 一旦命名空间被使用了,所有其他代码必须包含在命名空间之内。

下面是一个例子(我包含了 PHP 标签用来展示规则 1)：

```
<?php
namespace Sample1 ;
    function sayHello($name) {
        echo "Hello there: " . $name ;
    }

    echo "This is a test <br/>" ;

    sayHello( "Johnny" ) ;
?>
```

这会产生如下的输出：

```
This is a test
Hello there: Johnny
```

乍一看,我们的例子代码似乎没遵守规则 4,但实际上一旦 Sample1 被定义了,此后的代码实际上都是这个命名空间的一部分。

也可以定义多个命名空间并有一个开放的或全局性的命名空间。这使我们必须解决许多类似命名实体的东西。一个比较好的定义命名空间的方法是使用大括号({和})把命名空间的内容括起来。基于上述这些原则我们可以重写上面的代码,结果如下所示：

```
<?php
namespace Sample1 {

    function sayHello($name) {
        echo "Hello there: " . $name ;
    }

    echo "This is a test <br/>" ;

    sayHello( "Johnny" ) ;
}
?>
```

121 这段代码的输出结果与此前的版本完全相同。建议使用大括号来定义多个命名空间。这样做对命名空间的可视化以及使代码可维护很有帮助。一旦你开始使用多个命名空间,

那么你必须定义一个全局的命名空间——规则 4 在这里再一次发挥作用。考虑这样两个代码文件：第一个文件（*chap9\_listing2.php*）有两个命名空间，并包含在第二个文件内。

第二个代码文件为代码的主体定义了一个全局的命名空间（一个没有标识符的命名空间），但是也使用（引用）了包含进来的命名空间内的代码。

```
<?php
// 以下为文件 chap9_listing2.php
namespace Sample1 {

    function sayHello($name) {
        echo "Hello there: " . $name ;
    }
}

namespace Sample2 {

function sayHello($name) {
    echo "I am happy to make your acquaintance " . $name ;
}
}

?>
<?php
// 以下为文件 chap9_listing3.php
namespace {

    function sayHello($name) {
        echo "greetings from the global scope: " . $name ;
    }

include "chap9_listing2.php" ;

Sample1\sayHello("Frank");
echo "<br/>" ;
Sample2\sayHello("Peter");

echo "<br/>" ;
use Sample1 as S1;
S1\sayHello("Frank");
echo "<br/>" ;
Sample2\sayHello("Peter");

echo "<br/>" ;
sayHello("Charlie") ;
}
?>
```

全局命名空间包含了额外的两个命名空间，当然也可以在命名空间前面加注释（请注意：注释并不是代码）。这段代码的输出结果如下所示：



```

Hello there: Frank
I am happy to make your acquaintance Peter
Hello there: Frank
I am happy to make your acquaintance Peter
greetings from the global scope: Charlie

```

如你所见，调用不同命名空间中的同名函数会导致不同的输出。引用命名空间中实体的最好方法是为实体的调用加上命名空间作为前缀，其间用反斜杠（\）分割。命名空间可以嵌在另外的命名空间中，但这样做增加了额外的复杂度。每一层的命名空间都要用反斜杠分割（\）。因此，如果 Sample1 内有另外的命名空间叫做 email\_code，同时有一个叫做 sendEmail 的函数在这个命名空间内，我们将需要像下面这样引用这个函数：

```
Sample1\email_code\sendEmail();
```

在 chap9\_listing3.php 的靠下的位置，有这样的两行代码：

```

use Sample1 as S1;
S1\sayHello( "Frank" );

```

这里我们正在使用命名空间的另一个优良特征：假名。如果你正在使用一些不由你控制的外部代码，它有一些比较长且奇怪的名字，你可以通过假名这个特征对其进行重命名。在上面的例子里我们把 Sample1 重命名为 S1，接下来调用了它的 sayHello 函数。

最后一行没使用任何前缀调用了 sayHello( "Charlie" )，这展示了如何使用全局命名空间的方法。

## 闭包（匿名函数）

另一个 PHP 5.3 的增强是闭包，也被称作匿名函数。这个概念允许你建立没有名字的函数。我们可以把匿名函数分配给一个变量，就像下面例子中做的那样（请注意分配本身符合变量分配规则，它要求以分号结束）。

```

$person_info = function($name, $age, $eyecolor)
{
    echo "greetings: " . $name . "<br/>";
    echo "You are : " . $age . " years old<br/>";
    echo "and your eye color is: " . $eyecolor . "<br/><br/>";
};

$person_info( 'Peter' , '43' , 'brown' );
$person_info( 'Dawn' , '15' , 'green' );

```

我们把一个匿名函数（闭包）分配给了一个叫做 \$person\_info 的变量，这个函数有三

个参数：\$name、\$age 和 \$eyecolor。接下来我们就可以引用这个变量，并向它传值，几乎像调用实际的函数一样。

PHP 的文档指出，闭包在以回调函数遍历数组时和用一个函数逐一处理数组中的每一个元素时是非常有帮助的。下面是一个简单的例子：

```
$person_info = function($value, $key)
{
    echo $key . ": " . $value . "<br/>" ;
};

$person_array = array( 'name' => 'Dawn', 'age' => '15', 'eyecolor' => 'green' );
array_walk($person_array, $person_info);
```

在这个例子里，我们传递一个变量给 array\_walk 函数作为它的第二个参数，而不是传递一个函数的引用。

## NOWDOC

下一个我将为大家说明的 5.3 版的一个增强被称作 NOWDOC。所谓的 NOWDOC 是 HEREDOC 这一字符串管理功能的一个变种。如果你还记得，HEREDOC 可以像一般的在双引号之间的字符串一样工作，只是它允许在字符串中包含可变的内容，这部分可变的内容将在字符串内部被处理。NOWDOC 的行为则与单引号之中的字符串相同，由于使用了单引号，在单引号之中并不处理可变的内容（不能插值）。下面的例子用来说明 HEREDOC 和 NOWDOC 分别可以做些什么：

```
$myname = "Peter MacIntyre" ;

$str = <<<" HEREDOC_Example"
Lorem ipsum dolor sit amet,
nibh euismod tincidunt $myname .
HEREDOC_Example;

echo $str ;

$str2 = <<<' NOWDOC_Example'
Lorem ipsum dolor sit amet,
nibh euismod tincidunt $myname .
NOWDOC_Example;

echo "<br/>" . $str2 ;
```

下面是在浏览器中的输出结果：

Lorem ipsum dolor sit amet, nibh euismod tincidunt Peter MacIntyre.  
Lorem ipsum dolor sit amet, nibh euismod tincidunt \$myname.

- 124 NOWDOC 结构适用于构造比较长的且不需要对字符和变量进行转义的文本，比如标准邮件的主体或者报告中的声明文本。

## goto 操作符

如果你曾经用过 Basic 或者 Visual Basic，那你对那些语言中的 goto 语句应该已经有了足够的理解。当讨论是否应该加入这一特征时，在 PHP 开发队伍内产生了非常大的争议，看起来同意加入这一特征的一方最终胜出。这个操作符允许你从一个代码文件中跳到另外的位置。这是一个非常强大的增强，因此我在这里介绍它。但显然如果它不被正确使用，也是一个很糟糕的部分。正是因为这个原因，在附录中有一个专门关于 goto 的小节。

在使用 goto 语句时，同时存在一些限制：

- 无法跳转到其他的代码文件里。
- 不能跳出或跳进一个函数或方法（你只能在同一个域内跳转）。
- 不能跳进像 while 或者 switch 的循环结构，但可以从中跳出来。

为了定义 goto 的目的地，要用一个标签（可以是任何字母或数字，但一定以字母开始），接下来跟着一个分号。为了激活跳转，需要在 goto 后加上标签名。下面是一个例子：

```
$letter = "A" ;  
if ($letter == "A" ) {  
    goto landing_area_a;  
} else {  
    goto landing_area_b;  
}  
  
landing_area_a:  
echo 'The Eagle has landed<br/>' ;  
  
landing_area_b:  
echo 'The Hawk has landed<br/>' ;
```

浏览器上的输出可能和你期望的不一样：

```
The Eagle has landed  
The Hawk has landed
```

导致所有注释都输出到浏览器的原因是：一旦你到达 goto 处，代码就会向前跑，最终下一个标签 landing\_area\_b: 会被忽略掉，但是回显语句由于是在后面所以会被执行。一个

125

处理这类问题的办法是加入另一个 goto 语句, 这个 goto 将跳过剩下的你不想执行的代码。

```
landing_area_a:
echo 'The Eagle has landed<br/>' ;
goto lands_end;

landing_area_b:
echo 'The Hawk has landed<br/>' ;

lands_end:
```

当然, 你很快就会看到为什么针对是否把这个特征加入 PHP 5.3 有如此之多的争论。借助 goto 语句有可能会写出“意大利面条式代码”, 这类代码在文件中进行全局性跳转, 会使代码非常难于维护。

另一项警告是使用 goto 语句存在导致无限循环的潜在可能。请看下面的代码并试着跟踪它。我不建议你在这段代码载入你的 PHP 环境, 除非你知道如何终止它。

```
starting_point:
$letter = "A" ;
if ($letter == "A" ) {
    goto landing_area_a;
} else {
    goto landing_area_b;
}

landing_area_a:
echo 'The Eagle has landed<br/>' ;
goto lands_end;

landing_area_b:
echo 'The Hawk has landed<br/>' ;

lands_end:
goto starting_point;
```

下面是浏览器中开始几行的输出结果：

```
The Eagle has landed
The Eagle has landed
The Eagle has landed
The Eagle has landed
The Eagle has landed
The Eagle has landed
```

正如你所见, 这就是一旦粗心将会由 goto 操作符所导致的无限死循环。如果这事真的发生, 并且你使用鼠标的速度足够快, 那么你可以在网页服务器溢出前把浏览器停掉。





不论你怎么做，一定要努力避免写下这样两行代码（我实际上在几年前看到一个无能的程序员用大型机语言写下如下的代码）：

```
landing15:
goto landing15;
```

这是最紧凑的 goto 循环，同时也是最难定位的。从这个例子可以看到，PHP 给了你足够绳子，但要不要用这些绳子勒死自己则完全取决于你自己。毕竟你已经被警告过了！

## DateTime 和 DateTimeZone 类

PHP 开发者应该对日期和时间函数保持足够的小心，这些函数通常用来完成日期相关的任务，比如为数据库相关记录加入时间戳或者计算两个日期期间的差异。自从 5.2 版本开始，PHP 就提供了一个 DateTime 类来同时解决大部分日期和时间相关的信息，也就是说你无须使用许多 date 和 time 函数。新的 DateTimeZone 也能够完成同样的功能。我们之所以在这一章里关注这个类，是因为它毕竟是一个 PHP 中的新东西，尽管从严格意义上这不能说是 5.3 版的一个新特征。

这些年来随着门户网站和社交网络的发展，时区管理变得越来越重要。如今，向一个站点发送一些信息，同时让同一个站点的人认识到你是在世界的哪个角落是一个不言而喻的需求。但请记住，像 date() 这样的函数将从运行脚本的服务器端获得默认的信息，因此除非客户端的人告诉服务器究竟她在那里，否则自动决定时区位置是非常困难的一件事。

总计有 4 个与日期和时间相关的类，我们在这次讨论中将关注其中三个。所有这一切都从 DateTime 的构造函数开始。这一方法需要两个参数：时间戳和时区。下面是这个构造函数的语法：

```
$dt = new DateTime("2010-02-12 16:42:33", new DateTimeZone('America/
Halifax'));
```

这行代码创建了 \$dt 实例，并且把一个日期和时间的字符串作为第一个参数传给了它，通过第二个参数设置了时区。第二个参数实例化了一个 DateTimeZone 对象，接下来把它作为第二个参数传给了构造函数。你也可以先实例化一个 DateTimeZone 的变量，接下来在 DateTime 构造函数中使用它。

```
$dtz = new DateTimeZone('America/Halifax') ;
$dt = new DateTime("2010-02-12 16:42:33", $dtz);
```

显然我们正在对这些类进行硬编码，对于你的代码这种类型的信息也许并不总是有效，或者这并不是你想要的。为此，我们也可以从服务器的 .ini 文件中获取时区信息，接下来在 DateTimeZone 类中使用它。下面的代码可以用来获取服务器 ini 文件中的值：

pin5i.com  
最新编程教材  
站拼吾爱  
把这些信息用

会把这些信息用

•

•

•

•

January 到 December]

格式化字符	描述
年	
o	ISO-8601 所定义年数，值与 Y 所定义的一致，但如果 ISO 的周数 (W) 属于上一年的话，则表示上一年的年数 (PHP 5.1.0 导入) [例：1999 或 2003]
Y	一年的完整数字表示，四位数 [例：1999 或 2003]
y	年的两位数表示 [例：99 或 03]
时间	
a	小写的上下午 [am 或者 pm]
A	大写的上下午 [AM 或 PM]
B	Swatch 的 Internet 时间 [从 000 到 999]
g	12 小时格式的小时数，数字前不补零。
G	24 小时格式的小时数，数字前不补零。
h	12 小时格式的小时数，数字前补零。
H	24 小时格式的小时数，数字前补零。
i	分钟并补零 [00 到 59]
s	秒并补零 [00 到 59]
u	微秒 (PHP 5.2.2 导入) [例：654321]
时区	
e	时区标识符 (在 PHP 5.1.0 中导入) [例：UTC、GMT、Atlantic/Azores]
l	(大写的 i) 标识是否是夏令时 [1 表示夏令时, 0 表示不是]
O	同格林威治时间 (GMT) 偏差的小时数 [例：+0200]
P	同格林威治时间偏差的小时和分钟，中间用分号隔开 (PHP 5.1.3 中导入) [例：+02:00]
T	时区缩写 [例：EST、MDT]
Z	用秒表示的时区偏移。对于 UTC 西部的时区，偏移总是负值。UTC 东部的时区偏移值总是正值 [-43200 到 50400]
完整的日期和时间	
c	ISO-8601 的日期 (PHP 5 中导入) [2004-02-12T15:19:21+00:00]
r	按照 RFC 2822 格式化的日期 [例：Thu, 21 Dec 2000 16:01:07 +0200]
U	自从 UNIX 时代 (January 1 1970 00:00:00 GMT) 开始以来的秒数 [请参照 time()]

到现在为止，我们已经为构造函数提供了日期和时间，但有的时候你可能会想从服务器得到日期和时间的值。为达成这一目标需要提供字符串“now”作为第一个参数。下面的代码与前面的例子所做的事情相同，但是它将从服务器获得日期和时间。事实上正由



于它是从服务器获得时间，这个类的应用将会更加广泛：

```
$timeZone = ini_get('date.timezone') ;
$dtz = new DateTimeZone($timeZone) ;
$dt = new DateTime("now", $dtz);

echo "date:" . $dt->format("Y-m-d h:i:s");
```

浏览器会显示服务器上返回的值：

```
date: 2010-02-18 12:38:14
```

DateTime 的 diff 方法会像你期待那样动作，它会返回两个日期的差。这里的关键点是，一个 DateInterval 类会被实例化，并作为 diff 方法的结果返回。DateInterval 这个类也有一个 format 方法，但由于这个类是用来处理两个日期的偏差，格式存在差异。格式信息请参照表 10-2。

表10-2 DateInterval类的格式字符

字符描述	
每个格式字符	
必须以 % 开头	格式
%	百分号 %
Y	年数，一个数字，至少两位，如果是一位那么补零 [01,03]
y	年数，数字 [1,3]
M	月数，一个数字，至少两位，如果是一位那么补零 [01,03,12]
m	月数，数字 [1,3,12]
D	天数，一个数字，至少两位，如果是一位那么补零 [01,03,31]
d	天数，数字 [1,3,31]
a	天的总数 [4,18,8123]
H	小时数，数字，至少两位，如果是一位那么补零 [01,03,23]
h	小时数，数字 [1,3,23]
I	分钟数，数字，至少两位，如果是一位那么补零 [01,03,23]
i	分钟数，数字，[1,3,59]
S	秒数，数字，至少两位，如果是一位那么补零 [01,03,23]
s	秒数，数字 [1,3,57]
R	负值时用“-”，正值时用“+” [-,+]
r	负值时用“-”，正值时为空白 [-,+]

因此为了获得两个日期间的差异（更准确地说是两个 DateTime 对象），我们会写出下面的代码：

```
$timeZone = ini_get('date.timezone') ;
```



```
$dtz = new DateTimeZone($timeZone) ;
```

DateTime and DateTimeZone Classes | 129

```
$start_dt = new DateTime("2009-02-12 16:42:33", $dtz);
$dt = new DateTime("now", $dtz);
```

```
// 建立一个新的 TimeInterval 实例
```

```
$dt_diff = $start_dt->diff($dt) ;
```

```
echo "<br/><br/>The difference between: " . $start_dt->format("Y-m-d")
. " and " . $dt->format("Y-m-d") . " is"
. $dt_diff->format('%y year, %m months, and %d days');
```

一个 DateTime 对象的 Diff 方法被传入了另一个 DateTime 对象并进行调用。接下来我们用 format 方法来准备浏览器中的输出。

现在让我们更仔细一点来观察 DateTimeZone 类。你可以用我们曾经看到过的 get\_ini 函数从 *php.ini* 中获取时区设置。你也可以通过 getLocation 方法获取时区以外的信息。它可以提供时区所在的国家、经纬度以及一些说明。使用下列很少的代码，你可以开始一个简单的 GPS 系统：

```
$timeZone = ini_get('date.timezone') ;
$dtz = new DateTimeZone($timeZone) ;

echo "Server's Time Zone: " . $timeZone . "<br/>";

foreach ( $dtz->getLocation() As $key => $value ){
    echo $key . " " . $value . "<br/>";
}
```

这会在浏览器中产生下面的输出：

```
Server's Time Zone: America/Halifax
country_code CA
latitude 44.65
longitude 62.4
comments Atlantic Time - Nova Scotia (most places), PEI
```

如果你想设置与 Server 不同的时区，那么把那个值传给对象的构造函数。下面我们把时区设置为意大利的罗马，并把信息用 getLocation 进行显示。请注意我们选择了用 getName 方法获得时区（尽管在之前的代码中我们对其进行手动设置）：

```
$dtz = new DateTimeZone("Europe/Rome") ;

echo "Time Zone: " . $dtz->getName() . "<br/>";

foreach ( $dtz->getLocation() As $key => $value ){
    echo $key . " " . $value . "<br/>";
}
```



你可以在 <http://www.php.net/manual/en/timezones.php> 上找到有效时区列表。

131

类里面提供了强大的日期和时间处理能力，我们所讨论的不过是冰山一角。确保你自己了解了 PHP 站点上的这些类，也知道它们究竟可以干什么。

## 额外的5.3特征

事实上有大量的改善伴随着 5.3 版本。确保你自己去 <http://www.php.net/ChangeLog-5.php> 检查过完整的 Bug 列表和增强列表。



# 高级优势

到目前为止，这本书尝试描绘出 PHP 语言最好的一面。我们已经覆盖了 PHP 大部分基本概念，也讨论了每一章节下的精华部分。在这一章里，我们将看到 PHP 中更为高级的特征，并会以一些有用的参照和资源的汇总作为结束。基于它们，如果你想，你将可以在 PHP 的世界中走的得更远。

## 正则表达式

我们所关注的第一个高级特征是正则表达式。正则表达式提供了一种非常高级的在字符串中匹配模式的方法。尽管 PHP 中提供了很多的字符串函数，但仍然有些你想完成的任务只有通过正则表达式才能完成。有两种类型的正则表达式：POSIX(可移植性操作系统接口)和 Perl 兼容。由于 Perl 兼容的正则表达式更快一点，也更健壮，在这里我们将只对这类正则表达式进行考查。正则表达式有三个常规用途：字符串匹配、字符串替换和字符串切分。

## 字符串匹配

让我们先看一下字符串匹配。当你在一个给定字符串中查找确定的字符串或模式时，你需要限定模式。通常来讲会用到斜杠 (/)，但你可以使用任何其他的非字母和数字字符来扮演同样的角色(但不能是反斜杠)。因此，如果你在寻找“fox”这一模式，你可以把它设置成 /fox/ 或 #fox#，当然只要模式的开始和结束字符相同，别的字符也可以，但这个字符不能是你正在查找的模式的一部分(如果你的模式是 fox#y，你就需要使用 /fox#y/ 或者 {fox#y}，但一定不能是 #fox#y#)。用来进行匹配的函数是 preg\_match:



这个函数如果找到了一个结果，那么返回 1(由于一旦找到一个匹配，搜索将停止)，如果什么也没找到，那么返回 0(如果发生错误，那么返回 false)。

134 我们将使用“the quick brown fox jumps over the lazy dog.”来做一些模式匹配，下面是第一个例子：

```
$string = "The quick brown fox jumps over the lazy dog";  
echo preg_match('/fox/', $string); // returns 1
```

此处我们在所提供的字符串中对“fox”进行全局性查找。我们也可以用基本的字符串函数完成这一功能，但更复杂的部分仍然还没有开始。preg\_match 函数支持一些模式量词。表 11-1 中列出了一些常用的模式量词和对应的例子。

表11-1 preg\_match表达式的模式量词

- ^ 在字符串开始处测试模式
- \$ 在字符串结尾处测试模式
- .
- \ 通用转义字符，当把其他量词作为正常字符串搜索时使用
- [ ] 声明有效字符的范围：[0-5] 意味着 0 和 5 之间
- { } 用来表示在前一个模式下允许多少个字符

使用这些量词，我们可以更精确地查找我们要找的东西，也可以更精确地定位查找地点。下面是一些例子：

```
echo preg_match('/^fox/', $string); // returns 0  
echo preg_match('/^The/', $string); // returns 1  
echo preg_match('/^the/', $string); // returns 0  
echo preg_match('/dog$/ ', $string); // returns 1  
echo preg_match('/f.x/', $string); // returns 1
```

我们的最后一个例子更加复杂：我们将试着在 999-999-9999 这样的模式中检验一个包含了区号的北美电话号码。我们将会在 2 到 9 之间查找第一个字符(北美的电话不以 1 开头)，接下来的两位处于 0~9 之间。接下来是一个横线和 3 个处于 0~9 之间的字符。再接下来是另一个横线和 4 个 0 到 9 之间的字符。这听起来有点复杂，是一般的字符串函数不能有效处理的情形。下面是为我们完成这一功能的代码：

```
$phone_num = "903-543-5454";  
$pattern = "/^[2-9]{1}[0-9]{2}-[0-9]{3}-[0-9]{4}$/";  
  
echo preg_match($pattern, $phone_num); // 返回 true
```

请注意我们可以把模式放入一个变量，这样就可以重用它。比如说：我们想测试宅电、传真、手机和工作的电话，那么我们可以重用这个模式。

## 字符串替换

你可以用 `preg_replace` 来完成模式的查找和替换，模式相关的定义与上述相同。下面是一个使用 “quick brown fox” 字符串，并把 “fox” 替换成 “cheetah” 的例子。

```
$string = "The quick brown fox jumps over the lazy dog" ;
echo preg_replace('/fox/', 'cheetah', $string) ;
```

浏览器上的输出结果是：

```
The quick brown cheetah jumps over the lazy dog
```

作为一个高级点的例子，我们来看一下那个电话号码变量，并把所有的 4 替换成 7。

```
$phone_num = "903-543-5454";
echo preg_replace('/4/', '7', $phone_num) ;
```

这看起来很容易，你可能想要同时替换另一个部分来提高效率。为达成这一目的，仅需要把替换的对应关系分别放入两个数组，就像下面这样：

```
$look_for = array("/4/", "/-/");
$replace_with = array('7', '*');
echo preg_replace($look_for, $replace_with, $phone_num) ;
```

在这里我们试图把所有的 4 替换成 7，并把所有的 - 替换成 \*。这会产生下面的输出结果：

```
903*573*5757
```

## 字符串分割

另一个你可以使用正则表达式来干的事情是通过周边的特征，解析出具体的信息。为做这件事情需要使用 `preg_split` 函数。这个函数需要两类信息：你想查找的内容的模式和待查找的字符串，最终这个函数会返回不在模式里的信息。这让人有点困惑，我们来看一个例子。在这里我们正在查找一个数学公式中的操作符（我们的模式），但我们只希望得到操作数。`preg_split` 函数会返回操作数所处的位置：

```
$pattern = "#[+*/-]#" ;
$formula = "36+15/5*12" ;
$operands = preg_split($pattern, $formula) ;
var_dump($operands) ;
```

方括号告诉我们要把方括号中的每一个项目都看做是单个的待查找项目，又由于通常的

模式分隔符 (/) 是待查找项目中的一个, 所以必须使用 #。最终输出结果是:

136

```
array(4) { [0]=> string(2) "36" [1]=> string(2) "15" [2]=> string(1) "5" [3]=>
string(2) "12" }
```

正则表达式可以非常强大非常复杂。踩到这个圈子里的时候要小心, 一旦你不能使用一些简单的 PHP 函数达成你的目的, 也就意味着你需要更深入地理解这些函数。关于进一步的帮助和更复杂的例子, 请参照 *php.net* 上的网页。

## SimpleXML

在 Web 上, XML 每天都在收获越来越多的支持。在以前, 不同系统间用的信息共享方法是另外的方法, 但现在 XML 已经成为数据共享中的领跑者。PHP 提供一些不同的方法来处理 XML 文档, 在这一节里, 我们将看一下 SimpleXML 库 (默认与 PHP 一起安装)。它对于管理不复杂的 XML 文档很有帮助。也就是说为了管理不复杂的 XML 文档, 你必须知道目标 XML 的主要元素, 这同时也是为了了解数据本身和究竟怎么精准正确地处理数据, 考虑下面已经格式化好的图书馆中的包含了 3 本书的 XML 文档:

```
<?xml version="1.0" ?>
<library>
  <book>
    <title>Programming PHP, 2nd Ed.</title>
    <isbn>0-596-00681-0</isbn>
    <authors>
      <author>Rasmus Lerdorf</author>
      <author>Kevin Tatro</author>
      <author>Peter MacIntyre</author>
    </authors>
    <publisher>OReilly</publisher>
    <price>39.99</price>
    <pubdate>April, 2006</pubdate>
  </book>
  <book>
    <title>PHP: The Good Parts</title>
    <isbn>978-0596804374</isbn>
    <authors>
      <author>Peter MacIntyre</author>
    </authors>
    <publisher>OReilly</publisher>
    <price>37.99</price>
    <pubdate>March, 2010</pubdate>
  </book>
  <book>
```



```
<title>JavaScript: The Good Parts</title>
<isbn>978-0-596-51774-8</isbn>
<authors>
    <author>Douglas Crockford</author>
</authors>
<publisher>OReilly</publisher>
<price>29.99</price>
<pubdate>May, 2008</pubdate>
</book>
</library>
```

XML 字符串将被保存到一个叫做 `$xml_data` 的变量里，接下来会被交给 SimpleXML 进行处理。

大多时候你是从硬盘或者从外部的某个源得到一个文件，但这次由于我们的目的，我们会自己建立这个 XML 的内容。这将影响你在最初的时候究竟怎么处理 XML 文件，并导致区别。SimpleXML 是基于对象的，因此我们可以使用第一行代码 (`$xml_doc = new SimpleXMLElement($xml_data);`) 来把待处理的 XML 文件作为一个对象进行处理。这个类的构造函数会把 XML 转为多维数组。当我们处理数组的时候，可以使用 `foreach` 来帮助我们浏览这个结构。当我们对 XML 的层级进行遍历的时候，可以按照我们认为合适的方式处理相关数据。在下面的代码中，我们把数据显示在浏览器中，同时检测每一个层级下是否有需要列在一起的多个作者：

```
$xml_doc = new SimpleXMLElement($xml_data);
foreach ($xml_doc as $book) {
    foreach ($book as $key=>$value) {
        if ($key == "authors") {
            echo $key . " = " ;
            foreach ($value as $author=>$detail) {
                echo $detail . ", ";
            }
        } else {
            echo $key . " = " . $value;
        }
        echo "<br/><br/>" ;
    }
}
```

这是基本的 XML 处理过程，并不怎么好看，但当对 XML 所有节点进行遍历时，你几乎可以按照这种过程做任何事情上面的代码产生的原始输出如下所示：

```
title = Programming PHP, 2nd Ed.
isbn = 0-596-00681-0
authors = Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre,
```



```

publisher = OReilly
price = 39.99
pubdate = April, 2006
title = PHP The Good Parts
SimpleXML | 137
isbn = 978-0596804374
authors = Peter MacIntyre,
publisher = OReilly
price = 37.99
pubdate = March, 2010
title = Javascript: The Good Parts
isbn = 978-0-596-51774-8
authors = Douglas Crockford,
publisher = OReilly
price = 29.99
pubdate = May, 2008

```

如果你正在从磁盘读入一个文件，可以使用下面的代码检验它是否存在，接下来把它用 `simplexml_load_file` 函数读进来，以待进一步处理。两个存取方法都建立相同的对象来让 PHP 处理，因此你可以使用前面的 `foreach` 例子代码：

```

if (file_exists('library.xml')) {
    $xml = simplexml_load_file('library.xml');
} else {
    // 处理不存在的文件
}

```



有两个其他的 SimpleXML 函数能帮助你把数据读进来以待处理，这与 XML 数据源有关。`simplexml_load_string` 和我们上述例子中使用方法类似：`new SimpleXMLElement ($xml_data);` 读入 XML 数据中的字符串以待处理。另外的 `simplexml_import_dom` 函数则从 DOM 节点读入 XML。

## 集成开发环境

下面我们切换一下话题，讨论一下市场上最好的 PHP 集成开发环境。好的集成开发环境很多，其中有一些是比较高级的文本编辑器，讨论时会忽略这些。排在前面的集成开发环境是 Komodo、Zend Studio for Eclipse 以及 PhpED。它们每一个都有专门的长处，因此在你开发时请确保结合你想干什么对每一个都进行评估。下面是对各个集成开发环境主要特征的简单描述。



这并不意味着推荐这里所列出的任何一款产品而排斥其他在市场上也很好的 IDE。我们想做的仅仅是帮你体会一下市场上究竟都有些什么。

### ActiveState的Komodo

Komodo IDE by ActiveState 是一个支持多种开发语言的开发环境，也就是说你可以使用它来开发 PHP、Python、Ruby、ASP、JavaScript 等。这是它最大的一个卖点，因此也导致它主要被需要维护许多不同 Web 环境的程序员所使用。下面是 Active State 对它自己产品的描述：

**语言无关：**Komodo IDE 支持 PHP、Python、Ruby、Perl、TCL、JavaScript、CSS、HTML 以及一些模板语言如 RHTML、Template-Toolkit、HTML-Smarty、Django。

**平台无关：**买一个许可就可以在各种平台下运行。

**更聪明地工作，而不是更辛苦：**Komodo IDE 中广获好评的功能包括基本的编辑功能、语法检查、颜色标识、正则表达式、调试工具等。同时它是可扩展的，你可以对其进行定制。

**共享工作负荷：**团队开发会在集成源代码控制、项目管理、多用户支持的帮助下运行得更快。

### 用于Eclipse的Zend Studio

Zend Studio for Eclipse 是一个非常强大和健壮的 IDE。它同 Zend 的其他工具集成得很好，如果你想使用 Zend 的全线工具的话，那么这是一个很大的优势。由于它是基于 Eclipse 平台的，因此可以使用许多 Eclipse 的插件产品，而并不只是局限于 Zend 的产品。团队开发可以使用 SVN 或者 CVS 进行集成。代码高亮显示、语法检查、健壮的调试、代码

概要分析、任务管理是它许多特征中的一部分。下面是 Zend 对它自己产品的宣传：

Zend Studio 7.0 是下一代专业级的 PHP 开发环境。通过使代码的开发和维护更快，迅速解决应用的问题，增强团队合作，它可以最大化你的开发生产率。

Zend Studio 是唯一为专业开发设计并支持 PHP 应用程序生命周期所需要的所有开发组件的 IDE。通过全面的功能集合：编辑、调试、分析、优化、数据库工具和测试等，Zend Studio 将缩短开发周期，简化复杂项目。

## NuSphere的PhpED

对于中小规模项目来说，PhpED 是一个很好的编辑器。它总计有三个版本，当然有效的特征越多，价钱越高。尽管只在 Windows 下有用，它仍然是一个很通用的编辑器，内建了代码高亮及代码文件夹等功能。此外，它是以项目为基础的，也就是说你可以同时管理处在不同编码阶段的多个项目。PhpED 通过 CVS 支持团队开发，但不支持 SVN 做代码管理。下面是 NuSphere 在其主页上的说明信息：

PhpED 是一款帮助 PHP 开发者利用最新技术的 IDE。PhpED 5.9 版支持 PHP 5.3 的所有特征，包括名字空间、对象引用、闭包、类级常量（关键字常量）、now-doc 等。PhpED 5.9 在 5.8 后推出，在我们的开发工具列表里它是支持 Web 2.0 的产品。通过 PhpED，你的 PHP IDE 将在各种层面上支持 PHP 5.3 的特征，如动态语法高亮分析器、代码自动完成、PHP 调试器。Php IDE 对以前版本 PHP 的支持也得到了增强。

## 主要网站

作为引用的另一方面，我将分享自己定期访问的主要站点。你将会发现它们对于你的开发、你的问题解决、你的代码调试是非常好的资源。这里有着海量的 PHP 网站，但我希望能通过这几页缩减你使用 Google 的时间。

### php.net

<http://www.php.net> 是 PHP 语言的大本营，在这里你可以找到发布了的语言文件及引用库的维护（包含使用语法和例子）。这个站点的一个极好的特征是它允许用户添加额外的建议。



这就使开发社区可以贡献最佳实践、经验教训及各个函数相关的陷阱。它也包含了每月举行的小组会议的信息。如果你能够找到离你近的 PHP 小组,你应该尽量参加它的一些工作。

这是一种很好地了解你周围 PHP 开发者的方法,也可以知道在你家附近 PHP 上正在发生什么。

图 11-1 展示了 php.net 在笔者写这本书的时候主页的截图。

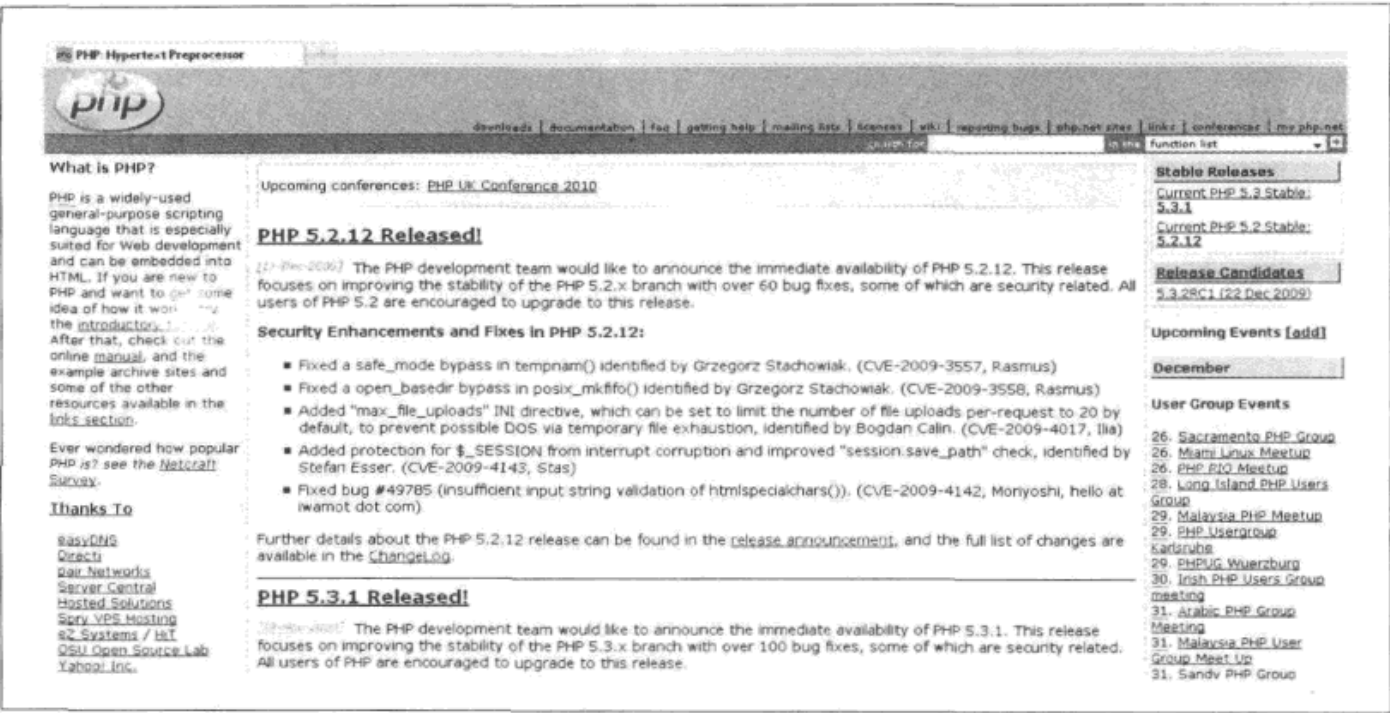


图11-1 php.net的主页

## zend.com

<http://www.zend.com> 是 Zend 公司的主页。它体现了 PHP 商业化的一面, PHP 公司有许多待售的产品 (包括前面讨论的 IDE), 包括培训课程、支持服务等。PHP 在网页开发的世界里获得越来越多的份额的同时, 在这里你可以获得所有最新的 Zend 的新闻和开发状态。也可以购买 Zend 培训和考试, 还可以搜索 PHP 黄页来确认通过了 Zend PHP 考试的名单。图 11-2 展示了 zend.com 在笔者写这本书的时候主页的截图。

## devzone.zend.com

另外一个很好的资源是由 Zend 运营的 DevZone 网站。这里是对 Zend 所有产品提供技术支持和实践的地方。尽管网站的副标题是“推进 PHP 的艺术”, 实际上它包含了 Zend 各



个产品线相关的资源(文章、教程、播客、书评、网络研讨会、论坛)。这里是一个查找 PHP 相关文章及白皮书的好地方。在这里也可以找到 PHP 的在线手册。图 11-3 展示了 Zend DevZone 在笔者写这本书的时候主页的截图。

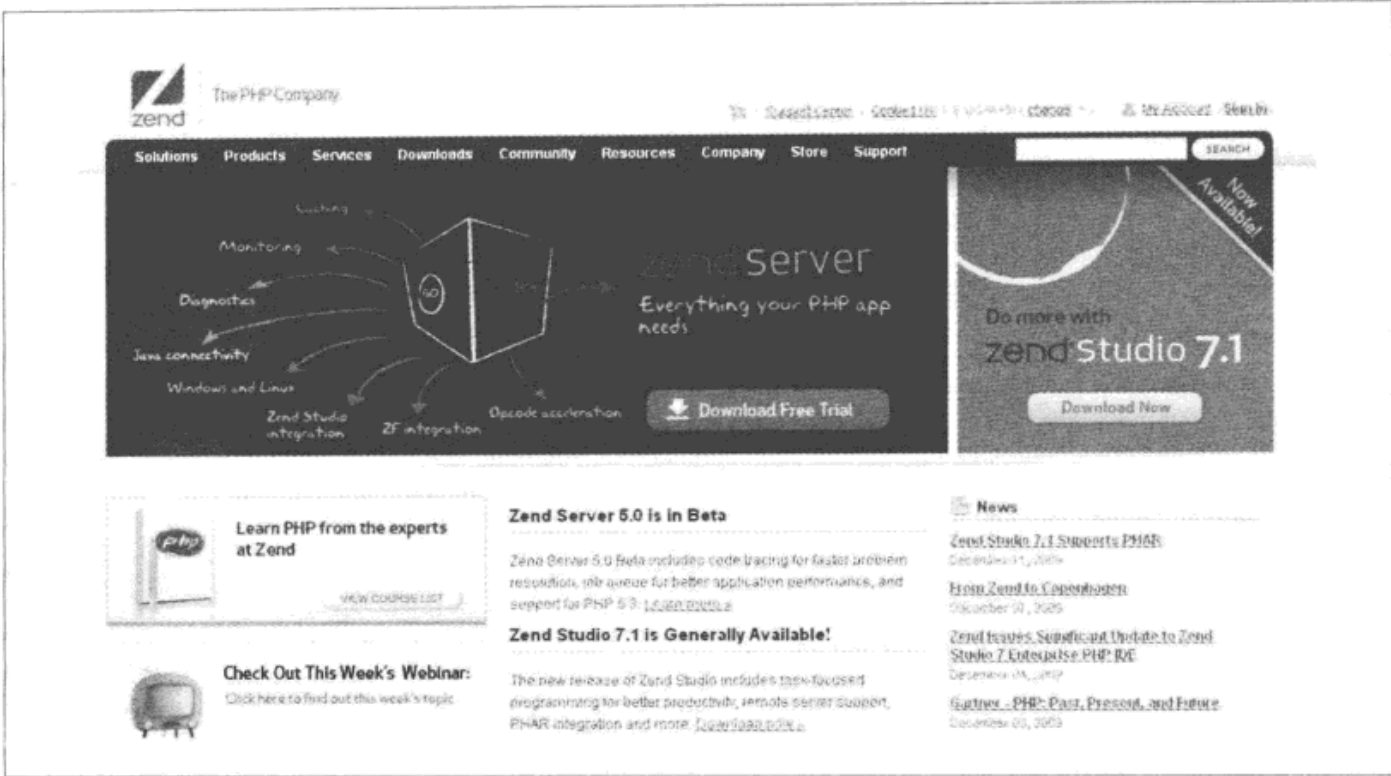


图11-2 zend.com 主页



图11-3 devzone.zend.com 主页

最后一个一定要有的书签是 *php|architect*。这是一个旨在提高 PHP 架构的杂志，它同时出纸版和 PDF 版。这是一个出版多年的超级技术杂志。它完全开放：我是这本杂志的编辑，因此可能会有偏见，但我还是要说它质量很高，内容很优秀。除了出版杂志，运行它的组织每年也召开两次 PHP 会议。这两次会议非常值得参加，是接触 PHP 社区众多人员的好办法。让我们回到网站，在这里除了可以找到优秀的书籍、播客、培训材料外，也有在线的新闻贴，通过它你可以知道 PHP 世界里所有的最新消息。图 11-4 展示了 *phparch.com* 在笔者写这本书的时候主页的截图。

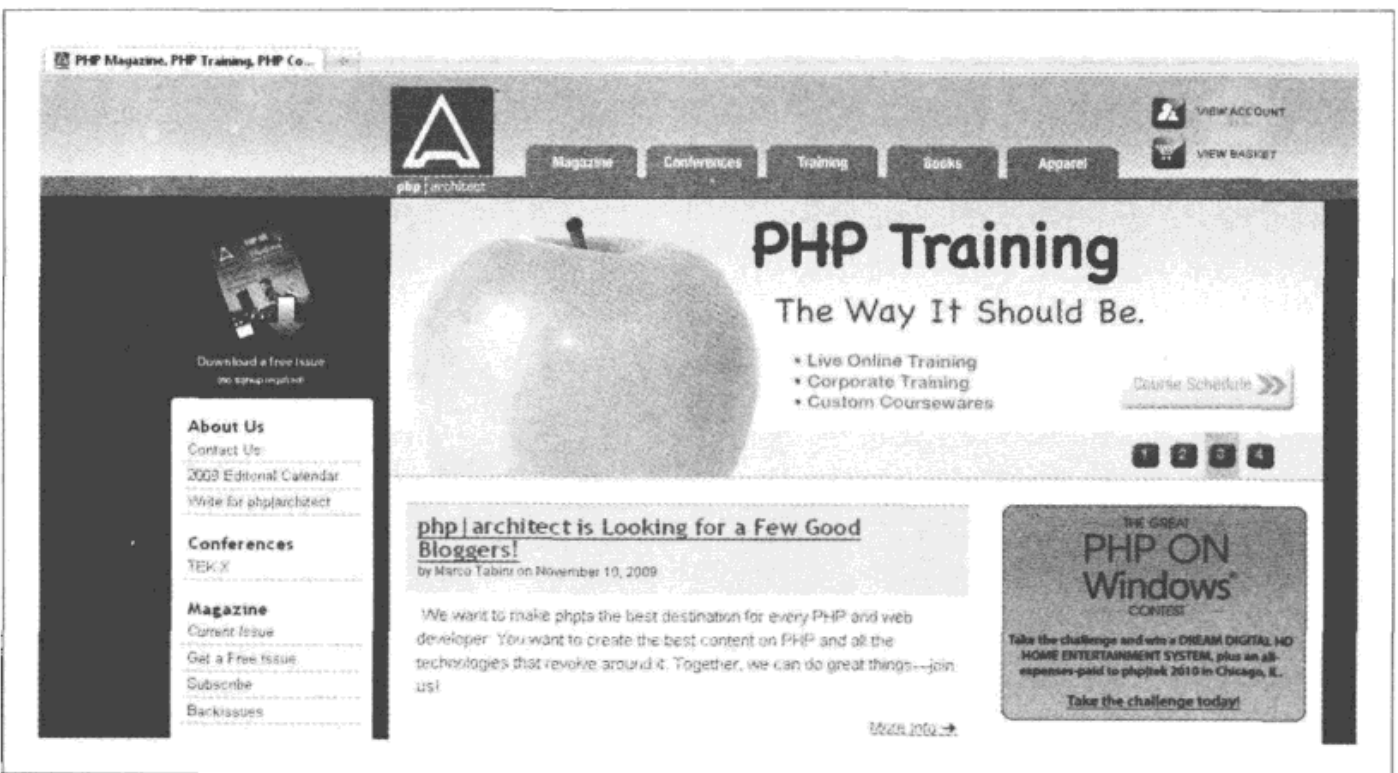


图11-4 phparch.com主页

## PHP/Web 会议

每一年世界范围内举行很多 PHP 和 Web 的会议。除了上面提到的（由 *php|architect* 主持的）会议之外，另一个主要会议是每年秋天在加利福尼亚州由 Zend 公司主持的 ZendCon。也有许多会议在欧洲（英格兰、西班牙和德国）、南美（里约）和加拿大（PHP 魁北克）召开，很值得参加。

144 确定会议的最好方法是检查会议列表页。在这里你可以找到会议什么时候开始以及是否开放议题。你可以随意提交议题，同时你也总是可以从多数发言者那里听到新的和有意思的主意。

在网页、播客和书籍上有着很多其他的 PHP 资源。花点时间浏览下上述提供的网站，使用你喜欢的搜索引擎来帮助你找到更多的资源。本着开源的哲学，请记得贡献你所找到的好东西。

PHP 有缺点这一事实很难被理解，毕竟这是一个世界上使用最为广泛的开发语言。NASA、Wikipedia、Yahoo! 和 IBM 以及其他一些公司每天都在使用它做关键性的数据处理和网页开发。事实上，我曾经的观点是 PHP 并没有真正意义上的缺点，只是有一些工作时必须注意的小陷阱。

然而经过一些深刻反省，我逐渐认识到 PHP 并不完美——完美究竟应该是什么样子？它由人类所创建（人类本身就是不完美的），更新的版本不断被推出（包含一些错误修正）。话虽如此，我们仍将花几页来看看 PHP 的弱点（或者说可感受到的弱点），以及如何与这些弱点一起工作或规避这些弱点。

## goto

首先要讨论的是把 goto 语句包含到 5.3 版本里面这件事。在我来看，goto 应该只被那些有足够经验并能够避免无限循环的人使用。这时你可能会想到第 10 章，现实中确实有大量的你可能会犯的编码错误。没有什么能真的保证你不写像下面这样的代码：

```
landing15:
goto landing15;
```



实际上，PHP 在 .ini 设置中有一个选项用于停止运行时间过长的脚本，默认设置时间为 30 秒——这个选项的名字是 max\_execution\_time。如果时限被超出，那么错误的脚本会被强行终止。因此并不会损害你的服务器（但无限循环始终是应该避免的）。

这实际上是 PHP 潜在的弱点，但是仅当你犯错误的时候它才出现。这并不真是 PHP



的过错，我们正在关注的是代码编写的领域，这与技能和程序员的逻辑有更强关系。PHP 给了你各种绳子，但只有作为开发者的你自己能决定不用这绳子勒死你自己（或别人）。

## 函数名与参数顺序

你可能还记得，PHP 是一个开源产品。这意味着它由世界上许多程序员共同完成。这导致了許多文化和语言对这个项目产生了影响。这可能会导致较多的混乱，但是平衡和控制在大地方都有，同时 Zend 正在盯着这件事情。然而即使如此，PHP 语言中的许多实例的命名规则并没有保持一致。比如，你会看到有些内部函数在命名时使用了下划线，像 `var_dump` 和 `strip_tags`，然而另外一些则是连在一起的，比如 `stripslashes` 和 `strpos`。这会成为一个很令人困惑的事情，因为你无疑会被强迫着查看函数名来确认它们的精确语法，这并不是一个偶而会发生的事情。

事实上还有另外一个层次上的，并会让你犯错的 inconsistency：如果你检查过，你会发现字符串函数与数组函数的参数位置相反。*php.net* 上的在线文档中，这些参数被称作 `$needle` 和 `$haystack`。比如说 `strstr` 函数的语法如下所示：

```
strstr ( string $haystack , mixed $needle [, bool $before_needle = false ] )
```

与此同时，`array_search` 函数的语法则如下所示：

```
array_search ( mixed $needle , array $haystack [, bool $strict ] )
```

让这类信息保持一致事实上有点麻烦。显然这些 PHP 的子系统是由不同开发者完成的，或者是同一个开发者，但他忘记自己做过什么了（请注意一个在函数名中使用了下划线，而另一个没有，这更容易导致困惑）。

为了使这一切保持有序，唯一的方法是记住：数组函数需要 `needle` 作为第一个参数，而字符串函数需要 `haystack` 做第一个参数，其中一个或两个函数的名字都可能用或者不用下划线。



这也是 PHP 认证有价值的地方，如果你能够通过认证考试，并很准确地记忆这样的信息，你就总是可以成为一个高薪工作的候选者。

## 松散类型

下一个我们希望进行考查的 PHP 的缺点是变量的数据类型声明。PHP 是松散类型的语言，这意味着你不必声明会存储在变量中的数据类型（整型、字符串、浮点或其他），PHP 将试图自己解决这些。与此不同的是有的语言被称为强类型语言，在这样的语言里变量在它建立的那一刻被告知会存储什么样的数据。在 PHP 代码中，你可以使用一个 `$notes` 变量，并分配一个字符串给它，然而接下来在代码中可以把整数赋值给它。尽管这可能会导致错误，但这不会影响 PHP 处理代码。

这就是问题所在：一旦一个变量被赋予类型，PHP 可以重新对其进行指定。这可能会在一部分开发者中导致不解，因为代码的含义有可能会被变化。这也可能让代码的调试和维护变得困难。

有的人则会强调其反面，说这是一种非常优雅的代码管理方式——让代码引擎来做这部分工作，开发者只关注最关键的部分（尽管它可能变得不容易维护）。因此，这也不能说一定是 PHP 的缺点，但确实是一些必须了解和注意的事情。

## 全局变量的注册

作为缺点讨论的最后一部分是真正的缺点，因为对它的使用会导致安全隐患。你可以在 `php.ini` 中打开和关闭 `register_globals` 选项。在近来的版本中（4.2.0 和以后的版本）这一选项默认是关闭的。你也可以在运行的 PHP 中通过 `ini_set` 函数管理这一设置。

如果不会导致安全漏洞 `register_globals` 对于节省时间非常有帮助，我认为它应该有更广泛的用途。它基于提交的 HTML 表单在内存里建立变量，因此，如果你有一个需要数据输入的表单，这一表单里有 `lastname` 和 `firstname` 这样的项目（如果 `register_globals` 被打开的话），当表单被提交时，在后续页的处理中，`$lastname` 和 `$firstname` 变量会被自动创建，同时输入的数据会被自动赋值给它们。

安全隐患是自此 PHP 脚本将对数据注入开放。比如说一个用 GET 提交的表单中有一个用 `lname` 表示 `lastname` 的输入，有的人就可以通过 URL 地址向这个域注入值。注入的东西可能是坏的数据、有危害的 JavaScript 代码，甚至是一些想不到的 SQL 命令。

如果你正在使用 4.2.0 之前的版本，确保你关闭了这个选项（如果你有在服务器层次做这件事情的权限），或者用 `ini_set` 函数关闭它。如果你不能关闭它，一定要避免使用它。



`Register_globals` 是一个过时的选项，在下一个大版本 (6.0) 中它将会从 PHP 中彻底消失。当前它存在的理由是向后兼容。

## 这就是全部吗

开发社区中的一些人可能认为 PHP 的另外一些地方也有缺点。但就像我之前声明的，这是视角和经验的问题。PHP 变得越来越强大和流行，用途也日益广泛，它必然会变得越来越好。

请记住 PHP 是一个开源的编程语言，它的改善来自于用户社区的贡献。如果你希望帮助去除 PHP 坏的地方，请加入 <http://www.php.net>。

## Symbols (符号)

& 引用变量 10, 30

<? php 文本序列, 4

\ (反斜杠)

用于转义字符, 34

用于命名空间标识符, 122

从输出中移除转义字符, 113-115

从字符串中移除, 41

[]

用于数组引用, 46, 48

正则表达式中用, 135

{ }

用于代码块, 27

for defining namespaces, 120

\$ 用于变量名, 9

\$\_ 超级全局实体前缀, 21

() 用于函数, 27

|| (OR) 条件测试, 16

++ 命令, 19

# 用于内联注释, 8

' (单引号) 用于字符串, 34

在数组的键中, 47

" (双引号) 用于字符串, 34

在数组的键中, 47

/\* ... \*/ for multiline comments, 8

/\* ... \*/ 用于多行注释, 8

// 用于内联注释, 8

## A

a+ option (file management functions) (a+ 选项 (文件管理函数)), 84

accessor methods (存储器方法), 69-70

ActiveState Komodo IDE (ActiveState Komodo 集成开发环境), 139

Add method (PieGraph class) (Add 方法 (PieGraph 类)), 105

adding elements to arrays (向数组添加元素), 48

AddLink method (FPDF) (AddLink 方法 (FPDF)), 99

addresses of SMS domains (SMD 域的地址), 91

addslashes function (Addslasher 函数), 41, 115

AliasNbPages method (FPDF) (AliasNbPages 方法 (FPDF)), 97

anonymous functions(closures) (匿名函数 (闭包)), 122

antispam graphics, generating (反垃圾图形

†: 索引所列页码为英文版页码, 请参照用“☞”表示的原书页码。



生成), 109

array function (数组函数), 46

array functions (数组函数), 51-57, 146

- math-type functions (数学型函数), 53
- sorting array elements (给数组元素排序), 51-53
- randomly (随机), 54

array\_merge function (array\_merge 函数), 56

array\_rand function (array\_rand 函数), 54

array\_search function (array\_search 函数), 54

array\_splice function (array\_splice 函数), 49

array\_sum function (array\_sum 函数), 54

array\_unique function (array\_unique 函数), 54

array\_walk function (array\_walk 函数), 57

arrays (数组), 45-57

- associative arrays (关联数组), 46
- for data validation (用于数据验证), 112
- dynamic (动态), 48-50
- indexed arrays (索引数组), 45
- multidimensional (多维), 47
- reading files into(读取文件), 86
- traversing (遍历), 50, 57

arsort function (arsort 函数), 51

asort function (asort 函数), 51

assigning values to function parameters (为函数参数赋值), 30

assigning values to variables (为变量赋值), 10

assignment expression (赋值表达式), 13

associative arrays (关联数组), 46

- merging (合并), 56

AUTO\_INCREMENT option (SQLite)

(AUTO\_INCREMENT 选项 (SQLite)), 78

averaging array values (数组平均值) 如果您有任何关于提高索引的建议, 请发邮件到 [index@oreilly.com](mailto:index@oreilly.com)

## B

backslash\ (反斜杠)

- for escaping characters (用于转义字符), 34
- for namespace identification (用于命名空间标识符), 122
- removing escapes from output (从输出中移除转义字符), 113-115
- stripping from strings (从字符串中移除), 41

bar charts, generating (条形图生成), 107-108

break statements (Break 语句), 17

browser tabs (浏览器标签), 22

BuildTable merhod(FPDF) (BuildTable 方法 (FPDF)), 102-104

built-in functions (内建函数), 32

by-reference assignment (引用赋值), 10, 30

by-value assignment (通过值进行赋值), 10, 30

## C

callback functions, unnamed (不命名回调函数), 123

calling functions (函数调用), 27

capitalization of strings, functions for (转大写字符串函数), 38

captchas, generating (验证码生成), 109

case management (strings) (字符串大小写管理), 38

cell, document (FPDF) (文档单元格 (FPDF)), 93

cell method (FPDF) (cell 方法), 93, 104

character case management (字符大小写管理), 38

characters, escaping (字符, 转义), 34

- removing escapes from output (从输出中移除转义字符), 113-115
- stripping backslashes from strings (从字符串中移除反斜杠), 41

classes 类, 59

- creating objects from (从 ... 创建对象), 65
- inheritance (继承), 60
- namespaces (命名空间), 119-122

closures (anonymous functions) (闭包 (匿名函数)), 122

comment lines (注释行), 8

community server (社区服务器版), 71

compact function (Compact 函数), 55

compound data types (复合数据类型), 10, 45

concatenating arrays (串联数组), 56

condition testing (see flow control)

- conditional return statements (条件测试 (见流控制))

条件返回语句), 28

constants (常量), 11-13

\_\_construct method (\_\_construct 方法), 65

constructor methods (constructor 方法), 65

\$\_COOKIE superglobal (\$\_COOKIE 超级全局实体), 21, 111

cookies (cookies), 20, 111

count function (Count 函数), 54

counting array elements (数组元素计数), 54

cross-site scripting (XSS) (跨站脚本 (XSS)), 115-116

## D

data encryption (数据加密), 116-117

data management using files (数据管理用文件), 79-87

data types (数据类型), 9

- of array elements (数组元素), 47
- loose typing (松散类型), 147
- in SQLite (在 SQLite 里), 78

data validation (数据验证), 111-113

- in set methods (在 set 方法里), 70

database interaction (数据库交互), 71-87

- escaping data for (用于 ... 的转义数据), 114
- file management as alternative to (用文件管理替代), 79-87

MySQLi object interface (MySQLi 对象接口), 71-74

PHP Data Objects (PDO) (PHP 数据对象 (PDO)), 74-77

SQLite database tool (SQLite 数据库工具), 77-79

date and time functions (日期和时间函数), 126-131

DateInterval class (DateInterval 类), 129

DateTime class (DateTime 类), 126-131

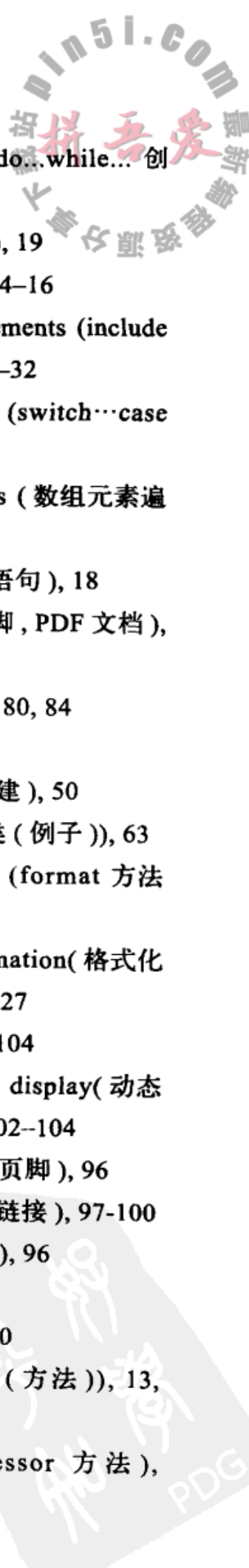
DateTimeZone class (DateTimeZone 类), 126-131

day format(DateTime) (日期格式(DateTime)), 127, 129

decision-making constructs (see flow control) default DateTime information (形成判断 (见流控制) 默认的 DateTime 信息), 126  
default function parameters (默认的函数参数), 29, 68  
define function (定义函数), 11  
defined constants (已定义常量), 11-13  
defining functions (正在定义的函数), 27  
deleting elements from arrays (从数组中删除元素), 49  
\_destruct method (\_\_destruct 方法), 66  
destructor methods (destructor 方法), 66  
development environments PHP (开发环境 PHP), 138-140  
    setting up (确立), 3  
DevZone website (DevZone 站点), 141  
diff method (DateTime) (Diff 方法 (DateTime)), 129  
difference between dates (日期间的偏差), 129  
directories, creating (目录, 创建), 82  
do...while... construct (Do...while... 创建), 18  
document cell (FPDF) (文档单元格 (FPDF)), 93  
documents, PDF(see FPDF Library)  
    documents, XML (see SimpleXML library) domains, SMS (文档, PDF(见 PDF Library))  
文档, XML (见 SimpleXML 库)  
域), SMS, 91  
double quotes (") for strings (字符串中的双引号), 34

in array keys (在数组的键里), 47  
dynamic arrays (动态数组), 48-50  
dynamic PDFs (动态 PDF), 102-104  
**E**  
echo command (回显命令), 4, 34  
Eclipse platform, Zend studio for (Eclipse 平台, 支持 ... 的 Zend Studio), 139  
editing strings (编辑字符串), 40-43  
elements, array (元素, 数组), 45  
    adding to arrays (向数组添加), 48  
    counting number of (计算 ... 的数目), 54  
    data types of(... 的数据类型), 47  
    extracting as variables(作为变量提取), 55  
    extracting variables into(把变量提取进 ...), 55  
    referencing(引用), 46, 47  
    removing from arrays(从数组删除), 49  
    sorting(排序), 51-53  
        randomly(随机地), 54  
    summing values of(为 ... 求和), 54  
    testing uniqueness of(测试 ... 的唯一性), 54  
traversing(遍历), 50, 57  
else clause(else 子句), 14  
elseif generation(elseif 子句), 15  
email generation(电子邮件生成), 89-92  
empty array, creating(空数组, 创建), 46  
encapsulation(封装), 60, 68  
encrypting passwords(加密的密码), 116-117  
endless looping(无限循环), 125  
entities, HTML(实体, HTML), 41





escaping sharacters with backslash( 带反斜杠的转义字符 ), 34  
removing escapes from output( 从输出中移除转义字符 ), 113–115  
stripping backslashes from string( 从字符串中去除反斜杠 ), 41  
expressions( 表达式 ), 13  
extension=statement (php.ini) (Extension=语句 (php.ini 中) ), 74  
extract function( 提取函数 ), 55  
EXTR\_SKIP parameter (extract function) (EXTR\_SKIP 参数 (提取函数) ), 56

## F

fclose function (fclose 函数 ), 80  
file\_exists function (file\_exists 函数 ), 80  
file function (file 函数 ), 86  
files ( 文件 )  
    data management with ( 用 ... 做数据管理 ), 79–87  
    determining size of ( 决定 ... 的大小 ), 84  
    including or requiring ( 包含或要求 ), 31–32  
    PDF (see FPDF library) XML (see simpleXML library) filesize function (PDF ( 见 FPDF 库 ) ( 见 SimpleXML 库 )  
filesize 函数 ), 80, 84  
filtering input (see input data validation)  
    flock function( ( 过滤输入 ( 见输入数据验证 )  
flock 函数 ), 80, 84  
flow control ( 流控制 ), 13–19  
    conditional return statements ( 条件返

回语句 ), 28  
do...while...constructs (do...while... 创建 ), 18  
for statements (for 语句 ), 19  
if statements (if 语句 ), 14–16  
include and require statements (include 和 require 语句 ), 31–32  
switch...case statements (switch...case 语句 ), 16–18  
traversing array elements ( 数组元素遍历 ), 50  
while statements (while 语句 ), 18  
footers, PDF documents ( 页脚, PDF 文档 ), 96  
fopen function (fopen 函数 ), 80, 84  
for statements (for 语句 ), 19  
foreach construct (foreach 创建 ), 50  
from class (example) (form 类 ( 例子 ) ), 63  
format method (DateTime) (format 方法 (DateTime)), 127  
formatting Date Time information( 格式化 DateTime 的信息 ), 127  
FPDF library(FPDF 库 ), 92–104  
    dynamic PDFs and table display( 动态 PDF 和表格显示 ), 102–104  
    headers and footers( 页眉页脚 ), 96  
    images and links( 图像和链接 ), 97–100  
    layout options( 布局选项 ), 96  
    watermarks ( 水印 ), 101  
fread function (fread 函数 ), 80  
functions (methods) ( 函数 ( 方法 ) ), 13, 27–32  
    accessor methods (accessor 方法 ), 69–70



anonymous (closures) (匿名函数 (闭包)), 122  
array functions (数组函数), 51–57, 146  
    math-type functions (数学型函数), 53  
    sorting array elements (给数组元素排序), 51–53, 54  
built-in versus user-defined (内建与用户定义), 32  
default parameters (默认参数), 29, 68  
file managements (文件管理), 80  
names for (为 ... 命名), 36, 146  
in object-oriented programming (在 ... 面向对象编程里), 59  
passing parameters (参数传递), 27–29, 146  
    by value versus by reference (传值与传引用), 30  
string functions (字符串函数), 36–43, 40–43, 146  
    character case management (字符大小写管理), 38  
    searching string content (搜索字符串的内容), 39–40  
    string trimming (去除字符串的空格), 36  
fwrite function (fwrite 函数), 80, 84

## G

get\_ini function (get\_ini 函数), 130  
GET method(HTTP) (GET 方法(HTTP)), 23, 24  
get methods (get 方法), 69–70  
\$\_GET superglobal (\$\_GET 超级全局实体), 22, 111  
getLocation method (DateTimeZone)

(getLocation 方法(DateTimeZone)), 130

global namespace (global 命名空间), 120  
goto statement (goto 语句), 124–126, 145  
graphical reports generation (图形化报告的生成), 105–109

## H

headers, PDF documents (头, PDF 文档), 96  
Hello World program (Hello World 程序), 4  
HEREDOC constructs (HEREDOC 的创建), 35, 123  
histograms, generating (直方图生成), 107–108  
history of PHP (PHP 的历史), 1  
hour format(DateTime) (小时的格式(DateTime)), 129  
html class (example) (Html 类(例子)), 60  
HTML entities (HTML 实体), 41  
html\_entity\_decode function (html\_entity\_decode 函数), 41  
HTML tage, stripping from strings (HTML 标签, 从字符串中删除), 40  
htmlentities function (htmlentities 函数), 41, 114  
htmlspecialchars function (htmlspecialchars 函数), 113  
HTTP GET method (HTTP GET 方法), 23, 24  
HTTP POST method (HTTP POST 方法), 23, 24  
hyperlinks in PDF documents (PDF 文档中的超级链接), 97–100

## I

IDEs for PHP programming (PHP 编程的集成开发环境), 138-140

if statements (if 语句), 14-16

Image method(FPDF) (Image 方法 (FPDF)), 98

images in PDF documents (PDF 文档中的图像), 97-100

in\_array function (in\_array 函数), 54

includable files (可包含的文件), 65

include\_once statement (include\_once 语句), 32

include statement (include 语句), 31-32

indexed arrays (索引后的数组), 45

    merging (合并), 56

inheritance (继承), 60

.ini file (see php .ini settings file) ini\_set function (.ini 文件 (见 php.ini 设置文件))

ini\_set 函数), 147

injection attacks (注入攻击), 115-116, 147

inline comments (内联注释), 8

input data validation (输入数据验证), 111-113

in set methods (在 set 方法里), 70

installing PHP (安装 PHP), 3

installing PHPMailer library (安装 PHPMailer library), 90

instantiation (实例化), 65

integrated development environments (集成开发环境), 138-140

integration with web pages (和网页集成), 19-25

    cookies (cookies), 20, 111

    \$\_GET superglobal (\$\_GET 超级全局

实体), 22, 111

\$\_POST superglobal (\$\_POST 超级全局实体), 23, 111

\$\_REQUEST superglobal (\$\_REQUEST 超级全局实体), 24

sessions (会话), 21, 111

internal links, PDF documents (内部链接, PDF 文档), 99

interpolative characteristics of double quotes (双引号的插值特征), 34, 35

is\_int function (is\_int 函数), 113

is\_numeric function (is\_numeric 函数), 113

is\_readable function (is\_readable 函数), 86

is\_writable function (is\_writable 函数), 86

## J

JPGraph library (JPGraph 库), 105-109

jumping within code files (see goto statement) (在代码文件中跳转 (见 goto 语句))

## K

key/value pairs (键 / 值对), 45

keys, array naming (键, 数组命名), 46

    numerical (indexed arrays) (数值 (索引后的数组)), 45

    selecting randomly (随机选择), 54

strings for (用于 ... 的字符串), 46

Komodo IDE (ActiveState) (Komodo 集成开发环境 (ActiveState)), 139

krsort function (krsort 函数), 51

ksort function (ksort 函数), 51

## L

latitude information (维度信息), 130

layout options, PDF files ( 布局选项, PDF 文件 ), 96

lcfirst function (lcfirst 函数), 38

length of strings, returning ( 字符串长度, 返回 ), 39

Lerdorf, Rasmus (Lerdorf, Rasmus), 1

libraries, PHP FPDF library ( 库, PHP FPDF 库 ), 92–104

JPGGraph library (JPGGraph 库), 105–109

PHPMailer library (PHPMailer 库), 89–92

SimpleXML library (SimpleXML 库), 136–138

links in PDF documents (PDF 文档中的链接), 97–100

locking files ( 锁定文件 ), 84

longitude information ( 精度信息 ), 130

looping, endless ( 无限循环 ), 125

loose typing ( 松散类型 ), 147

lowercase in strings, functions for ( 字符串种的小写字符, 用于 ... 的函数 ), 38

ltrim function (ltrim 函数), 36

## M

magic methods (magic 方法), 65

mail function (mail 函数), 89–92

matching strings with legular expressions ( 用正则表达式匹配字符串 ), 133–134

math-type array functions (math-type array 函数), 53

max\_execution\_time directive (max\_execution\_time 指令), 145

MD5 encryption algorithm (MD5 加密算法), 117

MD5 function (MD5 函数), 42

merging arrays ( 合并数组 ), 56

messaging( 消息 ), 89–92

methods (see functions) minute format(DateTime)( 方法 ( 见函数 ) 分钟的格式 (DateTime)), 129

mkdir functions(mkdir 函数), 80, 82

modifying strings( 字符串调整 ), 40–43

month format(DateTime)( 月份格式 (DateTime)), 127, 129

multidimensional( 多维数组 ), 47

multiline comments( 多行注释 ), 8

multi\_query method(MySQLi)(multi\_query 方法 (MySQLi)), 74, 79

MySQL database(MySQL 数据库), 71

mysql\_real\_escape\_string function (mysql\_real\_escape\_string 函数), 114

MySQLi object interface (MySQLi 对象接口), 71–74

## N

names ( 命名 )

array keys ( 数组的键 ), 46

conflicts, avoiding (see namespaces) functions ( 冲突, 避免 ( 见命名空间 ) 函数 ), 36, 146

functions without (see anonymous functions) ploymorphism ( 没有 ... 的函数 ( 见匿名函数 ) 多态 ), 59

variables ( 变量 ), 9

namespaces ( 命名空间 ), 119–122

nesting if statements ( 嵌套的 if 语句 ), 15

new keyword ( 关键字 ) new, 65



new line control option (FPDF) Cell method  
(换行控制选项 (FPDF Cell 方法)),  
94

now (Date Time value) (now (DateTime  
值)), 128

NOWDOC constructs (NOWDOC 构建),  
123

numerical arrays keys (数值数组的键), 45

NuSphere PhpED IDE (NuSphere PhpED 集  
成开发环境), 140

## O

object-oriented programming (面向对象编  
程), 59-70

objects (对象), 59

creating (创建), 65

namespaces (命名), 119-122

one-way encryption (单向加密), 116

OOP (see object-oriented programming)

open namespaces (OOP (见面向对  
象编程))

打开命名空间), 120

opening files (打开文件), 84

OR (||)condition test (OR (||) 条件测试), 16

organizing array elements (组织数组元素),  
51-53

randomly (随机的), 54

output, PDF (see FPDF library) (输出,  
PDF(见 PDF 库))

## P

page layout, PDF (页面布局, PDF), 96

PageNo method(FPDF) (PageNo 方法  
(FPDF)), 97

parameters, function (see passing parameters to

functions) parent classes (参数, 函数  
(见为函数传递参数)父类), 60

passing parameters to functions (为函数传  
递参数), 27-29, 146

default values (默认值), 29, 68

by value versus by reference (传值或传  
引用), 30

password encryption (密码加密), 116-117

password, string functions for (密码, 用  
于 ... 的字符串函数), 42

PDF generation (see FPDF library) PDO  
(see PHP Date Objects) phone  
number, finding (example) (PDF 生  
成(见 FPDF 库))

PDO (见 PHP 数据对象)

电话号码, 查找(例子)), 134

PHP5.3 goto statement (PHP 5.3  
goto 语句), 145

PHP, defined (PHP, 定义的), 2

PHP, history of (PHP, ... 的历史), 1

PHP, where used (example) (PHP, 在那里使  
用(例子)), 2

PHP 5.3 (PHP 5.3), 119-131

closures (anonymous functions) (闭包  
(匿名函数)), 122

date and time functions (日期和时间函  
数), 126-131

goto statement (goto 语句), 124-126

namespaces (命名空间), 119-122

NOWDOC constructs (NOWDOC 的构  
建), 123

php|architect website (php| 网站架构), 143

PHP configuration (PHP 的配置), 19-25

cookies (cookies), 20, 111



- `$_GET` superglobal (`$_GET` 超级全局实体), 22, 111
  - `$_POST` superglobal (`$_POST` 超级全局实体), 23, 111
  - `$_REQUEST` superglobal (`$_REQUEST` 超级全局实体), 24
  - sessions (会话), 21, 111
  - PHP Data Objects (PDO) (PHP 数据对象 (PDO)), 74–77
  - prepared statements (准备好的语句), 75
  - PHP development (PHP 开发环境), 138–140
  - php.ini settings file (php.ini 设置文件), 4, 22
  - extension= statements (extension= 语句), 74
  - extracting timezone data from (从 ... 提取时区信息), 126
  - security-related settings (安全相关设置), 117
  - php.net website (php.net 网站), 140
  - PHP setup (PHP 安装), 3
  - php\_strip\_whitespace function (php\_strip\_whitespace 函数), 7
  - PHP web conferences (PHP web 会议), 143
  - PHP websites (PHP 网站), 140
  - phparch.com website (phparch.com 网站), 143
  - PhpED IDE (NuSphere) (PhpED 集成开发环境 (NuSphere)), 140
  - phpinfo.php file (phpinfo.php 文件), 4
  - PHPMailer library (PHPMailer 库), 89–92
  - phpmailer.php file (phpmailer.php 文件), 90
  - pie charts, generating (饼图, 生成), 105–107
  - polymorphism (多态), 59
  - POST method (HTTP) (POST 方法 (HTTP)), 23, 24
  - `$_POST` superglobal (`$_POST` 超级全局实体), 23, 111
  - precedence, included or requirde files (优先权, 包含或者必须), 32
  - preg\_match function (preg\_match 函数), 133
  - preg\_replace function (preg\_replace 函数), 135
  - preg\_split function (preg\_split 函数), 135
  - prepared statements(PDO) (准备好的语句 (PDO)), 75
  - primitive types (原始类型), 10
  - print command (打印命令), 34
  - private scope (私有域), 68
  - programming resources (编程资源), 140
  - properties (属性), 59
    - get and set methods (get 和 set 方法), 69–70
    - scope of (... 的域), 68
  - protected scope (保护域), 68
  - public scope (公有域), 68
- ## Q
- queryexec method (queryexec 方法), 79
  - quote function (quote 函数), 115
  - quotes for strings (字符串的引号), 34
    - in array keys (在数组的键里), 47
- ## R
- random extraction of array value (数组中值

的随机提取), 54

random sorting of array values (数组中值的排序), 54

randomizing string content (随机处理字符串的内容), 42

real\_escape\_string function (real\_escape\_string 函数), 115

receiving parameters (see default function parameters) reference approach to parameter assignment (收到的参数 (见默认函数参数))

引用方式为参数赋值), 30

reference approach to variable assignment (引用方式为变量赋值), 10

references, website on PHP (引用, PHP 的网站), 140

referencing array elements (引用数组元素), 46, 47

(遍历数组元素), 50, 57

register\_globals directive (register\_globals 指令), 118, 147

regular expressions (正则表达式), 133–136

splitting strings (切分字符串), 135

string matching (字符串匹配), 133–134

substituting strings (替换字符串), 135

removing elements from arrays (移除数组中的元素), 49

replacing strings (替换字符串), 40

reports, graphical (报告, 图形化), 105–109

\$\_REQUEST superglobal (\$\_REQUEST 超级全局实体), 24

require\_once statement (require 语句), 31–32

resources for further reading (进一步阅读的资源), 140

retrieving database data for display (获取数据库中的数据来显示), 73

return values (see functions) (返回值 (见函数))

rsort function (rsort 函数), 51

rtrim function (rtrim 函数), 36

## S

salt-and-pepper encryption (调味料式加密), 117

scalar data types (标量数据类型), 10, 45

scope 域

defined constants (定义的常量), 11

of properties (属性的), 68

variables (变量), 11

scripting attacks (脚本攻击), 115–116

searching arrays (搜索数组), 54

searching strings (搜索字符串), 39–40

second format(DateTime) (秒的格式 (DateTime)), 129

security (安全), 111–118

cross-site scripting and SQL injection (跨站点脚本和 SQL 注入), 115–116

input data validation (输入数据的验证), 70, 111–113

output validation (输出验证), 113–115

password encryption (密码加密), 116–117

register\_globals directive (register\_globals 指令), 118, 147

server value for timezone (服务器上的时区值), 126

- session.save\_path directive (session.save\_path 指令), 22
- session\_start function (session\_start 函数), 22
- \$\_SESSION superglobal (\$\_SESSION 超级全局实体), 22, 111
- sessions (会话), 21, 111
- set methods (set 方法), 69–70
- setcookie function (setcookie 函数), 21
- setLink method (FPDF) (SetLink 方法 (FPDF)), 99
- SetX method (FPDF) (SetX 方法 (FPDF)), 94, 101
- SetY method (FPDF) (SetY 方法 (FPDF)), 101
- sha1 encryption algorithm (sha1 加密算法), 116
- shuffle function (shuffle 函数), 54
- SimpleXML library (SimpleXML 库), 136–138
- simplexml\_import\_dom function (simplexml\_import\_dom 函数), 138
- simplexml\_load\_file function (simplexml\_load\_file 函数), 138
- simplexml\_load\_string function (simplexml\_load\_string 函数), 138
- single quotes ( ' )for strings ( 字符串中的单引号 ( ' )), 34
  - in array keys ( 在数组的键之中 ), 47
- size of file, determining ( 文件的大小, 决定 ), 84
- size of strings, returning ( 字符串的大小, 返回 ), 39
- SMS generation (SMS 生成), 89–92
- sort function (sort 函数), 51
- sorting array elements ( 排序数组中的元素 ), 51–53
  - randomly ( 随机地 ), 54
- spaces (see whitespace) 空格 ( 见空白 )
- spaghetti code ( 意大利面条式代码 ), 125
- spam-prevention graphice, generating ( 垃圾阻止的图形, 生成 ), 109
- special characters, removing for output ( 特别的字符, 从输出中移除 ), 113–115
- special data types ( 特别的数据类型 ), 10
- splitting strings with regular expressions ( 用正则表达式切分字符串 ), 135
- SQL (see database interaction) SQL injection (SQL ( 见数据库交互 ) SQL 注入 ), 115–116, 147
- SQLite database tool (SQLite 数据库工具), 77–79
- stateless, websites as ( 无状态的网站 ), 19
- statements ( 语句 ), 9
- str\_replace function (str\_replace 函数), 40
- str\_shuffle function (str\_shuffle 函数), 42
- str\_word\_count (str\_word\_count), 39
- string functions ( 字符串函数 ), 36–43, 40–43, 146
  - character case management ( 字符大小写管理 ), 38
  - searching string content ( 搜索字符串内容 ), 39–40
  - string trimming ( 字符串去除空格 ), 36
- strings ( 字符串 ), 33–43



for array keys ( 用于数组键 ), 46  
defined ( 定义的 ), 33  
quotes ( 用于 ... 的引号 ), 34  
in array keys ( 在数组键里 ), 47  
regular expressions for ( 用于 ... 正则表达式 ), 135  
for string splitting ( 用于字符串切分 ), 135  
for string substitution ( 用于字符串替换 ), 135  
regular expressions with for matching strings ( 和 ... 一起的正则表达式用于匹配字符串 ), 133-134  
saving URLs as ( 把 URL 保存为 ... ), 99  
strip\_tags function (strip\_tags 函数 ), 40  
stripping HTML from strings ( 从字符串种去除 HTML ), 40  
stripslashes function (stripslashes 函数 ), 41  
stristr function (stristr 函数 ), 39  
strlen function (strlen 函数 ), 39  
stroke method(PieGraph class) (Stroke 方法 (PieGraph 类 )), 105  
strpos function (strpos 函数 ), 40  
strstr function (strstr 函数 ), 39  
strtolower function (strtolower 函数 ), 38  
strtoupper function (strtoupper 函数 ), 38  
substituting strings( 替换字符串 ), 40  
substr function (substr 函数 ), 40  
syvstrubg searches( 子字符串搜索 ), 39-40  
summing array values( 求和数组值 ), 54  
superclasses( 超类 ), 60  
superglobals( 超级全局实体 ), 21-25  
switch...case statements(switch...case 语

句 ), 16-18  
syntax, PHP( 语法, PHP ), 7

## T

table class (example) (table 类 (例子 )), 62  
table display, PDF documents( 表格显示, PDF 文档 ), 102-104  
tabs, web browsers( 标签页, 网页浏览器 ), 22  
tags, stripping from strings( 标签, 从字符串中删除 ), 40  
telephone numbers, finding (example) ( 电话号码, 查找 (例子 )), 134  
text messaging( 文本消息 ), 89-92  
\$this variable(\$this 变量 ), 9, 66  
time and date functions( 时间和日期函数 ), 126-131  
time format(DateTime) ( 时间格式 (DateTime) ), 128  
timezone format(DateTime) ( 时区格式 (DateTime) ), 128  
timezone management( 时区管理 ), 126  
totaling array values( 所有数组值的和 ), 54  
transaction-based database programming( 基于事务的数据库编程 ), 77  
traversing arrays( 遍历数组 ), 50, 57  
trim function(trim 函数 ), 37  
trimming strings( 去除字符串的空格 ), 36  
\$trusted array (example) (\$trusted 数组 (例子 )), 112  
two-dimensional( 二维数组 ), 47

## U

ucfirst function (ucfirst 函数 ), 38, 113  
ucwords function (ucwords 函数 ), 38, 113



uniqueness of array elements, testing ( 数组元素的唯一性, 测试 ), 54  
unlocking files ( 解锁文件 ), 84  
unnamed functions (unnamed 函数 ), 122  
unset function (unset 函数 ), 50  
uppercase in strings, functions for ( 字符串的大写字符, 用于 ... 的函数 ), 38  
URLs, saving as strings (URLs, 保存为字符串 ), 99  
user-defined function (UDFs) ( 用户自定义函数 (UDFs)), 32  
usort function (usort 函数 ), 53

## V

validation ( 验证 ), 111–113  
    in set methods ( 在 set 方法中 ), 70  
value approach to parameter assignment ( 值方式参数赋值 ), 30  
value approach to variable assignment ( 值方式变量赋值 ), 10  
var\_dump function (var\_dump 函数 ), 37  
variable expansion ( 变量扩展 ), 34  
variables ( 变量 ), 9–11  
    assigning values to ( 赋值给 ), 10  
    converting array elements into ( 把数组元素转换为 ... ), 55  
    extracting array elements ( 把数组元素提取为 ... ), 55  
    loose typing ( 松散类型 ), 147  
    saving URLs as strings ( 把 URL 保存为字符串 ), 99  
    scope ( 域 ), 11  
    superglobals ( 超级全局实体 ), 21–25  
variables\_order directive (variables\_order

指令 ), 25

## W

w+ option (file management function) (w+ 选项 ( 文件管理函数 )), 84  
walking through arrays ( 数组遍历 ), 50, 57  
watermarks, PDF documents ( 水印, PDF 文档 ), 101  
web browser tabs ( 浏览器标签页 ), 22  
web conferences, PHP ( 网络会议, PHP ), 143  
web pages, integration with ( 网页, 与 ... 集成 ), 19–25  
    cookies (cookies), 20, 111  
    \$\_GET superglobal ( \$\_GET 超级全局实体 ), 22, 111  
    \$\_POST superglobal ( \$\_POST 超级全局实体 ), 23, 111  
    \$\_REQUEST superglobe ( \$\_REQUEST 超级全局实体 ), 24  
    sessions ( 会话 ), 21, 111  
websiter for further reading( 进一步阅读的网站 ), 140  
week format (DateTime)( 周 的 格式 (DateTime)), 127, 129  
while statements(while 语句 ), 18  
whitespace in PHP code( 空格在 PHP 代码中 ), 7  
    trimming from strings( 从字符串种删除 ), 36  
write method (FPDF)(write 方法 (FPDF)), 100  
writing to files( 写文件 ), 84

## X

XAMPP package(XAMPP package), 3

XML documents, consuming (SimpleXML library) (XML 文档, 使用中的 (见 SimpleXML 库))

## Y

year format(DateTime)( 年 的 格 式 (DateTime)), 128, 129

## Z

Zend Corporation(Zend 公司的网站), 141

Zend Studio for Eclipse(Eclipse 版的 Zend Studio), 139



## 关于作者

彼得 B. 麦金太尔在信息技术产业主要是在软件开发领域已有超过 20 年的经验。他的基本技术非常广，包括几个 Web 开发语言、客户机 / 服务器工具及关系数据库系统，还包括 PHP、MySQL、PowerBuilder、Visual Basic、Active Server Pages，以及 CA 可视化对象。他是一个 Zend 认证工程师，他通过 PHP 4.x 的认证考试，在加拿大大西洋省区他是第一个获得这个认证的，对此他感到十分自豪。

多年来，彼得使用 SQL 与 Sybase PowerBuilder 开发了几个大型系统，与此同时也用 Clipper 语言为爱德华王子岛政府编写几个 X 基系统。他在数据建模 / 架构，包括逻辑和物理数据库设计方面也有相当丰富的专业知识。

彼得为四个 IT 行业的出版物贡献过内容：Using Visual Objects(Que Corporation)、Using PowerBuilder 5 (Que Corporation)、ASP.NET Bible (Wiley Press) 和 The Web Warrior Guide to Web Programming (Thompson Course Technology)，并为其其他 8 个书籍提供了技术编辑服务。彼得是 Programming PHP (第 2 版) 的合著者 (O'Reilly 出版)，这本书已被翻译成三种欧洲语言。同时也是 Zend Studio for Eclipse Developer's Guide (Addison - Wesley 出版) 的合著者。他曾担任 PHP|Architect 杂志的特约编辑和作者。

彼得曾在北美和国际计算机会议发言，包括在美国新奥尔良 CA-world、在德国科隆的 CA-TechniCon 和在澳大利亚墨尔本 CA - 世博会。彼得在加拿大爱德华王子岛生活和工作，在那里他是 MJL 公司的项目经理 (首席系统架构师)，这是一家专门从事汽车和保险行业的网络和移动网络应用的企业。他业余还经营自己名为 Paladin Business Solutions 的软件公司，可以通过其网站联系他。



## 关于封面

PHP 的封面上的动物：The Good Parts 是一个长靴拍尾蜂鸟（*Ocreatus underwoodii*）。长靴拍尾是一个物种，顾名思义，有一些与众不同的特色：一个裂开的尾巴，有时被认为类似于一对网球拍，一双细长的柄和小头，腿上是柔和的白色羽毛，好像是穿着靴子。雌鸟可以运动胸前白色的羽毛。

长靴拍尾蜂鸟是南美洲的品种，可以发现于沿安第斯科迪勒拉区域的玻利维亚、厄瓜多尔、秘鲁和委内瑞拉的热带雨林。由于其相当广泛的栖息地，此鸟在南美洲西部被认为是比较常见的。尽管如此，长靴拍尾蜂鸟很受欢迎。观鸟者和摄影师访问该地区，可能是由于该物种的独特外观。

2004 年，来自美国加州大学伯克利分校和加州理工学院的研究人员开始了一项旨在发现为什么秘鲁蜂鸟大多停留在低海拔地区的研究，毕竟那里空气稀薄，食物竞争相对较少。这项研究也涵盖了长靴拍尾蜂鸟。毫不奇怪，研究人员指出，在高海拔地区，空气稀薄，这导致蜂鸟体力和机动能力降低，进而阻碍了它们茁壮成长。长靴拍尾蜂鸟的形象也出现在 1996 年厄瓜多尔的邮票上。封面图片是来自《卡塞尔的自然史》。